Efficient Task Scheduling and Resource Allocation using Osmotic Computing

A Thesis

submitted in partial fulfillment of the requirements for the award of the degree of

Doctor of Philosophy

by

Akashdeep Kaur

Registration No. 901803005

under the guidance of

Dr. Rajesh Kumar
Professor

Dr. Sharad Saxena
Associate Professor

Department of Computer Science and Engineering

Thapar Institute of Engineering and Technology

Patiala-147001, Punjab, India

September, 2024

Candidate Declaration

I hereby certify that the work, which is being presented in the thesis, entitled **"Efficient Task Scheduling and Resource Allocation using Osmotic Computing"**, in partial fulfilment of the requirements for the award of the degree of **Doctor of Philosophy** and submitted to Thapar Institute of Engineering and Technology is an authentic record of my own work carried out during the period July 2018 to September **2024** under the supervision of **Dr. Rajesh Kumar and Dr. Sharad Saxena.**

The matter presented in this thesis has not been submitted elsewhere for the award of any other degree or diploma from any institution.

Date: 24/9/24

Akashdeep Kaur

Candidate

This is to certify that the above statement made by the candidate is correct to the best of our knowledge.

Date: 24/9/24

Dr. Rajesh Kumar
Professor, DCSE
Supervisor

Dr. Sharad Saxena
Associate Professor, DCSE
Supervisor

To my Mamma, Daddy, and Sai.

ACKNOWLEDGEMENTS

First and foremost, I express my deep sense of gratitude to Waheguru ji for blessing me with the opportunity to pursue and complete the Ph.D.

I would like to express my deepest gratitude to my honorable supervisors, Dr. Rajesh Kumar and Dr. Sharad Saxena, whose guidance and support have been invaluable throughout my PhD journey.

To Dr. Rajesh Kumar, I am immensely thankful for your unwavering encouragement and sage advice at every step. Your academic insight, philosophical reflections, and motivational support have not only shaped my research but also inspired my personal growth. The values of honesty and hard work that you instilled in me have guided my work ethic throughout this journey. Your attention to detail and commitment to excellence have been instrumental in the quality and depth of my work.

To Dr. Sharad Saxena, I am profoundly grateful for your expertise and rigorous analytical approach. Your critical evaluations and thorough feedback pushed me to constantly refine and elevate my research. Your ability to scrutinize every detail with precision has had a lasting impact on the depth and rigor of my work.

I would like to extend my sincere thanks to Dr. Shalini Batra (Head, DCSE), Dr. Sushma Jain (Ph.D. Coordinator), and my Ph.D. committee members, Dr. Neeraj Kumar, Dr. A. K. Lal, and Dr. Anju Bala, for their guidance and support throughout my research. Their valuable insights and feedback have greatly contributed to the successful completion of this thesis.

I would also like to extend my heartfelt thanks to my parents Mr. Baldev Singh and Mrs. Joginder Kaur and brother Sarbjeet Singh for being the best support system I could ever ask for. I am truly blessed to have you in my life and I am forever grateful and indebted for your unwavering love and belief in me. Your encouragement has been my constant source of strength throughout this journey. I owe you more than I can ever repay.

A special thanks to my best friend Shabnam Bawa, whose companionship has been invaluable. Your friendship has been a constant source of joy and support. I am also grateful to my dear friends Paluck Arora, Sukhmanpreet Kaur, Sukhandeep Kaur, and Jasleen Kaur who have played a significant role in supporting me through this journey. Your presence has made this journey much more meaningful and memorable.

Together, all of you have set high standards for me to follow, and I am forever grateful for your mentorship, love, and support.

Akashdeep Kaur

ABSTRACT

The rapid proliferation of Internet of Things (IoT) devices, projected to reach 30.9 billion by 2025, has led to an unprecedented surge in data generation and a heightened demand for real-time, low-latency processing. As IoT applications become increasingly critical in sectors such as healthcare, transportation, and smart cities, traditional cloud-centric computing models face significant challenges in meeting the stringent requirements for timely data processing and service delivery. The inherent latency and bandwidth limitations of centralized cloud architectures create bottlenecks that hinder their effectiveness in handling the vast and dynamic data streams generated by IoT devices.

To address these challenges, there is a pressing need for computing paradigms that can adapt to the distributed nature of modern networks and efficiently manage the high volume of data generated. This work explores Osmotic Computing (OC), a novel paradigm designed to bridge the gap between edge and cloud computing by optimizing resource allocation and service migration across heterogeneous environments. OC leverages emerging technologies such as 5G and Mobile Edge Computing (MEC) to enhance the responsiveness and scalability of computing systems, enabling them to meet the demands of real-time applications. This work focuses on developing an efficient framework for task scheduling and resource allocation in 5G networks, Intelligent Transportation Systems (ITS), and IoT environments.

The first framework, OCTRA-5G, is developed to address the limitations of traditional task scheduling and resource allocation in 5G networks. Traditional models struggle to efficiently manage the increased complexity and demand of 5G environments. OCTRA-5G utilizes OC principles to segregate services into microservices and macroservices, optimizing their scheduling and migration. Through simulations on sets of 10, 20, and 30 gNBs, the framework demonstrated substantial improvements in performance using algorithms such as FCFS, SJF, and PS, effectively reducing latency and improving overall efficiency.

The second framework, OsCoMIT, is introduced to tackle the challenges in Intelligent Transportation Systems (ITS). With the rise in intelligent vehicles, there is a critical need for efficient

resource allocation at the edge network to handle service requests swiftly and effectively. OsCoMIT employs a Proportional Fairness (PF) algorithm to manage computational and memory resources for these vehicles. This framework improves upon traditional algorithms like FCFS and PS by enhancing resource utilization and system performance. The results of the framework are validated through statistical analysis using ANOVA which shows that OsCOMIT performed better than other algorithms.

The third framework, DQN-Osmosis, addresses the dynamic nature of IoT, edge, and cloud environments by introducing Deep Q-Networks (DQN) for intelligent decision-making. In rapidly changing network conditions, traditional decision-making approaches may not adapt quickly enough. DQN-Osmosis uses reinforcement learning to optimize service migration and task offloading, significantly improving resource allocation and processing efficiency compared to Random Agent, Q-Learning, and SARSA. The effectiveness of this approach was confirmed through the Wilcoxon signed-rank test.

The fourth framework, $\mu - osmotic$, a novel approach designed to address the challenges of dynamic resource allocation in Intelligent Transportation System. As these devices generate vast amounts of data, the need for efficient computational paradigms at the network edge becomes critical. The proposed $\mu-osmotic$ framework leverages Osmotic Computing principles and the Advantage Actor-Critic (A2C) algorithm to optimize the distribution of service requests between edge and cloud resources. By dynamically managing resources on the basis of real-time values of the metrics such as CPU usage, memory consumption, and energy efficiency, $\mu-osmotic$ enhances service performance, reduces latency, and ensures effective resource utilization. Through comprehensive evaluation and comparison with other algorithms, the framework demonstrates significant improvements.

Contents

List of Tables

List of Figures

List of Algorithms

Chapter 1

Introduction

In this chapter, the focus is to emphasize the emergence of computing paradigms that are driven by the exponential increase in the production of data from IoT devices and in the surge of demand for low-latency as well as scalable applications. The growth projected by the markets highlights the immediate need for adaptable and efficient computing models. The IoT devices are expected to reach 30.9 billion by 2025. The traditional cloud-centric approaches thus feel the challenge in meeting the low latency requirements of the time-sensitive applications. This chapter sets the stage by discussing the core concepts of cloud, edge, and fog computing and highlights their limitations in traditional IoT environments. It introduces Osmotic Computing that aims to optimize the service deployment and resource allocation across heterogeneous environments. The chapter also offers insights into the principles, their advantages, and potential applications across numerous sectors.

1.1 Cloud, Edge, and Fog Computing: Key Concepts

The global market for cloud computing is expected to reach $1.3 trillion by 2027 [5], highlighting the success of cloud computing but also emphasizing the need for more efficient and scalable solutions. Based on the IoT analytics issued in 2024, it is expected that the number of connected IoT devices will reach 30.9 billion by 2025 [6]. This growth necessitates a computing model to handle the diverse and massive volume of data and network traffic generated by these de-

vices. Latency is critical for applications like autonomous vehicles, smart grids, and remote surgery. Even a millisecond delay can have significant consequences. Traditional cloud-centric approaches often struggle to meet these stringent latency requirements. This amount of data transmission over the devices and network may result in throttling of bandwidth, increasing unnecessary delay and posing a negative impact on the industry. Distributed computing, including edge computing, was first introduced as a result of the necessity to address the problems. Using these approaches, the entire process is moved to a decentralized data center. This reduces latency by moving processing and storage closer to the data source.

For example, in a smart transportation system, during peak hours, vehicles may request for collision detection, route optimization, and real-time navigation near urban centers, creating high demand for those edge resources. But during off-peak hours or in less populated regions, the resource usage drops significantly, leaving many edge servers underutilized leading to resource wastage, as idle resources in low-traffic regions remain unused while high-traffic areas experience congestion and latency issues. To handle the emerging complex IoT applications, more research is being done on the distributed paradigms. The lack of scalable, flexible, and interoperable solutions to provide these applications in complex, dynamic, and heterogeneous computing environments is what drives osmotic computing, one of the most promising [7].

Osmotic Computing has developed as an enhancement and extension of other existing computing paradigms like fog, cloud, and edge computing. Osmotic computing dynamically balances workloads between cloud and edge resources in order to optimize resource consumption and service distribution across diverse settings.

The word *"cloud"* first appeared in the telecommunications industry when users began sending data through virtual private networks (VPNs). The definition of cloud computing according to the National Institute of Standards and Technology (NIST) [8] is: "Cloud computing is a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction."

The primary objective of cloud computing is to utilize and combine remote resources to achieve high throughput. It addresses QoS, scalability, delivery paradigms, virtualization, and interoperability [9]. Large amounts of storage and processing power are positioned in close proximity

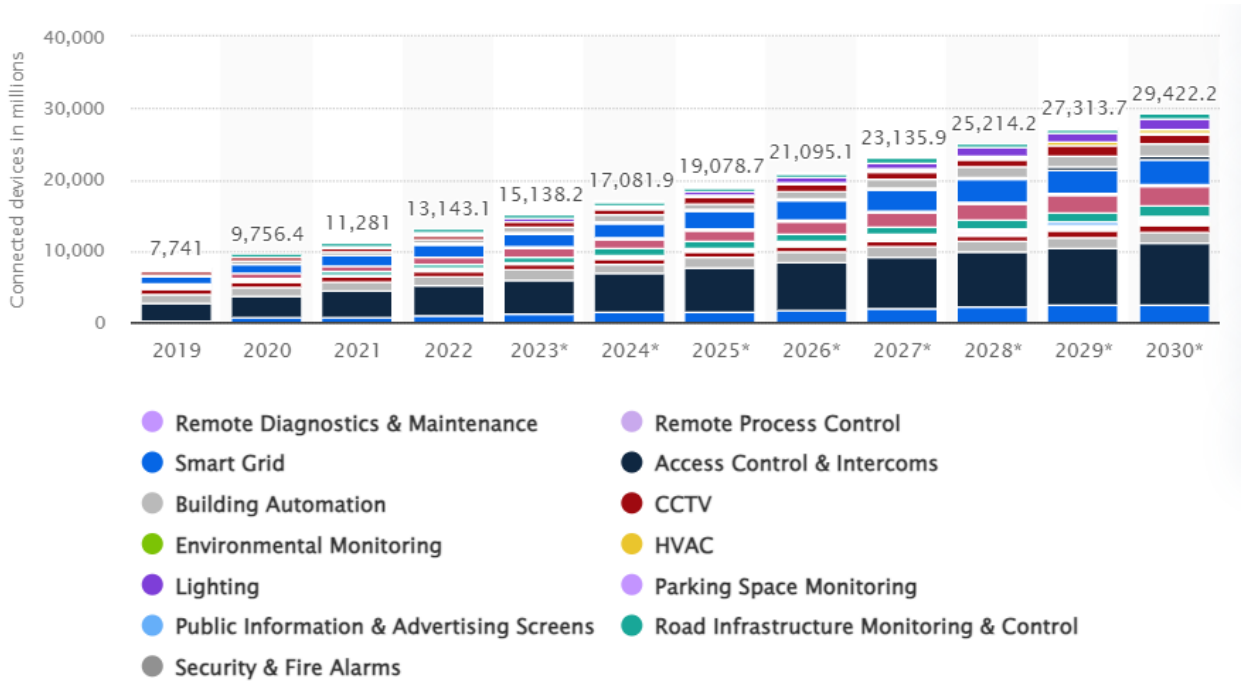


Figure 1.1: Growth of connected devices from 2019 to 2030

to the sensors or mobile devices in edge computing. These are referred to as fog nodes, cloudlets, or microdatacenters [10]. When content delivery networks (CDNs) were first implemented in the 1990s to improve web speed, edge computing had its start [11]. Edge Computing (EC) uses the cloud computing architecture to expand the CDN concept. Cloudlet can operate just like cloud computing. For isolation, resource management, safety, and metering, the code is contained within a virtual machine or a lightweight container. The promise of edge computing was initially illustrated by Brian Noble *et al.* [12], who showed how edge computing might be used to implement voice recognition on a mobile device with limited resources by offloading the work to a nearby server [13].

The Internet of Things (IoT) was introduced in 1999 for use in supply chain management. It leads to the concept of computer sensing information without any human intervention and is widely adapted in the fields such as transport, environment, defense, home, and healthcare [14] [15, 16]. The advent of IoT took us to the post-cloud era, where an extensively large amount of data is produced by the devices immersed in our lives, along with other devices that are deployed at the edge to consume this data. This has led to bandwidth coming to a standstill and the speed of data transmission is also becoming a bottleneck. Figure 1.2 illustrates the conventional cloud computing

structure. This figure shows that the raw data generated by the devices is transmitted to the cloud, and the data consumers send requests to the cloud, and the cloud responds by sending the requested data. But, due to the vast amount of data generated by IoT devices, this infrastructure is not enough

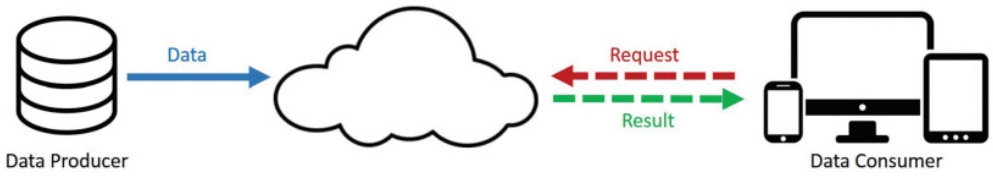


Figure 1.2: Cloud Computing Paradigm [1]

for them. It will lead to more bandwidth and computing resource usage, more energy consumption, and privacy issues. Therefore, offloading the data to the cloud will be more efficient. Figure 1.3 shows the edge computing paradigm. This paradigm results in less bandwidth usage, less energy consumption, and less latency as the data is offloaded near the data producer.

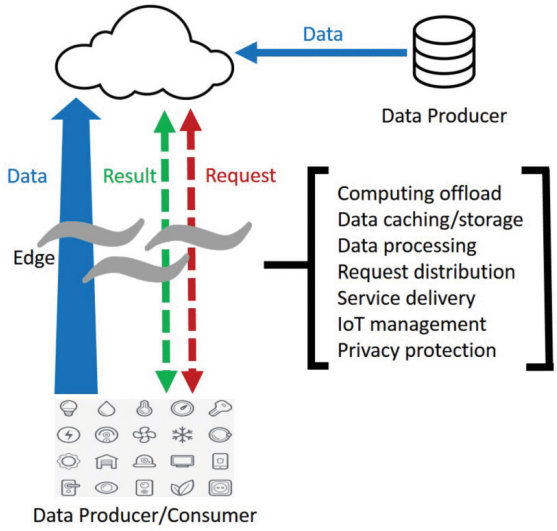


Figure 1.3: Edge computing paradigm [1]

1.2 Osmotic Computing

Osmotic Computing was introduced by Villari *et al.* [2] in 2016. According to Villari *et al.*, the definition of Osmotic computing (OC) is ”Osmotic computing is a new paradigm that is driven by the significant increase in resource capacity/capability at the network edge, along with

support for data transfer protocols that enable such resources to interact more seamlessly with datacenter-based services"

The idea of Osmotic Computing fostered due to an increase in the resource capacity at the edge layer. It enables automatic deployment of microservices in distributed environments. These microservices are interconnected and composed at the edge of cloud infrastructures [2]. In chemistry, "Osmosis" represents the seamless diffusion of molecules from a solution with a higher concentration to a lower concentration. This process represents how the services should be migrated into the cloud network. Figure 1.4 shows the solute particles allowed through the semipermeable membrane.

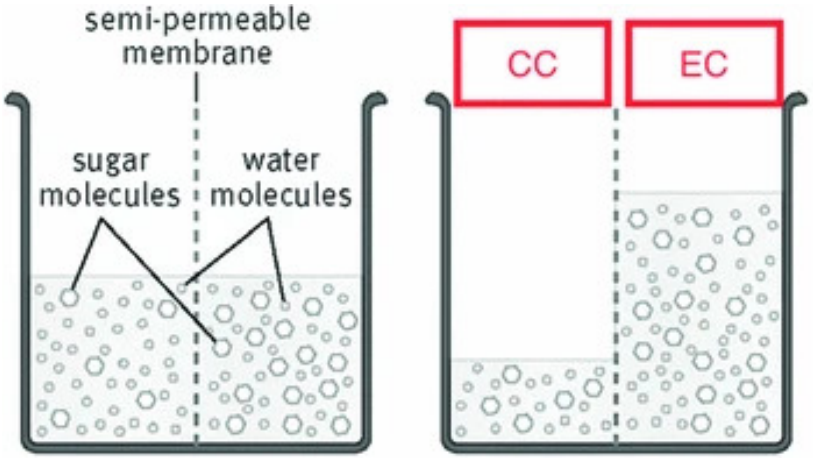


Figure 1.4: The principle of Osmosis and Osmotic computing

Drawing inspiration from the process of osmosis, where fluids move across a semipermeable membrane to reach equilibrium, osmotic computing facilitates the dynamic movement of data and tasks between the cloud, fog, and edge layers. This movement is based on real-time resource availability and the specific requirements of each application. Imagine, for instance, a complex image recognition task that is initially processed at the edge by a local device. If the device encounters resource constraints or requires specialized processing power, osmotic computing allows the task to seamlessly migrate to a more powerful fog or cloud resource, ensuring efficient and timely completion. Depending on the type of application, the service can be moved to edge or cloud.

It uses dynamic and adaptive migration policies to optimize latency. In case of low latency processing services, they are prioritized for edge execution ensuring minimal delay. However,

in case of resource constrained edge servers, latency sensitive processes are offloaded to fog or cloud servers. Although it introduces latency, osmotic computing uses latency-aware scheduling and resource orchestration. This dynamic allocation seeks to optimize resource utilization and achieve a balance between resource availability and processing requirements. Unlike traditional distributed computing models with static resource allocation, osmotic computing fosters a more adaptive and responsive approach that caters to the ever-changing demands of applications and the underlying computing environment [2]. It uses well-known technologies like virtualization [17, 18] and software-defined infrastructure (SDI) [19] to make a flexible and scalable computing environment that can assign resources on demand.

1.2.1 Analogies between Osmosis and Osmotic Computing

i. *Solvent*: The component of the solution that absorbs the solute is called the solvent. It is the only component that can move through the semipermeable membrane. In the computing paradigm, the services need to be moved between the servers by categorizing them into microservices and macroservices. Here, S can be represented by the Equation 1.1.

$$S = \{S_1, S_2, \ldots, S_k\} \tag{1.1}$$

ii. *Solution*: It is the entire network or the entire infrastructure that includes the services (S), resources (R), servers (V), datacenters (D), and edge device (E). The N represents the entire network as shown in Equation 1.2.

$$N = \{S, R, V, D, E\} \tag{1.2}$$

iii. *Solute*: In the process of osmosis, the component of a solution that is not allowed through the membrane is called the solute. In osmotic computing, energy (E), computational power (C), current load (L), and processing time (T) are solute. Here, σ represents the solute, which can be represented as shown in Equation 1.3

$$\sigma = \{E, C, L, T\} \tag{1.3}$$

iv. *Concentration*: It is the ratio of the solute to the solvent. In osmotic computing, it refers to the total number of services that the available computational resources can handle. Concentration ($\mathcal{C}$) is the ratio of the total number of services (S) to the available computational resources (R), as shown in Equation 1.4.

$$\mathcal{C} = \frac{S}{R} \tag{1.4}$$

v. *Semipermeable membrane*: The semipermeable membrane determines the movement of the solvent to maintain the concentration of the solution. The movement depends on the timing (t), where (l) and how (m) of the movement of the services between the edge and the cloud data centers. That is decided by the semipermeable membrane. It is shown in Equation 1.5

$$F = M(t, l, m) \cdot S \tag{1.5}$$

1.2.2 Osmotic Computing Elements

i. *Microelements (MELs)*: MELs are the basic execution units in a communication network. They create abstractions by encapsulating tasks, task dependencies, and required data, enabling migration between different computing resources. This approach simplifies task management and facilitates seamless movement in a computing environment [18].

ii. *Osmotic Pressure Monitors (OPMs)*: These components continuously assess the health of the computing environment by monitoring resource availability (CPU, memory, and storage) in the cloud, fog, and edge layers. In addition, OPMs monitor network conditions (bandwidth and latency) and processing requirements (application requirements). By collecting all this information, OPMs can determine where MELs should go and how they should move between layers to ensure that resources are used efficiently and applications run smoothly [20].

iii. *Osmotic Middleware:* This software layer plays a vital role in coordinating MEL movements. It interacts with OPM in two ways: 1) it receives resource allocation recommendations based on data collected by OPM, and 2) it facilitates the migration of MEL across different computing layers using virtualization technology. The Osmosis middleware acts as a control center and directs tasks and data flows based on OPM's dynamic recommendations [21].

1.2.3 Advantages of Osmotic Computing

i. *Reduced Latency*: Tasks are processed as close to the data source as possible, minimizing the distance that data travels and significantly reducing latency.

ii. *Improved Efficiency*: Resource utilization is optimized by intelligently allocating tasks to the most appropriate layer: cloud for complex tasks that require high processing power, fog for tasks that require medium resources and local processing, and edge for basic computing and real-time response.

iii. *Enhanced Scalability:* Pervasive systems can dynamically adapt to different workloads by seamlessly scaling resources across cloud, fog, and edge layers. As the demand for processing power increases, tasks can be migrated to more powerful resources in the cloud or fog, while periods of lower activity can effectively utilize edge resources.

iv. *Cost Optimization:* By leveraging a combination of cloud, fog, and edge resources based on their processing capabilities and costs, osmotic computing can significantly reduce operational expenditures. Less demanding tasks can be handled by the more cost-effective edge and fog resources, while the cloud is utilized only when its superior processing power is truly necessary.

1.2.4 Applications of Osmotic Computing

i. *Industrial Automation:*Real-time monitoring and control of industrial processes require immediate feedback and decision-making. Osmotic computing can facilitate this by distributing computational tasks between cloud and edge resources, ensuring that critical data is processed locally for immediate response while leveraging cloud resources for more intensive computational tasks. This method can enhance the efficacy and dependability of automated systems in manufacturing, supply chain management, and various industrial applications.

ii. *5G:* Osmotic computing adds cutting-edge functionalities to 5G networks, especially in dynamic allocation of resources. Osmotic computing enhances the performance of 5G infrastructure by dynamically allocating computing resources in accordance with real-time demand and network circumstances. This adaptability is essential for accommodating a range of applica-

tions, including ultra-reliable low-latency communications (URLLC) and massive machine-type communications (mMTC), where fluctuating workload requirements necessitate effective resource management. Additionally, osmotic computing enhances network slicing capabilities by dynamically adjusting configurations to meet specific service requirements. This optimization not only improves service delivery but also ensures efficient utilization of network resources, enhancing scalability and flexibility in 5G deployments. Moreover, osmotic computing contributes to enhancing security and privacy in 5G networks by decentralizing trust management and implementing robust data anonymization techniques at the edge. These measures mitigate risks associated with centralized data processing, promoting secure and compliant operations across 5G-enabled applications.

iii. *Autonomous Vehicles:* Processing sensor data and making critical decisions for navigation requires minimal latency, which is a strength of osmotic computing. Autonomous vehicles rely on real-time data from cameras, LIDAR, radar, and other sensors to make quick decisions. By processing this data at the edge, near the vehicle, osmotic computing reduces latency, enhances safety, and improves the overall performance of autonomous driving systems. Additionally, cloud resources can be used for high-level data analysis and updates to navigation algorithms.

iv. *Augmented Reality (AR) and Virtual Reality (VR):* Effortless user experiences in AR/VR applications depend on low-latency data processing. Osmotic computing offers essential infrastructure by processing data near the user's device, minimizing latency, and facilitating a seamless and immersive experience. This is especially crucial for applications like gaming, simulations, and interactive media, where any latency might impair the user's experience.

v. *Healthcare:* Processing medical data at the edge can ensure patient privacy while enabling real-time health monitoring. Osmotic computing allows for sensitive health data to be processed locally on devices such as wearable monitors or hospital-based edge servers, reducing the risk of data breaches and ensuring compliance with privacy regulations. At the same time, cloud resources can be used for more extensive data analysis, Machine Learning (ML) model training, and long-term storage, facilitating advanced healthcare applications such as predictive analytics and personalized medicine.

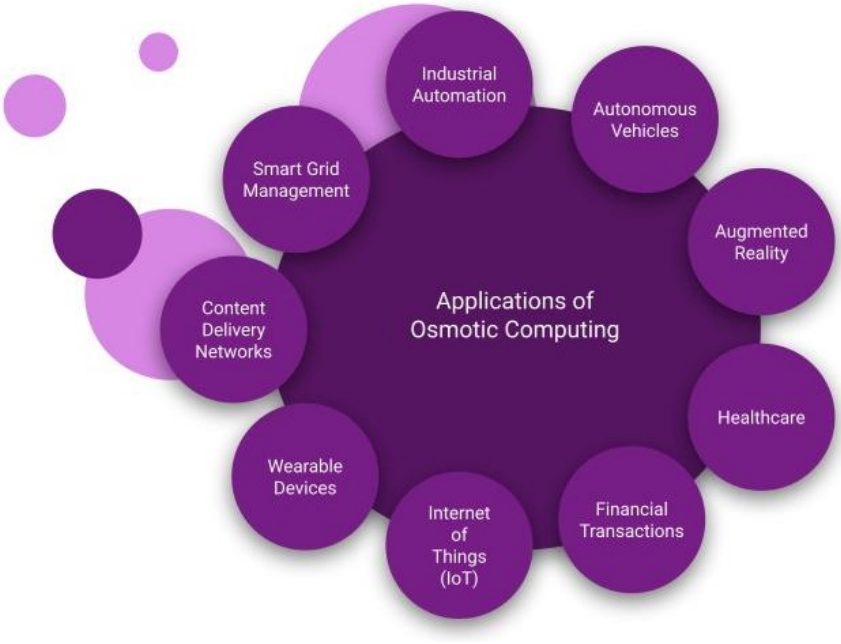


Figure 1.5: Applications of Osmotic Computing

vi. *Financial Transactions:* Optimizing the processing of sensor data from multiple devices can be achieved by integrating edge and cloud resources. Osmotic computing allows IoT devices to locally process initial inputs, minimizing data transfer to the cloud and enhancing response times for low-latency applications. Subsequently, the cloud can be employed for diverse functions, such as aggregating data from several devices, performing intricate analyses, and aiding in decision-making inside industrial IoT applications, smart cities, and smart homes.

vii. *Internet of Things (IoT):* Processing sensor data from numerous devices can be optimized by utilizing both edge and cloud resources. Osmotic computing allows IoT devices to perform initial data processing locally, reducing the volume of data sent to the cloud and enabling faster response times for time-sensitive applications. The cloud can then be used for aggregating data from multiple devices, performing complex analyses, and supporting decision-making processes in smart homes, cities, and industrial IoT applications.

viii. *Wearable Devices:* Smartwatches and fitness trackers can collect data locally and use cloud resources for in-depth analysis. These devices can process basic health information, such heart rate and steps achieved, locally, and provide the user with immediate feedback. Cloud computing capability can be leveraged to execute complex analyses, such as trend analysis,

anomaly detection, and personalized suggestions, all while maintaining battery life and timely updates.

ix. *Content Delivery Networks (CDNs):* Osmotic computing can be used to optimize content delivery based on user location and network circumstances. CDNs can lower latency, speed up load times, and improve user experience by processing and caching content at edge nodes that are closer to the end users. This method works especially well for online gaming, streaming services, and other applications where responsiveness and performance are vital. Analytics, worldwide distribution, and content storage are all possible with the cloud.

x. *Smart Grid Management:* Osmotic computing can enhance a smart grid's optimization of energy distribution and consumption dynamically. Osmotic computing enhances efficiency and reliability by facilitating real-time monitoring and management of energy usage through the processing of data from sensors and smart meters at the network edge. This localized processing enables rapid responses to fluctuations in supply or demand, aiding in grid stabilization and averting disruptions.

1.2.5 Example: Benefits of Osmotic Computing in Smart City Traffic Management

The traffic management systems in smart city environments use devices such as sensors and traffic cameras that are connected to vehicles to collect and process the vast amount of real-time data, including vehicle count, accident detection, and traffic violations for accident detection, which are critical to ensure safety and reducing congestion. In centralized cloud processing, all data is transmitted to the cloud for processing, which leads to high latency, and network congestion becomes a bottleneck during peak traffic periods. In localized edge processing, the devices rely entirely on edge devices, which limits their scalability and performance due to restricted computational and storage resources. Osmotic computing overcomes these challenges by dynamically migrating tasks based on resource availability, task priority, and network conditions. For example, in accident detection, the camera detects a collision. It processes the video feed locally on edge devices, notifying the emergency services immediately, reducing the response time, and leading to immediate response due to low latency. In traffic violation detection cases, the edge devices detect

vehicles violating traffic rules, and the processing for e-challans generation is done locally or on nearby fog nodes if the edge device is overloaded. Osmotic computing thus helps reduce latency, as in the case of critical tasks like accident detection, where the processing is done at the edge layer to ensure a real-time response. In case of traffic rule violations, the workloads are migrated between edge and cloud layers in case of increased traffic volumes. The power-intensive tasks such as machine learning analytics are offloaded to the cloud, conserving energy at the edge devices.

1.3 Task Scheduling and Resource Allocation

Task scheduling and resource allocation are fundamental components of computing systems, essential for optimizing performance, enhancing efficiency, and ensuring the effective utilization of resources. These concepts are particularly vital in distributed and cloud computing environments, where resources are heterogeneous and tasks can vary significantly in their computational requirements. This section provides a detailed overview of task scheduling and resource allocation, their importance, the challenges they present, and various approaches to address these challenges.

1.3.1 Task Scheduling

Task scheduling is important for enhancing system performance, energy efficiency, and resource usage in resource-constrained and highly dynamic networks. With the increase in the number of IoT devices in various domains such as transportation, healthcare, and smart homes, effective task scheduling mechanisms are important to orchestrate the execution of diverse computing tasks over interconnected devices. The distinctive attributes of these IoT devices, including variability in capabilities, constrained processing capacity, and restricted energy and memory resources, present a considerable challenge for job scheduling algorithms.

The traditional algorithms that are inherited from the conventional distributed computing paradigms often fail to address the specific constraints and requirements of the system. Edge computing uses the proximate edge servers to offload the computation-intensive tasks from the IoT devices, thus reducing the latency, and enhancing the overall responsiveness of the system. This distribution of computing tasks closer to the data source reduces the need for round-trip communication with remote cloud servers. It facilitates real-time decision-making and enables

applications with stringent latency requirements, such as autonomous vehicles and healthcare. The ML-based task scheduling algorithms have garnered attention in IoT research. The ML algorithms use historical data, contextual information, and sensor readings to predict the future workload and dynamically allocate the tasks across IoT devices in a manner that maximizes the resource utilization and minimizes energy consumption. Using predictive modeling and advanced analytics, the ML algorithms adapt to the changing device capabilities, environment conditions, and user preferences, thus optimizing the system's performance. The following are the advantages of task scheduling:

- *Minimize Makespan:* Reducing the overall time required for completing all tasks is important for improving the throughput of the system and meeting the deadlines. This involves balancing the load across all available resources to avoid bottlenecks and maximize the parallelism.
- *Maximize Resource Utilization:* The efficient utilization of available resources ensures that computational power is not wasted. High resource utilization leads to better performance and cost-effectiveness, especially in environments where resources are billed based on usage.
- *Maintain Load Balance:* Distributing tasks evenly across resources prevents some resources from becoming overburdened while others remain underutilized. Load balancing helps in achieving higher performance and preventing resource contention.
- *Meet Deadlines:* Ensuring that time-critical tasks are completed within specified time frames is vital in many applications, such as real-time systems and service-level agreements (SLAs). This requires prioritizing tasks based on their deadlines and urgency.

Following are the types of approaches for task scheduling:

- *Static Scheduling:* Tasks are assigned to resources based on pre-determined criteria before execution begins. This approach works well in predictable environments but lacks the flexibility to adapt to changing conditions.
- *Dynamic Scheduling:* Tasks are allocated to resources in real-time, contingent upon the present system status and workload. Dynamic scheduling exhibits greater flexibility and adaptability to changes; yet, it necessitates more advanced algorithms and may incur significant computing costs.

- *Heuristic-Based Scheduling:* Heuristic algorithms, such as the Earliest Deadline First (EDF) or Least Laxity First (LLF), provide practical solutions by simplifying complex scheduling problems with rules of thumb.
- *Metaheuristic Algorithms:* Advanced methodologies such as Genetic Algorithms (GA), Simulated Annealing (SA), and Particle Swarm Optimization (PSO) provide near-optimal solutions through the exploration of extensive search spaces using iterative enhancement strategies.

1.3.2 Resource Allocation

In the era of IoT devices, resource allocation is important for optimizing system performance and ensuring the utilization of scarce resources that are inherent to the IoT environment. It involves dynamic distribution of bandwidth, computational tasks, energy and memory among the devices to cater to the application demands that range from real-time data analysis to remote monitoring and control systems. However, the traditional resource allocation paradigms, such as static allocation schemes, may encounter some challenges in dynamic IoT environments that are characterized by the heterogeneity of devices, their dynamic nature, and variability in network conditions. Therefore, there is an increasing emphasis on developing adaptive and intelligent resource allocation algorithms that can dynamically adjust resource allocations in response to real-time changes in device capabilities, network conditions, and application demands. These advanced resource allocation mechanisms leverage machine learning, optimization techniques, and decentralized decision-making to achieve optimal resource utilization, enhance scalability, and improve overall system reliability in IoT deployments. By harnessing the power of adaptive resource allocation, IoT systems can effectively manage resource constraints, mitigate network congestion, and deliver superior quality of service, thereby unlocking the full potential of IoT technologies to revolutionize diverse industries and enrich human lives. For efficient resource allocation following points need to be considered:

- *Resource Heterogeneity:* Differences in resource capabilities and performance must be taken into account to ensure that tasks are matched with appropriate resources. This includes considering factors such as processing power, memory capacity, and network bandwidth.

- *Task Requirements:* Specific needs of each task, including computational power, memory, and data bandwidth, must be matched with the capabilities of available resources. Some tasks may require high computational power, while others may need large memory or fast storage.
- *Quality of Service (QoS):* Ensuring that resource allocation meets desired QoS metrics, such as response time, availability, and reliability, is critical for maintaining user satisfaction and meeting SLAs. This requires prioritizing resources for critical tasks and ensuring redundancy for high availability.
- *Scalability:* Efficient resource management is crucial in dynamic contexts when the system scales up or down. Resource allocation strategies must effectively manage fluctuating workloads and adjust to alterations in resource availability.

The Resource allocation strategies can be categorized into:

- *Centralized Allocation:* A single entity makes decisions for the entire system, which can simplify coordination but may become a bottleneck and a single point of failure.
- *Decentralized Allocation:* Decisions are made locally by individual resources or agents, which can enhance scalability and fault tolerance but may require more sophisticated coordination mechanisms.

1.3.3 Challenges in Task Scheduling and Resource Allocation

- *Complexity and Scalability:* As the number of activities and resources escalates, the intricacy of scheduling and allocation algorithms expands dramatically, complicating the attainment of optimal solutions in real-time. Efficient algorithms must reconcile the trade-off between optimality and computational complexity.
- *Uncertainty and Dynamics:* The dynamic nature of workloads and resource availability requires adaptive and robust algorithms that can respond to changes promptly. This includes handling unpredictable task arrivals, varying execution times, and fluctuating resource availability.

- *Heterogeneity:* Diverse and heterogeneous resources necessitate sophisticated algorithms that can account for different performance characteristics and compatibility issues. Ensuring compatibility and efficient utilization of heterogeneous resources is a complex problem.
- *Energy Efficiency:* With the growing concern for energy consumption in large-scale computing environments, algorithms must also focus on minimizing energy usage while maintaining performance. This involves optimizing resource allocation to reduce idle times and balance power consumption.
- *Fault Tolerance and Reliability:* Ensuring the system's continued functionality during resource failures is essential for maintaining reliability. This necessitates the formulation of algorithms capable of identifying faults and reallocating tasks to available resources effortlessly.

1.3.4 Osmotic Computing for Task Scheduling and Resource Allocation

Task scheduling and resource allocation are integral components of osmotic computing, playing vital roles in optimizing system performance, enhancing scalability, and ensuring efficient resource utilization. In osmotic computing environments, task scheduling involves dynamically allocating computational tasks to available resources based on real-time system conditions, task requirements, and resource availability. Unlike traditional static or centralized scheduling approaches, osmotic computing enables decentralized task scheduling mechanisms that leverage local knowledge and adaptive algorithms to distribute tasks across the network effectively. Resource allocation in osmotic computing revolves around dynamically provisioning and managing computational resources across interconnected nodes to meet the evolving demands of applications and users. Osmotic computing environments exhibit fluidic resource allocation characteristics, where resources are dynamically shared and redistributed based on demand fluctuations, workload variations, and node capabilities. This dynamic resource allocation paradigm fosters resource efficiency, resilience, and adaptability, enabling the system to gracefully handle changes in workload patterns, device failures, or network disruptions. Moreover, osmotic computing introduces novel concepts such as ”virtual membranes” and ”resource gradients” to facilitate resource diffusion and task migration across the network. Virtual membranes act as dynamic boundaries that regulate the flow of tasks and resources between different computing nodes, ensuring efficient resource utilization while

maintaining system stability. Resource gradients represent the spatial distribution of resources across the network, guiding the migration of tasks towards regions with abundant resources and minimizing resource contention and bottlenecks.

Osmotic Computing offers a transformative approach to task scheduling and resource allocation in distributed computing environments, emphasizing adaptability, autonomy, and resilience. By harnessing principles from chemistry, osmotic computing promises to unlock new levels of efficiency, scalability, and flexibility in distributed systems, paving the way for innovative applications in domains such as edge computing, IoT, and cloud computing. Following are the advantages of Osmotic Computing

- *Enhance Scalability and Flexibility:* Osmotic computing allows for seamless integration and management of diverse resources, enabling scalable and flexible resource allocation across cloud and edge environments.
- *Reduce Latency:* Osmotic computing can markedly decrease latency and enhance responsiveness for time-sensitive applications by processing data nearer to its source. This is especially advantageous for applications necessitating real-time processing and minimal latency transmission.
- *Improve Fault Tolerance and Reliability:* Through distributed resource allocation and redundancy, osmotic computing enhances system reliability and fault tolerance. In the event of resource failures, tasks can be dynamically reallocated to available resources, ensuring continuous operation.
- *Achieve Better Load Balancing:* Osmotic computing facilitates dynamic load balancing across heterogeneous environments, optimizing resource utilization and preventing bottlenecks. This leads to improved performance and efficiency in handling varying workloads.

1.4 Thesis Organization

This thesis is structured into seven chapters, each addressing distinct aspects of the research on *Efficient Task Scheduling and Resource Allocation using Osmotic Computing*. The following provides a brief overview of the content covered in each chapter:

Chapter 1: Introduction

This chapter elucidates the evolving landscape of computing paradigms, driven by the exponential increase in data from IoT devices and the growing demand for low-latency as well as scalable applications. It discusses the projected growth of the global cloud computing market and the critical need for efficient and adaptable computing models. With IoT devices expected to reach 30.9 billion by 2025, traditional cloud-centric approaches face challenges in meeting stringent latency requirements for applications such as autonomous vehicles and remote surgery. This chapter sets the stage by exploring the foundational concepts of cloud, edge, and fog computing, highlighting their limitations in contemporary IoT environments. It introduces osmotic computing as a promising paradigm aimed at optimizing resource allocation and service deployment across heterogeneous environments, offering insights into its principles, advantages, and potential applications across various sectors.

Chapter 2: Literature Review

In this chapter an in-depth exploration of both foundational and contemporary research related to osmotic computing is presented. It begins with an introduction to the concept of osmotic computing, as proposed by Villari *et al.* in 2016, highlighting its innovative approach to microservice deployment inspired by the chemical process of osmosis. This chapter also examines the integration of osmotic computing with emerging technologies such as microservices, blockchain, deep learning, 5G, and intelligent transport systems. Critical aspects of resource management, security, trust, and the role of Artificial Intelligence (AI) and machine learning in optimizing osmotic computing environments are also discussed. Additionally, the development of middleware platforms and application-specific solutions tailored for various domains is reviewed. By covering these topics, this chapter aims to provide a comprehensive understanding of the current trends, challenges, and future directions in osmotic computing research.

Chapter 3: OCTRA-5G Osmotic Computing based task scheduling and resource allocation for 5G networks

This chapter focuses on the evolving landscape of 5G networks, where efficient resource management is paramount to ensuring optimal service delivery and network performance. It presents the OCTRA-5G framework, designed to enhance service segregation and resource allocation through a multi-layered approach. The framework is structured into three integral layers: Task Scheduling,

Resource Allocation, and Resource Pooling. The Task Scheduling layer utilizes osmotic computing principles to determine whether services should remain on the Edge Node (EN) or be migrated to the Cloud Network, thereby minimizing service retention times when resources are insufficient on the Edge. The Resource Allocation layer focuses on the effective distribution of resources to services that are retained on the EN, ensuring efficient utilization. Meanwhile, the Resource Pooling layer leverages the idle resources of User Equipments (UEs) connected to the network, aggregating these resources into the Osmotic Resource Database (ORD) for enhanced resource availability. Together, these layers form a robust framework that significantly improves resource management in 5G networks.

Chapter 4: OsCoMIT: Osmotic Computing-based Service Management for Intelligent Transportation Systems in 5G Network

This chapter explores the rapidly evolving domain of Intelligent Transportation Systems (ITS) and the integration of advanced communication technologies with efficient resource management strategies. It proposes a novel framework based on osmotic computing to address the critical challenges of task segregation and resource allocation within ITS, leveraging the capabilities of 5G networks. The framework is structured into three distinct layers: the Core Network Layer, the Edge Computing Layer, and the Intelligent Transportation Layer. By employing osmotic computing at the Edge Network Layer, the framework enhances performance and reliability through effective task segregation and intelligent decision-making. Additionally, the proposed Proportional Fairness algorithm ensures optimal resource allocation for microservices, further improving the efficiency and robustness of the system. Through these contributions, the proposed framework aims to significantly advance the functionality and applicability of ITS in the 5G era.

Chapter 5: **Optimizing Service Migration and Task Offloading in Osmotic Computing with Deep Q-Networks**

This chapter presents a comprehensive study on the integration of Deep Q-Networks (DQN) within the Osmotic Computing (OC) paradigm to enhance task scheduling and resource allocation across the IoT-Edge-Cloud continuum. Osmotic Computing is introduced as a solution to the latency and resource management challenges posed by the proliferation of IoT devices and the dynamic nature of edge and cloud environments. The focus of this Chapter is on developing a DQN-based approach for intelligent decision-making in OC environments. Through reinforcement learning,

the DQN algorithm adapts to changing network conditions, ensuring effective resource allocation and enhanced performance. The model's effectiveness is validated through extensive simulations, which demonstrate significant improvements in task processing speed and overall system efficiency. Furthermore, the chapter includes a statistical analysis of the simulation results using the Wilcoxon signed-rank test, confirming the superior performance of the proposed method compared to traditional approaches like Random Agent, Q-Learning, and SARSA algorithms.

Chapter 6: $\mu-osmotic$: A Novel Approach for Dynamic Resource Allocation in IoT Edge-Cloud Environments

This Chapter presents a novel approach to dynamic resource allocation between edge and cloud environments, leveraging the principles of Osmotic Computing (OC) combined with the Advantage Actor-Critic (A2C) algorithm. This chapter delves into the challenges associated with managing the vast amount of data generated by Internet of Things (IoT) devices, highlighting the importance of scalable and efficient computational paradigms. The core of the proposed framework, named $\mu-osmotic$, is built on a microservice architecture designed to optimize IoT data processing. To validate the effectiveness of this approach, the framework's performance is compared with other prominent algorithms, including Soft Actor-Critic (SAC), Proximal Policy Optimization (PPO), and Deep Q-Network (DQN). The results demonstrate that $\mu-osmotic$ outperforms these algorithms across various metrics, achieving significant improvements in energy consumption, CPU utilization, and memory usage. This chapter provides a comprehensive evaluation of $\mu-osmotic$, establishing it as a robust solution for dynamic resource allocation in IoT environments.

Chapter 7: Conclusion

This chapter concludes the thesis by discussing the overall conclusions of the research work. The chapter also suggests some future research directions and possible extensions of this work.

Chapter 2

Literature Review[1]

The chapter provides an in-depth exploration of the foundational and contemporary research of osmotic computing. It begins by introducing the concept of osmotic computing, as proposed by Villari et al. in 2016, highlighting its innovative approach to microservice deployment inspired by the chemical process of osmosis. It also examines the integration of osmotic computing with emerging technologies such as microservices, blockchain, deep learning, 5G, and intelligent transport systems. Additionally, it delves into critical aspects of resource management, security, trust, and the role of AI and machine learning in optimizing osmotic computing environments. Furthermore, the chapter discusses the development of middleware platforms and application-specific solutions tailored for various domains. The chapter aims to provide a comprehensive understanding of the current trends, challenges, and future directions in osmotic computing research by reviewing these topics.

2.1 Osmotic Computing

Osmotic Computing (OC) was introduced in 2016 by Villari *et al.* [2]. OC enables the automatic deployment of microservices. The concept is taken from the chemical process of *Osmosis*. The authors also presented an Osmotic Ecosystem as shown in Figure 2.1.

[1]The contents of this chapter are published as: Kaur, A., Kumar, R., & Saxena, S. (2020, November). "Osmotic computing and related challenges: A survey". In 2020 Sixth International Conference on Parallel, Distributed and Grid Computing (PDGC) (pp. 378-383). IEEE.

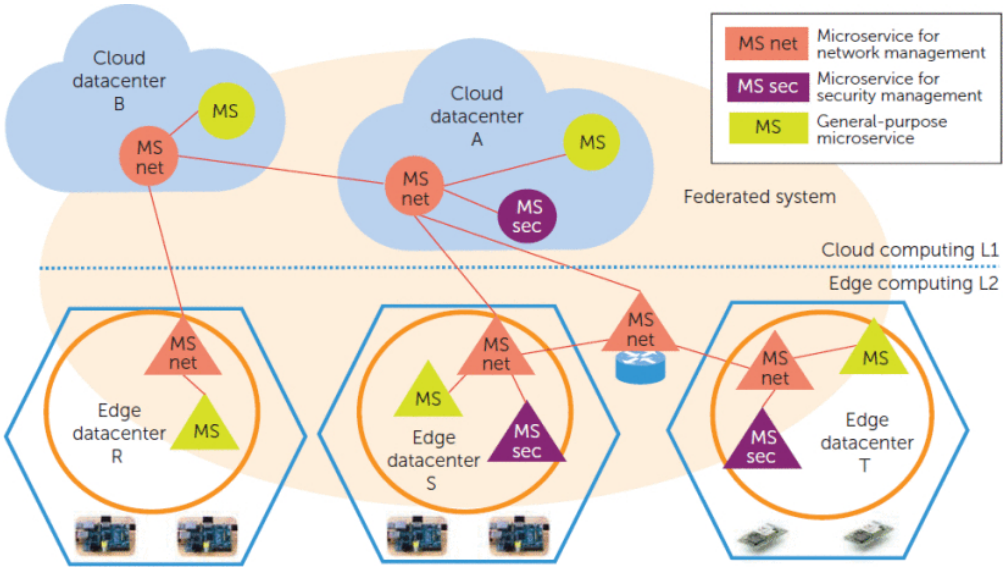


Figure 2.1: Osmotic Ecosystem [2]

Osmotic computing combines different computing paradigms, such as edge, cloud, and Fog, for IoT-cloud integration. It builds upon and expands the current methodologies. OC endorses the opportunistic administration of MicroElements (MELs). It is an abstraction that constructs and implements IoT applications across various resource kinds. These MELs are transferable across several resources. MELs are abstractions that encapsulate services, data, and resources; hence, IoT applications can be structured as a graph of MELs and transferred across various infrastructures according to requirements.

Carnevale *et al.* [22] presents a smart orchestration architecture to enable osmotic computing that focuses on the deployment of microservices. Figure 2.3) presents the MELs hierarchy as presented by the authors. The authors propose an osmotic-based architecture to facilitate the workflow for MELs registration, compute requests, and migration to a learning process.

Buzachis *et al.* [23] present a work in which they use a gamified cognitive rehabilitation use case to apply a close-loop OC flow model. In this the authors were able to maximize the usage of resources according to the user requests, ensuring the main requirements like fault tolerance, scalability, ease of management, and interoperability. The authors in [24] present a method to manage the security and trustiness in osmotic computing. They proposed a Software Defined Membrane (SDMem), responsible for the transfer of MELs. The basic idea is to leverage private blockchain over federated systems. Nardelli *et al.* [25] proposed Osmotic Flow, a model for holistic

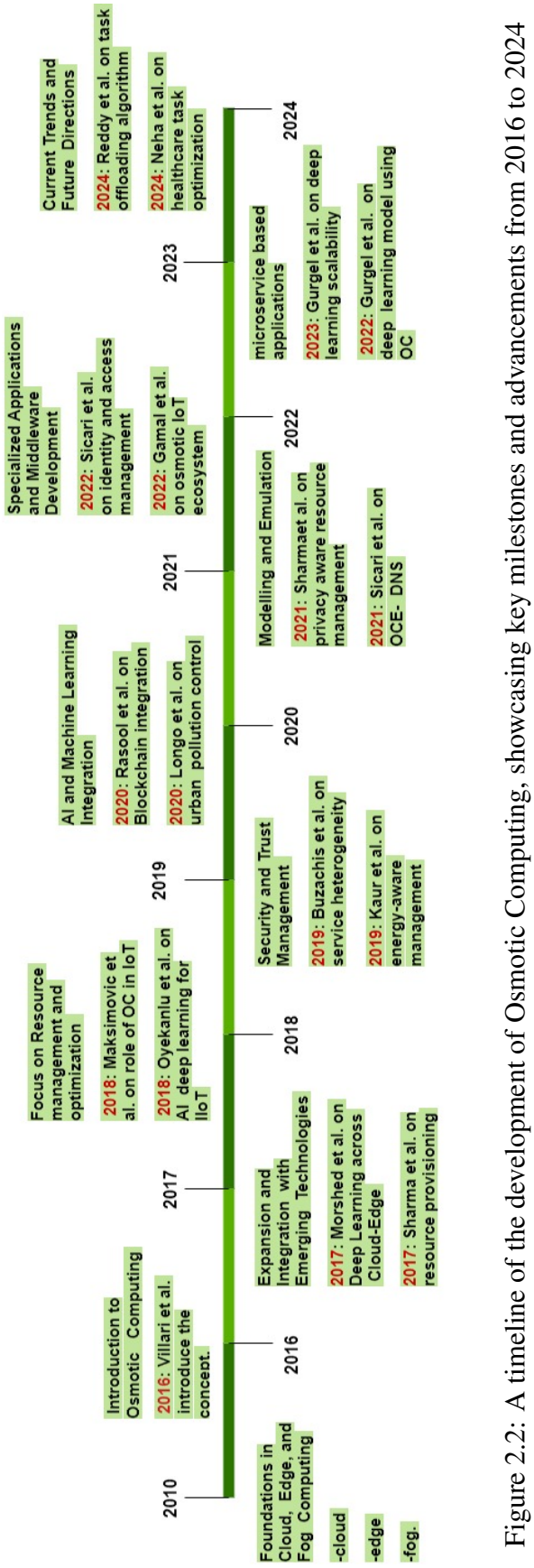


Figure 2.2: A timeline of the development of Osmotic Computing, showcasing key milestones and advancements from 2016 to 2024

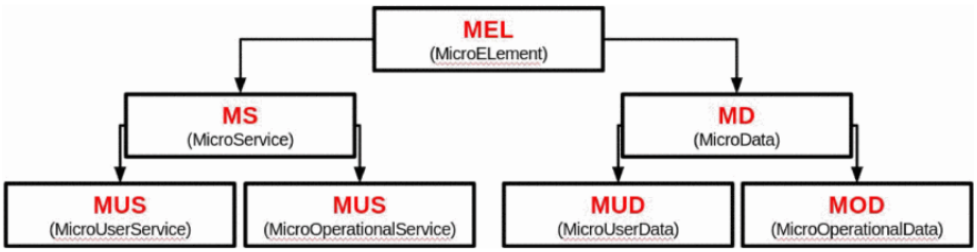


Figure 2.3: MicroELements hierarchy [3]

programming, that executes and maps IoT data applications on a distributed infrastructure. The model is modeled as a cyclic graph. The data transformation task encapsulated a microservice, a computational model, and a data analysis programming model for smart traffic light IoT application. M. Maksimovic [26] discusses the fundamental concepts of OC and examines its importance within the IoT framework, along with its advantages and limitations. Figure 2.4 presents the components of a literature survey that are discussed in this chapter, and Figure 2.2 present the evolution of osmotic computing over the years.

2.2 Osmotic Computing Integration with Emerging Technologies

2.2.1 Microservices

J. Thones [27] describes microservices as small applications that can be deployed, tested, and scaled independently and have a single responsibility. The author gives an example of a single thing that a microservice can be: a single responsibility in terms of a functional or non-functional requirement or a cross-functional requirement, or it can be a responsibility for serving a particular resource or resource representation. Larrucea *et al.* [28] provides a table of technologies for microservices. The authors discuss the process of migration to microservices and give a list of guidelines: be prepared for organizational changes, analyze the system, delineate the architecture, select components for migration, and execute coding, design testing, and integration. So, microservices can be called as fine-grained services that can be independently deployed and managed. Microservices represent a cutting-edge approach to software architecture where applications are broken down into

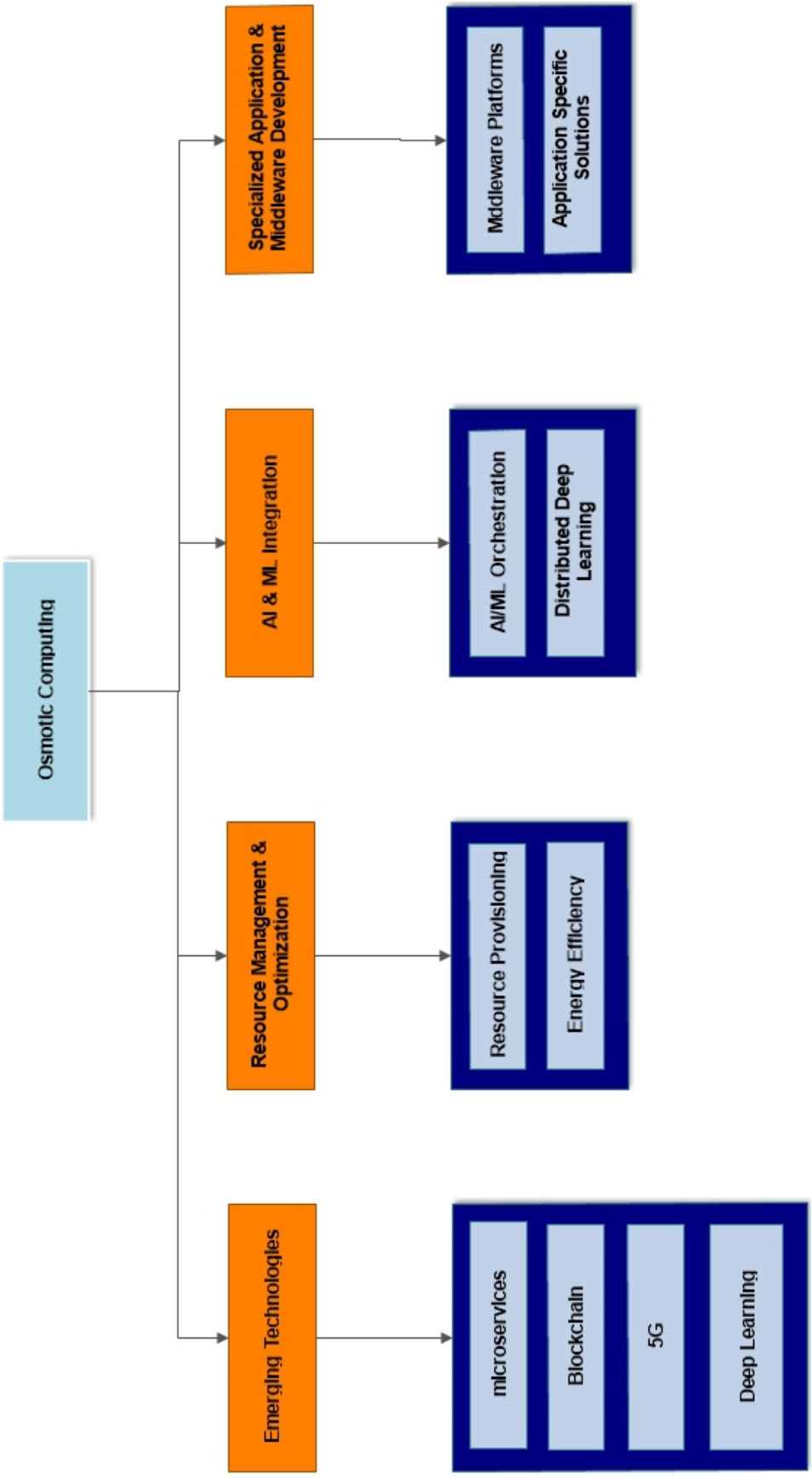


Figure 2.4: Overview of Key Topics Discussed in the Literature Review on Osmotic Computing

small, self-contained units that operate independently. Each microservice focuses on a specific function and communicates with others through well-defined interfaces, typically using lightweight protocols like HTTP/REST or messaging queues. This architecture contrasts sharply with traditional monolithic systems, where all functionalities are bundled into a single, indivisible unit. The microservices model offers numerous benefits that align perfectly with the principles of Osmotic Computing, particularly in enhancing task scheduling and resource allocation.

Souza *et al.* [29] introduce an integrated monitoring system for the monitoring of IoT applications in the form of microservices which are executed using osmotic computing. They used a smart parking application to evaluate and demonstrate the technique proposed. Through experimentation, the authors have validated that the framework can identify the variations in memory, CPU, and network latency of the microservices. Buzachis *et al.* [23] investigated the connectivity issues using different types of network overlays and deployed CoAP and FTP microservices to collect the results of transfer times. The authors conceptualized an ecosystem with a centralized virtual machine symbolizing the system's brain and several worker nodes. The apps that have been built are allocated to these worker nodes. The microservices operate within a Docker container, coordinated by Kubernetes. The test results indicate that the OVN is the superior answer relative to other algorithms. It generates a reduced overhead relative to a non-overlay approach. OVN serves as the foundational technology for the implementation of a dynamic osmotic network.

2.2.2 Blockchain

The concepts of blockchain and bitcoin were first proposed by Satoshi Nakamoto in 2008. They described how open distributed ledger and cryptology can be combined into a digital currency application. The term "blockchain" originated from its technological configuration—a sequence of blocks. Every block in the blockchain is interconnected with the preceding block using a cryptographic hash. A block is a data structure that facilitates the storage of a transaction list [30]. The widespread adoption of blockchain technology is attributable to the success of Bitcoin. It is referred to as a distributed ledger and constitutes an append-only data structure. The blockchain is seen as a chronological record of transactions [31]. Blockchain is maturing and is now applied in a number of applications like healthcare, supply chain, and contracts [32]. It makes the concept of a shared registry from distributed systems a reality for various application domains. It can be regarded as

a quality leap from distributed database technology [33]. It maintains a continuous list of data records. These records are approved by the participating nodes [34]. Buzachis *et al.* [35] Propose a MELs orchestration strategy utilizing an SDMem that capitalizes on blockchain capabilities within osmotic computing. The authors used OC to control and manage the heterogeneity provisioning of MELs, which can be divided across different layers of the infrastructure. The authors propose a secure approach for MEL orchestration using blockchain. Sharma *et al.* [36] discuss the use of blockchain in cyber-physical systems. Hassija *et al.* [37] present a crowdsourcing framework using blockchain for farmers to deal with their problems. Blockchain is implemented to provide data security by integrating it with neural network algorithms. Rasool *et al.* [38] presents the importance of using a technique to improve the reliability of an osmotic computing manager by using the blockchain. The authors highlighted the features of blockchain that can improve OC reliability. The use of blockchain features improves trust among the independent vendors of MultiCoT. The authors also list the challenges of integrating blockchain with osmotic computing. They have proposed a semi-private blockchain solution and improved the consensus algorithm. In the field of secure distributed computing, several solutions leverage serverless paradigms with a focus on dynamic behavior adaptation using Function-as-a-Service (FaaS). To address the need for rapid interoperability of on-demand services, the authors of [39] proposed a Blockchain-based solution named BCB-FaaS. This approach aims to ensure trust and accountability in function configuration, offering strong protection against cyber attacks. Additionally, a cost analysis demonstrated that Blockchain can be a cost-effective alternative for securing decentralized communications. Their work highlights the potential of Blockchain for secure communication in various scenarios, including human-to-machine, machine-to-machine, and human-to-human interactions within Cloud/Edge Computing and IoT environments.

2.2.3 Deep Learning

Machine learning (ML) underpins numerous aspects of the modern world, encompassing social networks and e-commerce platforms. These traditional machine learning algorithms were constrained in their capacity to process natural data in its raw state. Representation learning techniques utilize raw data to autonomously identify the necessary representations for classification or detection. Deep learning techniques are methods of representation learning characterized by

several levels of representation. They are derived by integrating fundamental non-linear modules. They convert the representation from one level to another [40, 41].

Deep learning, a machine learning branch, tries to model high-level abstractions of data. It has improved many different artificial intelligence tasks like speech recognition and object detection. There are three important reasons for the need for deep learning: an increase in chip processing capabilities, low hardware costs, and the advances in machine learning algorithms [42].

In the past, language modeling and computer vision have used deep learning. Nowadays, It is used for medical diagnostics and for the prediction to improve human health. But, transferring this large amount of data to the cloud data centers is a major limitation.

Osmotic Computing promotes data distribution of data analysis tasks across the edge and cloud layers. Morshed *et al.* [43] analyzes the research challenges involved in developing holistic deep learning algorithms that are data and resource-aware. The authors explored the orchestration of deep learning across cloud-edge-mobile environments using osmotic principles. Pacheco *et al.* [44] presents a model using osmotic computing and deep learning for person recognition. These applications utilize both cloud and edge computing. However, issues emerge when deep learning systems must contend with the limitations that result in inadequate scalability in cloud computing or edge computing environments. Gurgel *et al.* [45] introduce an adaptive load distribution utilizing Osmotic Computing, which influences the scalability of deep learning. Oyekanlu *et al.* [46] propose that current hardware can be utilized to engage in distributed edge computing for the Industrial Internet of Things (IIoT). The authors employed distributed Osmotic Computing and affordable hardware to address intricate deep-learning challenges.

The Internet of Things (IoT) has gained significance in applications such as smart cities, surveillance systems, and healthcare, each possessing distinct Quality of Service (QoS) requirements. Dynamic traffic management facilitates optimal load balancing and routing, enabling applications to attain their requisite level of Quality of Service (QoS). Osmotic Computing is a framework for the integration of edge and cloud computing. The data movement between the cloud and edge may generate substantial traffic. The routes require optimization. Consequently, identifying an effective route between the source and destination MEL is crucial. Deep learning can facilitate this procedure by leveraging extensive routing data. Absardi and Javidan [47] offers an innovative traffic management technique utilizing a recurrent neural network model.

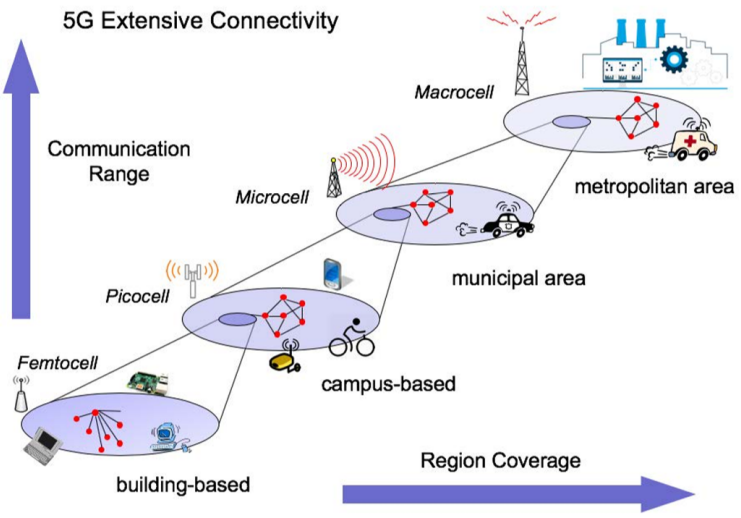


Figure 2.5: 5G Network Connectivity envision [4]

2.2.4 5G Networks

As compared to 4G, 5G has a new wireless interface to support higher frequency and spectrum efficiency along with improvement in signal management and accounting procedures to accommodate the needs for a diverse range of new applications that range outside of the traditional broadband category [48]. It offers comprehensive connection via diverse wireless access, spanning from femtocell to macrocell. Figure 2.5 illustrates the connectivity of the 5G network. Ding *et al.* [4] address the issues of 5G from the standpoint of vertical application domains. The subsequent applications are driven by 5G technology [4]:

- Smart mobility: Mobility applications encompass traditional route planning and autonomous driving services. The advantages encompass traffic distribution, accident mitigation, cost and emission reduction, and energy conservation.
- Smart energy: This application includes smart grid networking, power failure detection, power plant management, and monitoring.
- Smart health: With growing awareness about health and fitness, health applications are becoming popular among people. Another application, the AR/VR surgery, demands low latency and high bandwidth along with general requirements of security, low power, and data privacy.

- Industrial applications: Industrial applications signify the next generation of cyber-physical services such as manufacturing, 3D printing, and AI-enhanced building. These confer advantages to society.
- Consumer applications: Consumer applications such as mobile streaming, pervasive gaming, blockchain, and holographic technology demand high bandwidth, security, reliability, and low latency.

Deterministic execution of computation-intensive and delay-sensitive tasks is important for industrial applications. Peng *et al.* [49] introduce a task deterministic network (TDN) designed to facilitate communication with zero jitter and minimal latency for important tasks. The authors have detailed the functions of TDN, including clock synchronization, cycle mapping, and network slicing. Gupta *et al.* [50] addressed the problem of slice allocation problem using a machine learning algorithm. The authors also discussed the SDN and NFV technologies that provide better performance with the dataset used. Choudhary *et al.* [51] have presented a detailed description of security in 5G networks along with Osmotic computing and catalytic computing. The authors have presented a security model called CATMOSIS. It includes security features based on osmotic and catalytic computing. Monia *et al.* [52] overviews various security issues and presents the real-time applications with possible attacks. The authors have also presented security challenges and their possible solutions.

The notion of Intelligent Transportation Systems has emerged as an essential component of smart cities. The essential elements of Intelligent Transportation Systems (ITS) are sensing, data preprocessing, traffic pattern analysis, traffic forecasting, and information communication and control. The LiDAR sensor generates a three-dimensional point cloud. It is applicable for object detection, object tracking, obstacle detection, lane detection, and mapping within the perception system [53]. The camera captures photos and movies. The raw data contains two-dimensional measurements. These can be transformed into traffic-related statistics. They are applicable for object detection, video-based traffic sensing, vehicle identification, vehicle tracking, and anomaly detection [54]. The fundamental principle of ITS is to utilize the current infrastructure to enhance traffic efficiency. It pertains to tunnels, terminals, railways, traffic signage, etc. Intelligent transportation system is the integration of advanced information and communication technologies into

the vehicular infrastructure to enhance safety and sustainability. It aims to optimize the traffic and improve emergency services. In recent years, autonomous driving has become a hot topic for both car manufacturers and researchers. In some countries, specific regulations have been issued for autonomous vehicles. However, these vehicles are difficult to manage in urban areas where the traffic is high, and the interaction between the drivers and the autonomous vehicles will be high. So, a well-harmonized strategy is needed to achieve the purpose of driving the vehicles in a hybrid manner. Filocamo *et al.* [55] propose osmotic computing as a remedy for self-adaptive urban traffic in the context of autonomous vehicles. The authors have examined vehicle-to-vehicle (V2V) and vehicle edge-cloud (V2EC) interactions. The proposed framework facilitates interaction among automobiles, pedestrians, and infrastructure within a dynamic environment. This provides ongoing information regarding interactions and urban traffic. Kapoor *et al.* [56] proposed a distributed and localized congestion detection across large-scale road networks. Their algorithm studies the causal relation between congested road intersections. Lei *et al.* [57] proposed a task offloading scheme by exploiting the multi-hop vehicle computation resources. The authors formulated an optimization problem to obtain the corresponding offloading decisions. The simulation results show significant improvement in terms of the response delay.

2.2.5 Edge AI

OC presents a flexible management model utilizing microservices. In this context, deployment and migration strategies pertain to infrastructure necessities such as availability, load balancing, and reliability. Buzachis *et al.* [23] emphasize the Osmotic Ecosystem platform, which concentrates on the deployment of a blue-green mechanism utilizing microservices and incorporates technologies such as Docker, Agento, Kubernetes, and MongoDB. The experiments in the study report demonstrate the duration necessary to organize, implement, and dismantle the microservices. The administration of resources and services in the Mobile Augmented Reality Network (MARN) is cumbersome. The MARN networks comprise users with augmented reality and virtual reality applications. The primary requirements of these networks are minimal latency, high tolerance, and resilience. These can be acquired through near-user technologies like edge computing, which must be connected with cloud computing. The amalgamation of various networks can be achieved by osmotic computing. The authors in [58] propose a method utilizing OC to transfer and orga-

nize services across various tiers of the network. The authors provide a detailed account of the framework's components, encompassing its primary applications, classifications, service types, scheduling, and service migrations. The assessment of the suggested technique is conducted on 100,000 requests, revealing a notable enhancement with an error probability of 0.1% and a 55.72% energy conservation rate.

2.3 Resource Management and Optimization

Resource management guarantees that the network is equipped to address any emergencies or unplanned jobs. The integration of millions of IoT devices necessitates the critical migration of services and the assurance of operational safety. Sharma *et al.* [59] regard AIoT as an amalgamation of artificial intelligence and mobile Internet of Things. This necessitates an urgent reply, computational resources, and an auxiliary support system. The authors' work employs Osmotic Computing to formulate techniques for determining service-sharing ways through optimal and privacy-conscious resource management. Sharma *et al.* [58] provide an ideal solution utilizing osmotic computing for the efficient migration and scheduling of services across servers of disparate layers, along with a comprehensive overview of various components for the integration of osmotic computing. Carnevale *et al.* [60] proposed an architecture based on a multi-agent system using osmotic computing and MicroELements (MELs). Neha *et al.* [61] introduce an osmotic task manager designed to enhance mapping decisions for all services through a dual categorization: task level and processing level categorization. The suggested method classifies all incoming requests according to their resource requirements and optimally allocates tasks to resources. Latency in real-time data processing during computing resource supply is highly common.

The objective of Pervasive Social Networking (PSN) is to connect users with services and facilitate conversation, regardless of time and place. Online social networks have progressed alongside the increase in mobile users, necessitating more secure, reliable, and private services. Sharma *et al*. [62] have introduced a framework for prioritizing application-specific trust management. The suggested methodology employs a versatile mixing model and constructs the system based on six attributes. The framework employs OC for computational offloading. Offloading decreases the quantity of computations and the duration of computations. The framework forecasts user ratings

with minimal inaccuracies. This processing of IoT devices has largely impacted cloud-based applications, including finance, healthcare, smart grids, and traffic management which require data to be transmitted to data centers at large distances. But IoT handles this data by relocating the processing of data to the periphery of the data generation. Osmotic computing seeks to sustain the performance of geographically distributed services by ensuring a harmonious interaction between cloud and edge data centers. Nonetheless, data offloading around several data centers frequently elevates energy usage, adversely affecting the Quality of Service (QoS) for IoT applications. Kaur *et al.* [63] introduced a decision-making framework, En-OsCo, to enhance energy-conscious dynamic resource management. This framework comprises four essential components: i) Monitors the edge data centers using the Extended Kalman filter; ii) Uses Hyper-heuristics for optimal service allocation to the edge/ cloud configurations; iii) Reduces energy consumption and service latency; iv) Reduces the Hyper-heuristics search space by recording prior decisions through Universal Streaming Monitoring. Validation with Container CloudSim and HyFlex on PlanetLab datasets demonstrates that En-OsCo significantly reduces energy consumption and service delay, hence advancing its objective of enhancing the performance of IoT-based applications.

Sicari *et al.* [64] address concerns associated with the OC, including MEL nomenclature, delineating the geographical context of services rendered by MELs, concealing MEL migration to clients utilizing MELs in a seamless manner, and offering novel insights to establish MEL migration protocols. The authors have built an enhanced DNS system to reference Micro Elements (MELs) utilizing Extended Plus Codes (EPC), a revolutionary three-dimensional geocoding technique created to enable the accurate identification of geographical regions served by MELs. The EPC method, an extension of Google's Open Location Code, is crucial in this framework for facilitating fast MEL identification and migration management. The OCE-DNS infrastructure, along with a high-performance design, guarantees swift updates and retrievals of Resource Records (RR), hence facilitating seamless osmotic MEL migrations. The variance in service characteristics due to differing requirements is termed service heterogeneity, and managing services based on this heterogeneity presents a challenge. Sharma *et al.* [65] identifies the resolution of this problem in OC, as it facilitates the categorization of services according to their specific requirements. The authors introduce a fitness-oriented osmotic algorithm. The objective of their research is to optimize the use and augment the capacity of near-site computational infrastructure. The method uses a fitness

function to differentiate and assign services into microservices and macroservices. The findings are illustrated using numerical simulations.

2.4 Artificial Intelligence (AI) and Machine Learning (ML) Integration

Carnevale *et al.* [3] proposes an OC based multi-agent system based on MELs. MELs encapsulate resources and data and migrate them across different network layers. The authors proposed a Body Area Network (BAN) to explore the potential of osmotic computing. BAN consists of tens of sensors that generate streams of data about a patient for real-time analysis and to fill electronic medical records. But, with the growth of data streams, receiving real-time alarms about health status becomes a challenge. The proposed technique uses Artificial Intelligence (AI) to learn through osmotic resource monitoring. In the coming years, the biggest growth rate in the network traffic will be smartphones and internet-connected devices. These devices produce huge data. Machine Learning and Edge computing are emerging as paradigms to process this huge amount of data. Pacheo *et al.* [44] presented an OC architecture for IoT smart classrooms to test the deep learning model for person recognition. The results of the experiments show that the on-device ODI-MEC DNN interference outperformed the other by four times due to network latencies and bottlenecks.

2.5 Specialized Applications and Middleware Development

Gamal *et al.* [66] introduced a hybrid meta-heuristic approach that integrates osmotic behavior with a bio-inspired load-balancing method. OC facilitates the automated deployment of virtual machines moved over cloud infrastructure. Hybrid artificial bee and ant colony optimization have demonstrated their efficacy in dynamic conditions. The authors have employed these algorithms in conjunction with OC. The application of OC mitigates the limitations of these optimization strategies. The findings indicate a decrease in energy usage and the frequency of VM migrations. The proposed technique also improves Quality of Service (QoS), as indicated by Service Level Agreement Violation (SLAV), and mitigates performance degradation resulting from migrations.

Numerous studies have investigated methods to surmount the constraints of stationary urban pollution monitoring. Although these stations deliver accurate data owing to superior equipment and standardized methods, their geographic coverage is constrained by their stationary positions. Mobile Crowd Sensing (MCS) presents a viable option by utilizing mobile devices as data-collecting sensors. Nonetheless, the integration of disparate data streams from diverse device kinds, protocols, and semantics continues to be a difficulty. Osmotic computing resolves this issue by establishing an abstraction layer between mobile devices and the cloud. This layer enables data filtration and the incorporation of metadata, hence enhancing data processing efficiency. Longo *et al.* [67] expand upon these notions by creating middleware that consolidates data from mobile and IoT devices in urban settings, utilizing the advantages of osmotic computing for enhanced data management. This middleware, utilizing a message broker, streamlines data acquisition from mobile devices and permits elastic distribution of linked devices [68].

Apollon demonstrated strong performance in both functional and non-functional tests, although the crowd-sensing component necessitates methods to mitigate data flooding. The research also highlights existing issues within the Osmotic Computing paradigm, notably the absence of consistent deployment and behavior for microservice migration across osmotic layers. Autonomous driving is emerging as a prominent subject for automotive manufacturers and researchers. Filocamo *et al.* [55] provide a solution for self-adaptive urban traffic in the context of autonomous vehicles. The research focuses on vehicle-to-vehicle (V2V) and vehicle-to-cloud (V2C) interactions within urban traffic. The framework can adjust to a dynamic environment in which automobiles, pedestrians, and physical infrastructure interact with one another. The authors want to execute the proposed system in real time. Villari *et al.* [68] introduce an emulation framework named OsmoticToolkit. This is the inaugural emulation environment created to tackle the task. The authors present the functions of their research and validate them experimentally through an e-health scenario involving the deployment of microservice-based hospital apps. The experimental results demonstrate its efficacy regarding workloads, outcomes, and Quality of Service (QoS) needs while maintaining the Service Level Agreements (SLAs). The swift expansion of cloud technologies has resulted in the implementation of microservice architecture, which employs container virtualization to create resilient and modular systems. The rise in the number of operational containers necessitates effective management of these units. OC examines migration, optimization, and deployment of

microservices from cloud environments to edge and IoT devices. Connectivity difficulties must be addressed to accomplish this.

2.6 Summary of Literature Review

Author, Reference, Year	**Application**	**Objective**	**Method/Technology Used**	**Result**	**Gap**
Sharma *et al.* [62], 2017	Pervasive social networks	Framework that generates higher trust value with low monitoring cost	Flexible mixture model, lock-door policy, artificial bee ant colony optimization, threshold, probability based movement	Predicts user ratings with low error of range $\pm 2\%$	Mirrored sources in lock-door policy may or may not be authenticated
Morshed *et al.* [43], 2017	Medical diagnostics	Investigates research challenges in developing resource, data-aware deep learning algorithm	Deep learning algorithms	Identified challenges for cloud-edge driven DDL approach	No studies about combining different data sources using deep learning

Author, Reference, Year	**Application**	**Objective**	**Method/Technology Used**	**Result**	**Gap**
Sharma *et al.* [65], 2017	Service heterogeneity	A fitness based algorithm to provide OC support by using existing fog computing infrastructure more efficiently	Proposed an algorithm	Lower allocation latency and higher service handling probability, resource utilization	User privacy, infrastructure security
Rausch *et al.* [69], 2018	Message-oriented Middleware	Applies OC principles to Message-oriented middleware	Server-side messaging protocol (MQTT protocol), gateway network, geo-location	Architecture can serve as stand-alone application for device to device communication	Messages received during reconnection may not reach migrating client
Villari *et al.* [24], 2018	Blockchain	Manages trustiness, security of MELs	Private blockchain, signing and encryption, Json object	Manages security and trustiness of MELs	Blockchain can be applied inside workflow programs

Author, Reference, Year	Application	Objective	Method/Technology Used	Result	Gap
Pacheco *et al.* [44], 2018	IoT smart classroom	Presents OC architecture for an IoT smart classroom	CNN, Tiny YOLO V2, intel, movidius, DNN model	Edge performed better than fog layer	Network speed, transfer-process bottlenecks
Carnevale *et al.* [22], 2018	Healthcare	To motivate the need to move traditional HIS (hospital information system)	Kubernetes, Docker, MongoDB, LDAP	Shows deployment of back end systems over cloud-edge layer	MELs can be deployed using AI
Buzachis *et al.* [70], 2019	Communicati	Investigates connectivity issues in various overlay networks	Cloud, calico, OVN, weave, Kubernetes, container, flannel	Found OVN based solution better than others	Route can be dynamically changed

Author, Reference, Year	**Application**	**Objective**	**Method/Technology Used**	**Result**	**Gap**
Souza *et al.* [29], 2018	Smart Parking system	Monitoring system for IoT applications	Apache JMeter, apache tomcat, raspberry pi, open stack cloud, Ubuntu, JVM	Identifies variations in memory, CPU, network latency at the service, application, infrastructure level	Network traffic and memory consumption are not extensively explored
Gamal *et al.* [66], 2019	Tested on Simulator	Metaheuristic technique combines bio-inspired load balancing algorithm with osmotic computing	Hybrid ant colony and artificial bee colony optimization	Enhances QoS, decreases energy consumption, number of VM migrations, shutdown hosts compared to existing algorithms	More energy consumption than that of ACO and ABC

Author, Reference, Year	**Application**	**Objective**	**Method/Technology Used**	**Result**	**Gap**
Carnevale *et al.* [3], 2019	Healthcare	BAN scenario to explore osmotic computing potentiality	Reinforcement learning, time series algorithm	Proves potentiality of proposed system	Architecture needs to be validated
Sharma *et al.* [58], 2020	Mobile Augmented Reality Network	Uses OC for efficient resource scheduling, classifies services	Proposed algorithm	0.1 at 55.72% conservation in memory, energy	No tightly related for comparison, Resources follow a decreasing trend
Buzachis *et al.* [70], 2019	IoT	Proposes a technique for the fast re-deployment of Microservices	Docker, Kubernetes, Agento, MongoDB	Time performances prove goodness of proposed technique	AI can be integrated
Kaur *et al.* [63], 2019	IoT	Framework for Osmotic Computing, designed for energy-aware dynamic management of resources	Extended Kalman Filter (EKF), Hyper-heuristics, Universal Stream Monitoring	Low energy consumption, high QoS	Data migration over geographically distributed areas

Author, Reference, Year	**Application**	**Objective**	**Method/Technology Used**	**Result**	**Gap**
Longo *et al.* [71], 2020	Urban pollution monitoring	Design, develop middleware to integrate mobile and IoT data using OC	Raspberry Pi or Arduino, Apache Kafka, Kubernetes, mongo DB, Apache Spark	Unit and integration testing guarantees of functional and non-functional requirements	Standard for Osmotic Infrastructure deployment and relocation of microservices
Mirjana Maksimović *et al.* [26], 2018	IoT	Significance of OC in IoT Vision	Theoretical	Applied in IoT applications	-
Rasool *et al.* [38], 2020	Blockchain	presented importance of using the proactive Blockchain-enabled Osmotic Manager (B-OM) to improve reliability of OC	Blockchain, Multi-CoT	Trust management among multiple cloud vendors	Need of a reliable OC in MultiCoT

Author, Reference, Year	Application	Objective	Method/Technology Used	Result	Gap
Ruggeri *et al.* [39], 2022	Blockchain	To analyze and compare different solutions of server-less paradigms	Blockchain, cryptographic functions, data structure, smart contracts	Overhead introduced is acceptable, increases battery life of IoT devices	Need to find better trade-off between data privacy and system efficiency
szydlo *et al.* [72], 2022	Use of renewable energy sources	OC simulation based on autonomic osmotic agents and renewable energy sources	Management algorithms	Use of adaptation algorithm results in utilization of low-emission and solar energy sources	Method for individual devices to know how their decisions effect the system
Sharma *et al.* [59], 2021	Artificial Intelligent of Things(AIoT	Optimal and privacy aware resource management in AIoT	Build safety competition on top of configuration rewards	Presented qualitative comparison	Authentication and access control
Foilocamo *et al.* [55], 2018	Autonomous Vehicles	To present framework for self adaptive city traffic	ESP32	On-going research	Security needs to be considered

Author, Reference, Year	**Application**	**Objective**	**Method/Technology Used**	**Result**	**Gap**
Sicari *et al.* [64], 2021	DNS for distributed service relocation	To identify MELs that serve the specific area	Containers, Kubernetes, Overlay networks	Gives quick response	Need to define migration rules
Galletta *et al.* [73], 2020	Security of microservices in smart city scenario	To access the applicability of secret share techniques	Secret share algorithms	Results showed that 40% MELs do not allow to recompose themselves	Need to find the nodes that minimize chunks transfer
Galleta *et al.* [74]	MELs	Focuses on MEL addressability problem	RR, container, kubernetes	Results shows that the average time required for write request depends on the number of requests and cluster size	Need to define rules for MEL migration.

Author, Reference, Year	Application	Objective	Method/Technology Used	Result	Gap
Fazio *et al.* [75]	Path planning, 2021	To minimize routing path computation time	Dijkstras algorithm, Hadoop, Map reduce	Reduces the path computation time when the microservices are network is large	-
Buzachis *et al.* [23], 2018	eHealth	To motivate moving Health Information Technology to OC based innovative infrastructure.	Based on Ataxia and Dislexia use case	Deals with new challenges in cloud computing due to proliferation of devices	Limited microservices
Morabito *et al.* [76], 2023	Serverless engine	To modify openWolf serverless application	Strengthens security in serverless applications	Improves execution time of cyphertext by 65%	Need system to support more open source platforms

Author, Reference, Year	**Application**	**Objective**	**Method/Technology Used**	**Result**	**Gap**
Swain *et al.* [77], 2023	Physiological record system	To manage vast number of patient records and access them as and when needed	Microservices , Containers, APIs, NoSQL, JSON, YAML, Kubernetes	Theoretical system	Need to implement in real world.
Reddy *et al.* [78], 2023	Task offloading	To investigate task-offloading strategies for the three tier hierarchical model	Task offloading and scheduling algorithms	Noble improvements were observed in execution time, and resource utilization	Include preemption of tasks
Mlotshwa *et al.* [79], 2020	Security	To add security metrics along the layers of osmotic paradigm	Blockchain, Artificial Intelligence	Improves safety of microservices	Fog nodes can be incorporated
Souza *et al.* [80], 2018	Smart city	To deploy and execute smart city application using microservices in distributed environment	Container, Docker, Virtual machine	efficient utilization of computing resources	Need to consider other resources

Author, Reference, Year	Application	Objective	Method/Technology Used	Result	Gap
Hati *et al.* [81], 2020	Smart city	To implement security in smart city application using osmotic processing	Blockchain, AWS, RDS	Reduces the CPU utilization	Need to reduce energy consumption
Souza *et al.* [82], 2022	Smart city	To propose middleware platform to deploy develop and implement applications in smart city	Big data, Map/reduce technique	effectiveness in terms of scalability	Implement live action migration
Buzachis *et al.* [83], 2019	Video surveillance in smart cities	To ensure beneficial resource utilization	FaaS, osmotic Computing	Flexibility, scalability	-
Neha *et al.* [84], 2024	Healthcare systems	To propose offloading algorithm for compute intensive, latency intensive, and energy-consuming tasks	ELBO algorithm	Maximizes the overall efficiency of the healthcare system	Real world implementations

Author, Reference, Year	Application	Objective	Method/Technology Used	Result	Gap
Forti *et al.* [85], 2022	Smart campus	To propose decentralized framework to enable infrastructure and application operators declare management policies	-	Improves response time	need to verify the application
Souza *et al.* [86], 2022	Smart city	To propose model for data persistence in hybrid networks	Containers, SQL, microservices	100% data consistency	Still in progress
Gamal *et al.* [66], 2019	IoT applications	Platform for dynamically orchestration the process of resources across heterogeneous devices	MAPE-K model	Provides self-maintained system	Need to address the security and privacy issues
Loseto *et al.* [87], 2022	Cyber physical systems	To propose microservice architecture using osmotic computing	Containers, kubernetes, microservices	Increase feasibility and reduces development costs	Authors propose a lot of future work

Author, Reference, Year	Application	Objective	Method/Technology Used	Result	Gap
Oyekanlu *et al.* [88], 2018	Industrial Internet of Things	To propose approach using resource constrained edge analytics	C28x, Hampel identifier	Can detect and possibly prevent cyberattack	Need to decrease cost
Sharma *et al.* [89], 2018	Media handovers	To control the handoffs between fog layers	Osmotic Absorption key	Simulation results shows the effectiveness of the prorposed scheme	-
Okafor *et al.* [90], 2017	IoT Network	To handle high traffic load on the network	Virtual machine	Virtual environment offers high availability	Need to consider power management and security
Douhara *et al.* [91], 2020	IoT	to optimize power consumption	Kubernetes	9.9% reduction in power consumption	need to consider security
Spillner *et al.* [92], 2020	Functional decomposition applications	To provide a method for transitioning from cloud-native to continuum-native applications	Virtual machine	Works on virtual applications	Failed to account for application side properties

Author, Reference, Year	**Application**	**Objective**	**Method/Technology Used**	**Result**	**Gap**
Tapwal *et al.* [93], 2022	Blockchain	To achieve real-time consensus	Virtual blockchain	18% reduction in CPU, 92% reduction in memory, and 56%reduction in consensus time	Security needs to be considered.
Debauche *et al.* [94], 2022	Multimedia applications	To address the problem of congestion and computing time	Nvidia GPU, containers, Raspberry pi, kubernetes	Optimized cost and feasible	Need to consider energy consumption
Xu *et al.* [95], 2022	IoT	To enable heterogeneous connectivity in IoT applications	Raspberry pi, CoAP and HTTP	Connecting with IoT device through Proposed technique is faster, frees from repetitive tasks	Need to consider energy consumption

Author, Reference, Year	Application	Objective	Method/Technology Used	Result	Gap
Heiden *et al.* [96], 2022	Vehicular edge computing	To enhance the system performance using container retention	Containers, ML algorithms	Success rate increased by 7%and the profits obtained are 23% higher	Real services can be deployed
Reddy *et al.* [78], 2023	IoT	To find the best suitable device to execute the service	ML algorithms	Convergence rate improved	preemptive task assignment
Prakash *et al.* [97], 2020	Cloud network	Enabling migration of VMs using osmotic computing	LR and OPR algorithms	The proposed algorithm has minimum energy consumption	Need to analyze cost

2.7 Problem Formulation

The rapid proliferation of Internet of Things (IoT) devices, coupled with the advent of 5G networks and the growing demand for real-time data processing, has created a complex environment for resource management and task scheduling. Traditional computing paradigms struggle to efficiently handle the dynamic and distributed nature of modern networks, where the seamless

migration of services and optimal allocation of resources are crucial for maintaining performance standards.

Osmotic Computing (OC) has emerged as a promising paradigm for addressing these challenges by facilitating dynamic orchestration of resources between edge and cloud environments. However, several critical challenges remain in effectively implementing OC to meet the demands of 5G networks, Intelligent Transportation Systems (ITS), and IoT applications. These challenges include:

- Service orchestration management is a major issue in osmotic Computing considering hybrid Cloud/Edge/IoT systems. The current orchestration approaches are highly dependent on either manual or simple condition checks, that is error prone and complex to orchestrate the increasing number of services. Thus, an automated orchestration method is needed for increase efficiency [98, 99].
- Fog computing handles the issues of latency in latency sensitive services but the issue of redundancy, cost effectiveness and handling services on the basis of different characteristics remains. That difference in characteristics of services is called service heterogeneity. An approach to manage and control heterogeneous services by allocation of different servers is required [23].
- To offload data and computation to edge/cloud and for dynamic, optimal, and user demand matching offloading, orchestration needs to work closely with IoT devices. So, dynamic offloading in mobile-edge, cloud-edge systems is a challenge [25].
- Task scheduling is influenced by the mobility of users. A user may migrate from one location to another location, understanding the pattern of how user migrates is one of the main challenge in handling services [14, 20].
- Security remains a challenge as it has not been considered in systems related to Osmotic Computing. To enable microservice execution and migration, edge and cloud computing supporting security policy is needed [2].
- microservices execute at edge not on cloud datacenters in Osmotic Computing. So, a strategy for the allocation of resources to microservices is required [12].

- Osmotic computing is based on the use of macroservices and microservices. So, a standard is needed to classify microservices and macroservices for a computing environment [23].

2.8 Objectives

- To study and analyze the literature of Osmotic Computing.
- To devise a method(s) for task/services recognition and segregation.
- To design and develop technique(s) for efficient task scheduling and resource allocation.
- To analyze and validate the proposed technique(s).

Chapter 3

OCTRA-5G: Osmotic Computing based Task Scheduling and Resource Allocation for 5G Networks[1]

In the ever-evolving landscape of 5G networks, efficient resource management is paramount to ensure optimal service delivery and network performance. This chapter presents the OCTRA-5G framework, designed to enhance service segregation and resource allocation through a multi-layered approach. The framework is structured into three integral layers: Task Scheduling, Resource Allocation, and Resource Pooling. The Task Scheduling layer utilizes Osmotic Computing (OC) principles to determine whether services should remain on the Edge Node (EN) or be migrated to the Cloud Network, thereby minimizing service retention times when resources are insufficient on the Edge. The Resource Allocation layer focuses on effectively distributing resources to services retained on the EN, ensuring efficient utilization. Meanwhile, the Resource Pooling layer leverages the idle resources of User Equipments (UEs) connected to the network, aggregating these resources into the Osmotic Resource Database (ORD) for enhanced resource availability. Together, these layers form a robust framework that significantly improves resource management in 5G networks.

[1]The contents of this chapter are published as: Kaur, A., Kumar, R., & Saxena, S. (2022). "OCTRA-5G: osmotic computing based task scheduling and resource allocation framework for 5G." Concurrency and Computation: Practice and Experience, 34(28), e7369. https://doi.org/10.1002/cpe.7369

3.1 Introduction

Efficient task scheduling and resource allocation are critical in the dynamic and resource-intensive environment of 5G networks. Traditional scheduling algorithms, such as First-Come-First-Serve (FCFS), Priority Scheduling, and Shortest Job First (SJF), have been extensively studied for their straightforward implementation and ability to handle diverse workloads. This study integrates these algorithms within the OCTRA-5G framework to benchmark and validate the proposed osmotic computing-based approaches.

- First-Come-First-Serve (FCFS): FCFS is one of the simplest scheduling algorithms, where tasks are executed in the order they arrive. Its significance lies in its fairness, as no task is denied execution based on its complexity or priority. In the context of 5G networks, FCFS serves as a baseline to evaluate how the proposed framework improves resource allocation and reduces service delays compared to a non-prioritized approach.
- Priority Scheduling: Priority Scheduling executes tasks based on predefined priority levels, ensuring that high-priority tasks receive resources ahead of others. This algorithm is particularly relevant for 5G networks, where certain tasks, such as emergency alerts or time-sensitive applications, must be prioritized to meet service-level agreements (SLAs). Incorporating Priority Scheduling allows the framework to address the heterogeneity of tasks in 5G environments effectively.
- Shortest Job First (SJF): SJF prioritizes tasks with the smallest execution times, minimizing the average waiting time for tasks in the queue. Its inclusion in this study highlights the framework's ability to manage workloads efficiently and optimize network performance by reducing latency, a key metric in 5G systems.

These algorithms were chosen for their diversity in Task Management. They represent different scheduling philosophies—non-prioritized (FCFS), prioritized (Priority Scheduling), and optimized (SJF). Their well-established nature facilitates benchmarking against the OCTRA-5G framework to measure improvements in latency, resource utilization, and service retention times. Their ability to handle varying task complexities and priorities reflects real-world 5G workloads, making them suitable candidates for this study. By employing these algorithms, the OCTRA-5G

framework's performance is comprehensively evaluated, ensuring its robustness and practicality in handling diverse and dynamic 5G network scenarios.

3.2 Proposed OCTRA-5G Framework

Osmotic **C**omputing based **T**ask **S**cheduling and **R**esource **A**llocation for **5G** network (OCTRA-5G) addresses several key aspects: It maintains the resource database that maximizes the usability of Mobile Edge Computing (MEC) resources available, making decisions at the Osmotic Orchestrator (OO), executes the *microservices*, and allocates the resources to the services at the Edge Nodes (EN). In a cell-free 5G network scenario, consider G_k (mMIMO) access points, where $k \in \{1, 2, \ldots, m\}$, each consisting n antennas where, $n \in \{64, 65, \ldots, 128\}$, with $\mathcal{U}_e$ User Equipments (UEs) at a given instance of time. $\mathcal{C}_{G\mathcal{U}}$ represents the channel between $\mathcal{U}_e$ and G mMIMO access points. For ease of reference, key notations are given in Table 3.1. The proposed framework is divided into three layers: UE Layer, Ng-Radio Access Network (Ng-RAN) layer, and Core Network (CN) layer. The Framework is shown in Fig. 3.1.

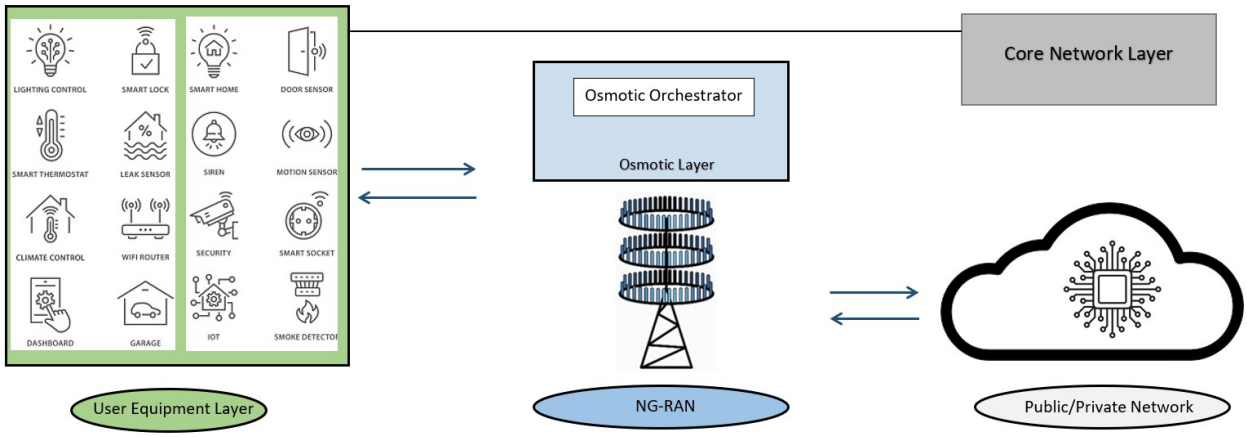


Figure 3.1: OCTRA-5G layers

3.2.1 User Equipment Layer

The User Equipment layer is a set of interconnected mobile devices such as smartphones, cars, robots, medical devices, etc. It is represented by $\mathcal{U}_e$, where $e \in \{1, 2, \ldots, n\}$, n is the maximum number of devices in the network. The devices in the network can request a service as required

Table 3.1: Summary of Key notations.

Decision Variables	
gNB	Radio base station
G	mMIMO
t	Time
U_e	User Equipment
U_{ei}	Service requests
$\mathcal{C}_{\lambda\mathcal{U}}$	Channel between U_e and λ
$\mathcal{T}^s_{comp}$	Computation time
$\mathcal{T}^s_{res}$	Resource reservation time
$\mathcal{T}^s_{tran}$	Data transmission time
$\mathcal{T}^s_{ener}$	Total energy consumption
$Exec_t$	Execution Time
θ^s	Calculation strength required per unit to perform service
d^s_{size}	Size of the Service
$\mathcal{C}_o$	Total Computation time an osmotic server on an edge layer can handle
$\mathcal{M}_{th}$	Threshold value for Memory at OO
$\mathcal{C}_{th}$	Threshold value for computation time at OO

by the application that is running on the device such as data storage, real-time data processing, agricultural services, GPS data, etc. These services require resources for execution, and the UE devices may lack computational or storage resources to handle the required services. Then, the devices send service requests to the Ng-RAN in MEC for access to the required resources.

3.2.2 Ng-RAN Layer

The objective of this layer is to deal with the delay-sensitive applications at the edge layer. It uses the available resources and reduces the burden on the public/private Cloud. The increase in the number of mobile devices and sensors has necessitated the need to execute the sensitive data closer to the user at the edge layer. Let S be the set of tasks to be performed, and let R be the set of resources required for S. The services sent by the edge devices for the resources are sent to the OO. At the OO, the services are segregated into *microservices* and *macroservices*. The *microservices*, being small in size, are executed at the Edge while the *macroservices* are migrated to the Cloud. The *microservices* require fewer resources, while the *macroservices* require more resources.

At the edge layer, there can be multiple OO. Let O_i be the set of OOs, where i is the identification number of the OO and $i \in \{1, 2, \ldots, m\}$. The O O has three components: (i) Osmotic Resource Database (ORD), (ii) Osmotic Resource System (ORS), and (iii) Osmotic Scheduler System (OSS), which are installed on the edge layer as shown in Fig. 3.2.

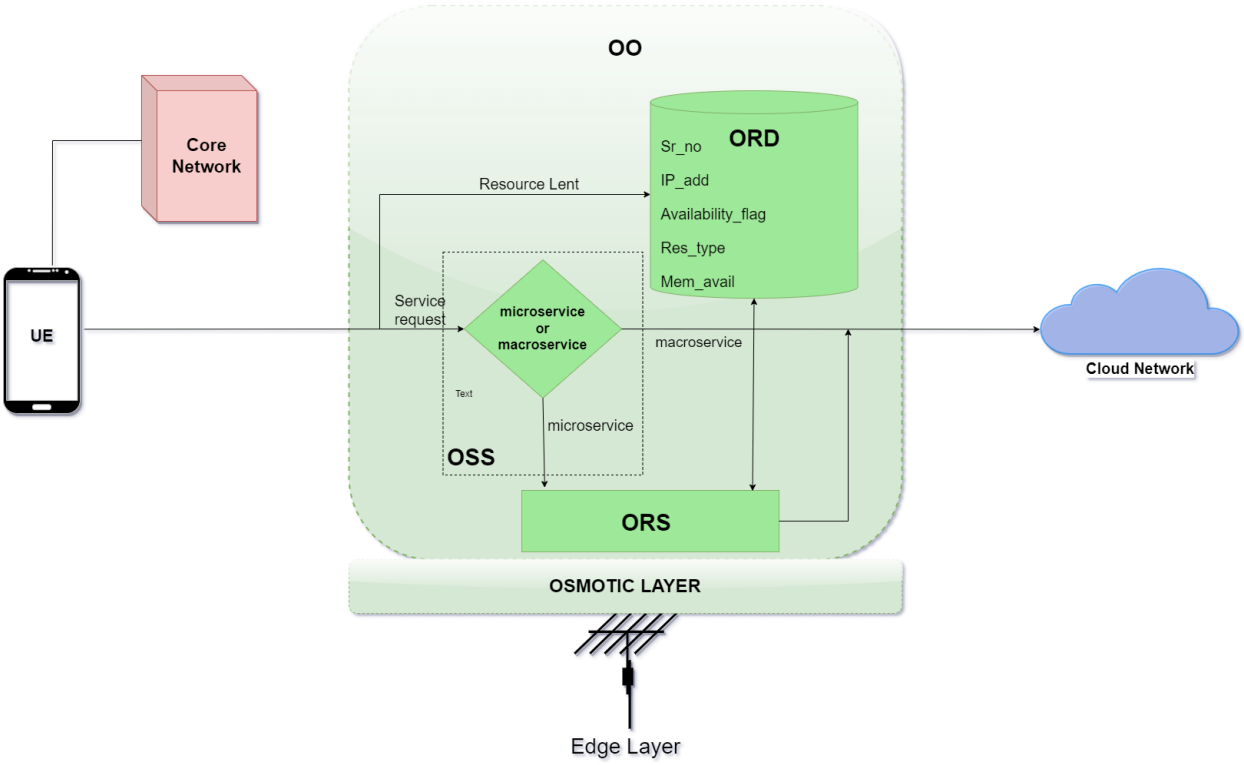


Figure 3.2: OCTRA-5G.

- *Osmotic Resource Database (ORD):* The $\mathcal{U}_e$ are connected to mMIMO system. These devices can lend their idle storage or computational resources to execute the services. OO maintains the lent resources in a database called ORD. It includes the parameters to track the available mobile resources. The parameters include IP address, availability flag, resource type, and memory available of the device offering resources. The capabilities of a device that lacks resources can be enhanced.

- *Osmotic Scheduler System (OSS):* OSS determines the optimal location for service execution. It classifies the services into *macroservices* and *microservices*. Edge layer resources are used for the execution of *microservices*, while the *macroservices* are migrated to the cloud layer for execution.

- *Osmotic Resource System(ORS):* The resource system allocates the available resources to the *microservices*. In case resources are unavailable in the resource database, the microservice is migrated to the next osmotic node.

3.2.3 Core Network (CN) Layer

The 5G core network layer is important for 5G network functionality. This layer includes various functions such as authentication, security, session management, and traffic generation for the connected devices. The layer provides a sophisticated interconnection network to support these functions.

3.2.4 System Model

Consider a Mobile Edge Computing network that consists of gNBs and User Equipments (UEs). Each UE is connected to a gNB that is in its vicinity. The edge server provides computation and storage services. It is connected to both the fronthaul and the backhaul networks. UE device generates a service request, $\mathcal{S}_{ei} = \langle ct_{ei}, s_{ei}, p_{ei}, d_{ei}, dt_{ei} \rangle$, where: ct_{ei} is the computation type, s_{ei} is the size of the service request, p_{ei} is the priority, d_{ei} is the ideal delay, and dt_{ei} is the delay threshold.

The location of each device at the time of service request is recorded at the edge node (EN) of the gNB, represented as $l_e = \langle x, y, al \rangle$, where x and y denote the coordinates on the horizontal plane, and al denotes the altitude. Upon receipt of a service request, the Osmotic Orchestrator (OO) determines whether to execute the service at the edge node or migrate it to the Public/Private Cloud based on the threshold values calculated. Each edge node has limited available computational capacity($\mathcal{C}_{th}$), memory capacity($\mathcal{M}_{th}$). The total number of resources and memory allocated should not exceed the amount of memory and resources available. These constraints can be represented as shown in equation 3.2 and **??**:

$$R_C : \sum_{i=1}^{n} x_i \cdot r_i \leq C_{th} \tag{3.1}$$

$$M_C : \sum_{i=1}^{m} y_j \cdot m_j \leq M_{th} \tag{3.2}$$

This decision is based on the requirements for resources, memory, and execution time. Both data transmission time and computation time of services directly affect the execution time. This segregation is performed based on the model shown in Equation 3.3:

$$\mathcal{R}_{all} = \begin{cases} if\ \mathcal{T}_{comp} < \mathcal{C}_{th}\ \ \&\ \ d_{size} < \mathcal{M}_{th}\ allocate\ to\ ORS \\ if\ \mathcal{T}_{comp} \geq \mathcal{C}_{th}\ \ \&\ \ d_{size} \geq \mathcal{M}_{th}\ allocate\ to\ Public/Private\ Cloud \end{cases} \tag{3.3}$$

Equation 3.4 calculates the Reference Signal Power (RSP) [100]. In this equation, Maximum Transmit Power (MTP) represents the transmit power in decibels per single channel. The Resource Block (RB) cell denotes the total number based on the cell bandwidth, with each RB consisting of 12 Resource Elements (REs).

$$RSP = MTP - 10 * log10 * (RBCell * 12) \tag{3.4}$$

$$TTP = MTP + 10 * log10 * (no.\ of\ antennas) \tag{3.5}$$

Using Eq.3.5, the Total Transmit Power (TTP) [101] of each mMIMO system is evaluated for various antenna configurations. For instance, antenna configurations for mMIMO systems may include 8, 16, 64, or 128 antennas. For a system with a Maximum Transmit Power (MTP) of 40 dBm, a sub-carrier spacing of 15 kHz, and a Resource Block (RB) cell size of 270, the Resource-Specific Power (RSP) can be determined as follows:

$$RSP = 40 - 10 * log10(270 * 12)$$
$$RSP = 40 - 35.10$$
$$RSP = 4.9dBm$$

Taking the same MTP and 8 antennas, the TTP can be calculated as:

$$TTP = 40 + 10 * log10(8)$$
$$TTP = 40 + 9.03$$
$$TTP = 49.03dBm$$

3.2.5 Implementation

At time t, a user $\mathcal{U}_e$ may require the execution of a service. In some cases, a UE or Internet of Things (IoT) device might lack the necessary computational capacity, storage, or other resources to execute specific services. Consequently, the UE device sends a request to the associated gNB.

In the OCTRA-5G framework, each gNB is equipped with memory and computational capabilities. The mMIMO antennas are endowed with greater power and are considered to possess an Operational Layer (OL), with each OL containing one OO. Each layer of the gNB has r resources and can support a maximum of s services generated by $\mathcal{U}$ users. To allocate resources to each service, the total execution time, memory requirements, and resource capacities are calculated. The following process is adhered to:

- *Resource Pooling:* The *ORD* database stores information regarding available resources at each edge network. This database includes details such as the resource type, current availability status, IP address, and maximum memory available on each device. To manage service heterogeneity efficiently, a resource pool is created to consolidate this information. The *ORD* maintains a flag indicating the availability of each resource. When a resource is allocated to a service, the *availability* flag is set to *false*. Upon successful completion of the service and subsequent release of the resource, the *availability* flag is updated to *true*.

- *Service classification*: Let $\mathcal{U}_{ei}$ represent the service requests made by each UE, where $\mathcal{U}_e$ denotes the identity number of the UE, and i indicates the request number. Thus, the total number of service requests denoted as $\mathcal{R}_T$, made by all UEs is given by the Equation 3.6:

$$\mathcal{R}_T = \sum_{n=1}^{e} \sum_{m=1}^{i} \mathcal{R}_{mn} \tag{3.6}$$

  The service categorization model categorizes services according to their operational attributes and resource demands, including size and computational needs, about certain task characteristics. Existing models allocate services based on available resources, which can lead to both heavy and light applications being processed by the same server, thereby creating complex optimization challenges [58]. This can result in overburdening a single server while causing resource wastage on others.

The heterogeneity of services is crucial for efficient migration, scheduling, and resource allocation. This model employs Osmotic Computing (OC) for service classification. OCTRA-5G classifies services according to memory size, computational capacity, and delay sensitivity. Delay sensitivity is addressed through the numerology concept of 5G.

The model supports migration between the Edge Network (EN) and Public/Private Cloud environments. It facilitates the distribution of services to either the Edge or the Cloud. The Optimization Objective (OO) determines which services should be executed on the edge layer and which should be migrated to the Public/Private Cloud based on execution time and the availability of resources and memory on the Edge. The execution time of a service is influenced by computation time ($\mathcal{T}^s_{comp}$) and resource reservation time ($\mathcal{T}^s_{res}$), which is equivalent to the service computation time $\mathcal{T}^s_{trans}$.

A fitness function is derived from these factors to represent the concentration. The duration necessary to return the computed results of the service is specified by Eq. 3.7, encompassing the time required to transmit data from the device to the Edge (t_{sd}) and from the Edge back to the device (t_{ds}). In this context, d^s_{size} signifies the size of service s, while b^s_m indicates the assigned bandwidth.

$$\mathcal{T}^s_{trans} = t^s_{sd} + t^s_{ds} + \frac{d^s_{size}}{b^s_m} \tag{3.7}$$

To allocate resources to the requesting services, a model is developed that focuses on creating a memory and time model to allocate the required resources to all requesting services.

Let θ^s denote the computational strength [102] required per unit to perform service s with a size of d^s_{size}. The computational strength θ^s required for each task unit of data is generated within the range of 1×10^7 CPU cycles per M and 2×10^7 CPU cycles per M. This computational strength can be expressed as $\theta^s \times d^s_{size}$. The computation time required to process this service is represented by Equation 3.8:

$$\mathcal{T}^s_{comp} = \frac{\theta^s * d^s_{size}}{C_{res}} \tag{3.8}$$

Where C_{res} is the computing resource allocated by the ORS.

Thus, the total execution time $Exec^s_t$ of a service can be calculated as given in Eq. 3.9:

$$\mathcal{T}^{s}_{exec} = \mathcal{T}^{s}_{comp} + \mathcal{T}^{s}_{trans} \tag{3.9}$$

The threshold value for segregating services is taken as the maximum size available on the Edge. $\mathcal{M}_{th}$ is the max value of *size* lent by a UE as shown in Equation 3.10.

$$\mathcal{M}_{th} = max(size) \tag{3.10}$$

The threshold value for segregation based on computation time can be calculated using Equation 3.12.

$$\mathcal{C}_o = \sum_{i=1}^{s} \left(\mathcal{T}_{exec} \right)_i \tag{3.11}$$

$$\mathcal{C}_{th} = \frac{\mathcal{C}_o}{s} \tag{3.12}$$

Where 's' is the number of services at that moment at the OO.

Algorithm 3.1 Service Classification/Task Scheduling

Input: *s, size, threshold*
Output: *Set of microservices and macroservices*

Fetch s //Service requests
Initialize $sctr = 0$ //Microservice counter
Initialize $i = 0, j = 0$ //List indices
Calculate the memory and time using Eq. 3.7 and Eq. 3.8
Calculate the threshold values for the OO using Eq. 3.10 and Eq. 3.12
for each service s **do**
    if $\mathcal{T}_{comp} < \mathcal{C}_{th}$ **and** $d_{size} < \mathcal{M}_{th}$ **then**
        $sctr = sctr + 1$
        microservice_list[i] = s //Keep detected microservice on EN
        $i = i + 1$
    else if $\mathcal{T}_{comp} \geq \mathcal{C}_{th}$ **and** $d_{size} \geq \mathcal{M}_{th}$ **then**
        macroservice_list[j] = s //Add to macroservice list
        $j = j + 1$ //Shift to Public/Private Cloud
    end if
end for

In Algorithm 3.1, the models for memory and time are computed using threshold values derived from Equations 3.10 and 3.12. These threshold values facilitate the classification of services into *macroservices* and *microservices*. Services are categorized based on their

memory and time values, which are compared against the threshold values. Services with values below the threshold are classified as *microservices*, while those with values equal to or above the threshold are classified as *macroservice*. Services identified as *macroservices* are then allocated to either public or private clouds, whereas those identified as *microservices* are retained on the Edge Network (EN) and added to the list of *microservices*.

- *Resource Allocation:* After classifying services as *microservices* and *macroservices*, resources from the resource pool created in Resource Pooling are allocated to the services identified as *microservices*. The process for resource allocation to *microservices* is detailed in the RA algorithms. To identify the optimal RA mechanism, the First Come First Serve (FCFS), Shortest Job First (SJF), and Priority Scheduling (PS) methods are employed as outlined in Algorithm 3.2.

  In Algorithm 3.2, the `ResAllocation` function utilizes two nested loops, one for services and one for resources. The outer loop (indexed by i) iterates over the services, while the inner loop (indexed by j) iterates over the resources. The *rctr* variable represents the total number of available resources at that moment, and *sctr* denotes the total number of service requests. For each service, its required computation power is compared with the computation power offered by each resource until a resource with sufficient power is identified. Services are allocated resources based on the FCFS method. Once a suitable resource is found, it is allocated to the service, and the task proceeds; otherwise, the search continues until all resources are examined. If no suitable resource is found, the service is migrated to the public or private Cloud.

  To implement the Shortest Job First (SJF) algorithm, the services are sorted based on their required computation time $\mathcal{T}^i_{comp}$. For the Priority Scheduling (PS) algorithm, services are sorted according to their priority p^i_e.

Algorithm 3.2 Resource Allocation

Input: *microservices, resources, sctr, rctr, schedule_info*
Output: *microservices with allocated resources*

main
Fetch microservice details.
Fetch Resource details into r
FStart=current_time //FCFS start time initialized
FMS=RESALLOCATION(microservices, sctr, rctr, r) //First Come First Serve
FEnd= current_time-FStart
SStart=current_time //SJF start time initialized
Sort r into sr on the basis of $\mathcal{T}^i_{comp}$.
SMS=RESALLOCATION(microservices, sctr, rctr, sr) //Shortest Job First
SEnd= current_time-SStart
PStart=current_time //PS start time initialized
Sort r into pr on the basis of Priority and $\mathcal{T}^i_{comp}, d^i_{size}$.
PMS= RESALLOCATION(microservices, sctr, rctr, pr) //Priority Scheduling
PEnd= current_time-PStart
max_ms=max(FMS,PMS,SMS)
min_time= min(FEnd, SEnd, PEnd)
end main
function RESALLOCATION(microservices, sctr, rctr, res)
    set count=0
    while i $\leq$ sctr **do**
        Flag = 0
        i=0
        identify requirements $\mathcal{T}^i_{load}, d^i_{size}$
        j=0
        for j$\leq$ rctr **do**
            $\mathcal{T}^j_{comp} = \frac{\theta^j * d^s_{size}}{C_{res}}$
            if $\mathcal{T}^i_{comp} \leq \mathcal{T}^j_{comp} \&\& availability^j = true$ **then**
                Perform the task
                Set Flag=1
                count=count+1
            end if
            if Flag=0 **then**
                Add the microservice to the macroservice pool. //Migrate the service to the Public/Private Cloud.
            end if
            j=j+1
        end for
        i=i+1
    end while
    return count
end function

3.3 Performance Evaluation

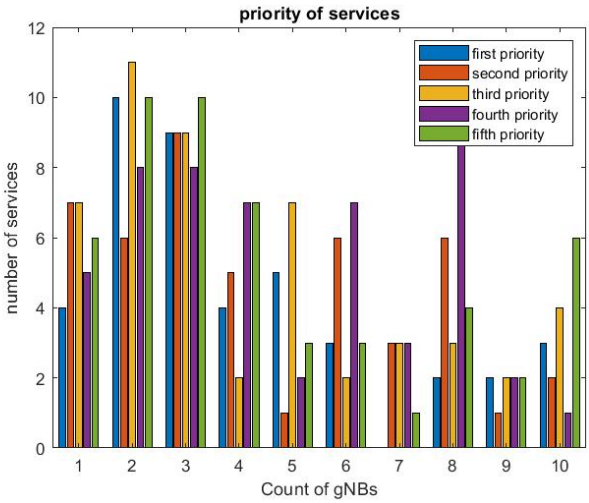
3.3.1 Simulation Settings

The effectiveness of OCTRA-5G is evaluated using MATLAB R2021b. Simulations are conducted on a PC with an Intel i5 CPU @ 1.19 GHz and 8 GB of RAM. The evaluation scenario includes multiple mMIMOs and UEs. Each edge server in the network is equipped with an OL, and each OL is associated with one OO. Table 3.2 shows the simulation settings used for OCTRA-5G.

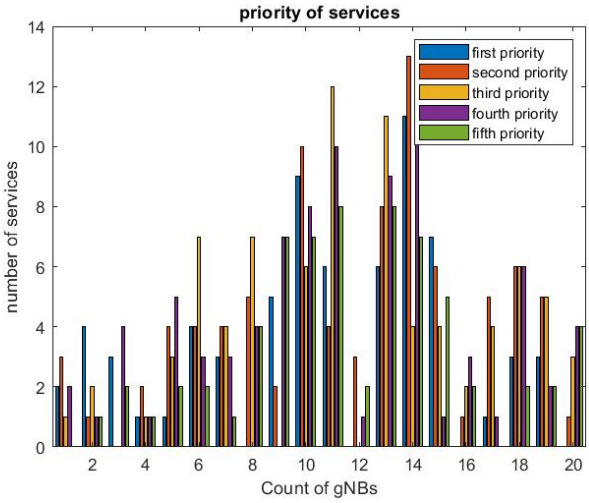
Table 3.2: Summary of Simulation Settings for OCTRA-5G

Parameter	**Value/Description**
Simulation Tool	MATLAB R2021b
Hardware	Intel i5 CPU @ 1.19 GHz, 8 GB RAM
Network Components	Multiple mMIMOs, UEs, gNBs
Maximum UEs per gNB	80
Area Coverage per gNB	500 meters radius
Types of Services	Microservices, Macroservices
Computation Requirements	Dynamically assigned for each service
Bandwidth Allocation Considerations	Numerology and service priority
Network Densities Tested	10, 20, and 30 gNBs
Metrics Evaluated	Connection type, service type

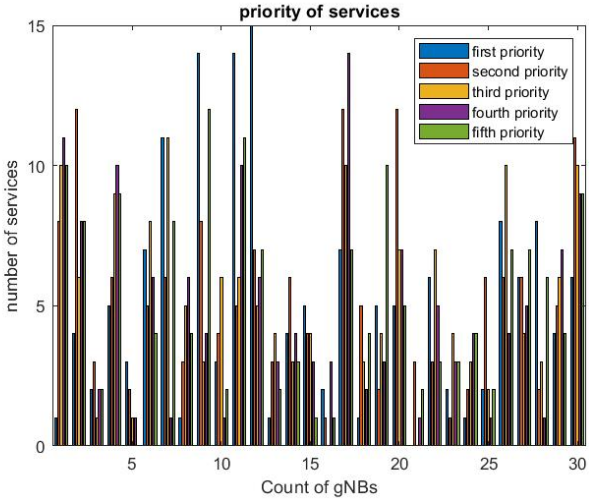
Each gNB can connect up to 80 UEs. For every connected UE, various parameters are computed, including the connection type (requesting service or lending resource), service type (*microservice or macroservice*), computation requirement, size requirement, and bandwidth. Bandwidth allocation for each service considers Numerology, which accounts for the priority of services assigned by the core network, as depicted in Figures 3.3a, 3.3b, and 3.3c. These figures illustrate the service request priorities for 10, 20, and 30 gNBs. The relevance of observing networks with 10, 20, and 30 gNBs shows the scalability and adaptability of the classification algorithms in varying densities of network infrastructure. For instance, in Figure 3.3a, gNB 7 has no service with priority 1. Services requiring higher priority receive more bandwidth with reduced delay. Each gNB covers an area with a radius of 500 meters, ensuring all connected UEs are within this range, as shown in Figure 3.4. This figure represents service requests categorized into *microservices* and *macroservices* within a 500-meter radius.



(a) Priority-wise list of service requests on each OO on a network of 10 gNBs.



(b) Priority-wise list of service requests on each OO on a network of 20 gNBs.



(c) Priority-wise list of service requests on each OO on a network of 30 gNBs.

Figure 3.3: Priority-wise list of service requests on each OO on a network.

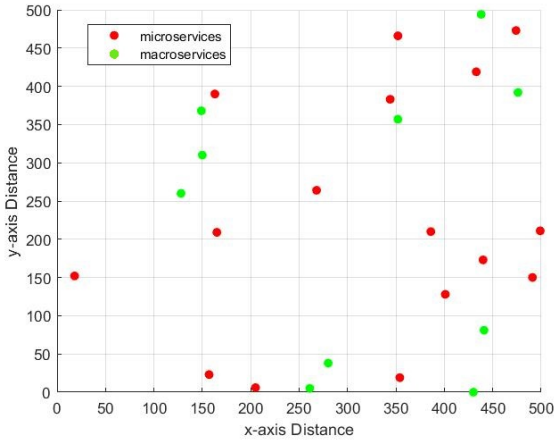


Figure 3.4: Segregated *microservices* and *macroservices* on 1 gNB.

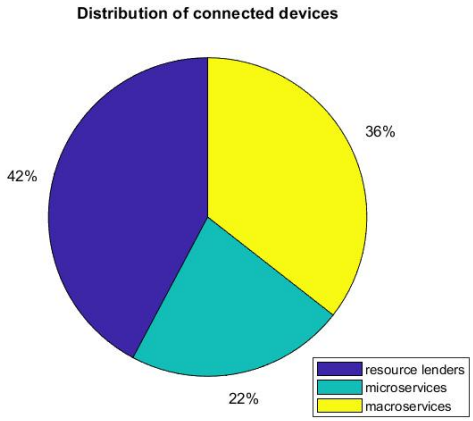
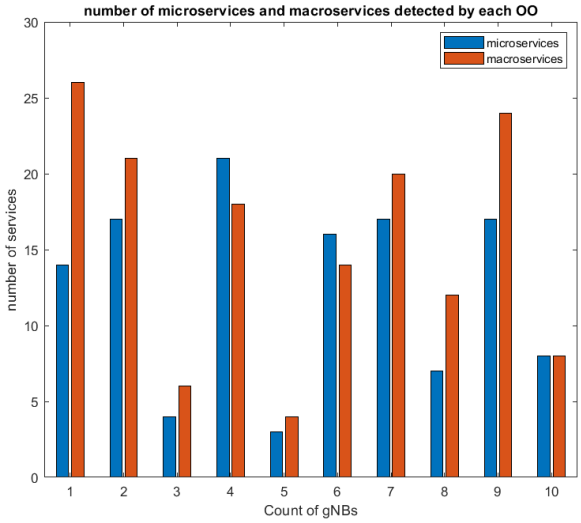
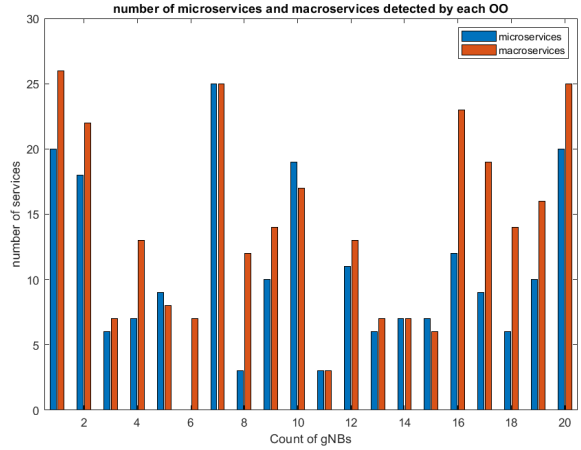


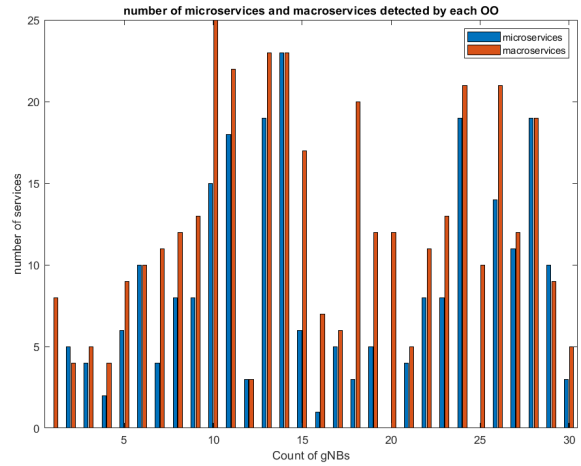
Figure 3.5: Distribution of connected devices on 1 gNB.



(a) *microservice* and *macroservice* requests on a network of 10 gNBs.



(b) *microservice* and *macroservice* requests on a network of 20 gNBs.



(c) *microservice* and *macroservice* requests on a network of 30 gNBs.

Figure 3.6: *microservice* and *macroservice* requests on a network of 10,20, and 30 gNBs.

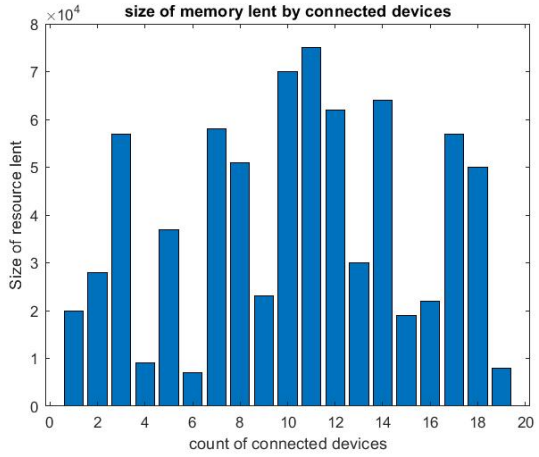
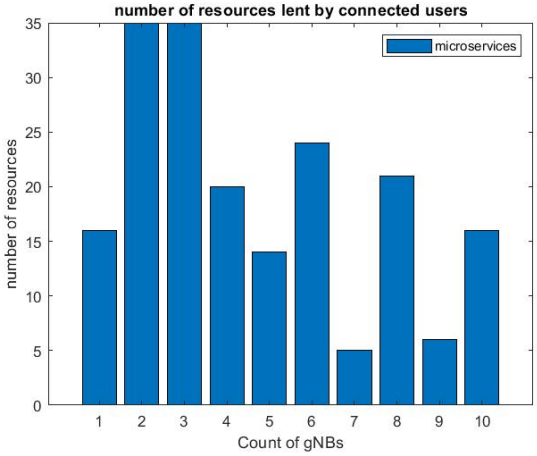
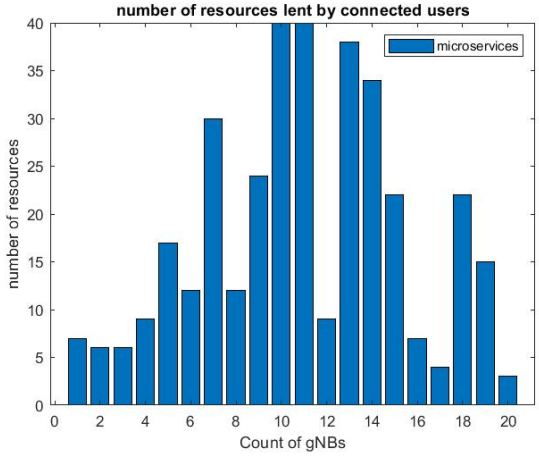


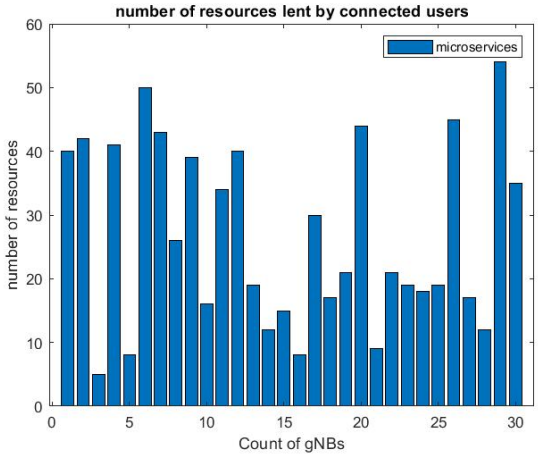
Figure 3.7: Storage resources lent by UE on a single OO.



(a) Number of resources lent by UE on a network of 10 gNBs.

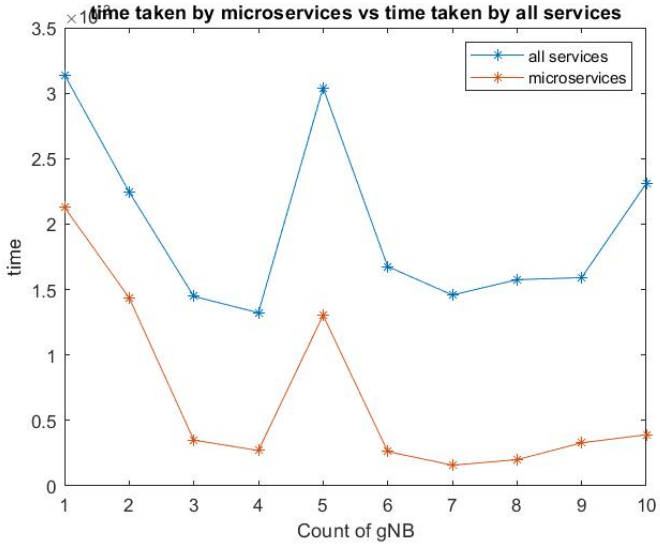


(b) Number of resources lent by UE on a network of 20 gNBs.

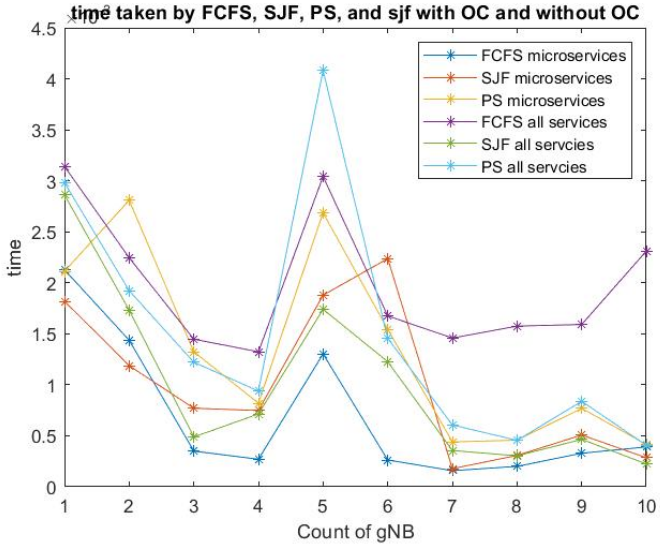


(c) Number of resources lent by UE on a network of 30 gNBs.

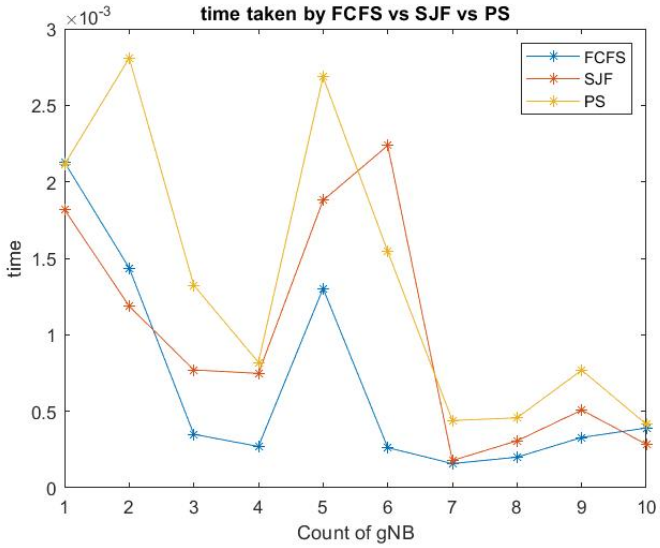
Figure 3.8: Number of resources lent by UE on a network.



(a) Time consumed by RA algorithm to allocate resources to all service requests vs all microservices.



(b) Time consumed by FCFS, SJF, PS with OCTRA-5G vs without OCTRA-5G.



(c) Time consumed by FCFS RA algorithm vs SJF RA vs PS RA Algorithm to allocate resources to *microservices* in all OOs.

Figure 3.9: Allocation of resources.

3.3.2 Performance Evaluation of Proposed Algorithms

- *Performance of Service Classification/Task Scheduling Algorithm:* Algorithm 3.1 classifies services into *microservices* and *macroservice*. Figure 3.5 illustrates the distribution of connected devices into resource lenders, *microservices*, and *macroservices* on a single gNB. By computing a threshold value using Equations 3.10 and 3.12, the algorithm evaluates each service's parameters against these threshold values. Services with parameter values below the threshold are categorized as *microservices*, while those with values equal to or exceeding the threshold are classified as *macroservice*. The results of this classification algorithm for networks with 10, 20, and 30 gNBs are presented in Figures 3.6a, 3.6b, and 3.6c, respectively. Figure 3.6a depicts a network consisting of 10 gNBs, with varying numbers of service requests per gNB. In this figure, the service requests are divided into *microservices* and *macroservices* for each gNB. For example, gNB 1 has 18 *microservices* and 11 *macroservices* out of a total of 29 service requests.

- *Resource Pooling:* From the UEs and IoT devices connected to a gNB, the Resource Pooling algorithm collects available resources from resource lenders. Resources are voluntarily lent by users, as some connected UEs may lack sufficient resources to execute the required services, while others may have surplus resources not needed at that moment. Figure 3.7 illustrates the storage resources lent by connected users to a single gNB. Figures 3.8a, 3.8b, and 3.8c depict the number of users lending resources across a set of 10, 20, and 30 gNBs, respectively. evaluation, 10 gNBs are utilized to execute First Come First Serve (FCFS), Shortest Job First (SJF), and Priority Scheduling (PS) algorithms.

- *Resource Allocation Algorithm: Microservices* are assigned resources from the ORS (Resource Pool) according to the allocation strategy detailed in Algorithm 3.2. This strategy considers First Come First Serve (FCFS), Shortest Job First (SJF), and Priority Scheduling (PS) approaches. If a resource compatible with a microservice is unavailable in the resource pool, the microservice is relocated to either a public or private cloud environment.
  Figure 3.9a illustrates the results of Resource Allocation (RA) for a scenario involving 10 gNBs, each connected to a different number of UEs. The comparison focuses on the time consumed for RA to all services versus RA to *microservices*. The results indicate that OCTRA-

5G yields reduced time consumption and enhanced resource utilization on the Edge Node (EN). OCTRA-5G improves efficiency by segmenting services into *microservices* and *macroservice*. It allocates resources based on *microservices* only, ensuring that available resources on the Edge are utilized effectively. In contrast, the traditional RA algorithm does not differentiate between *microservices* and *macroservices*, leading to increased time consumption as it assesses all services against all available resources, including large services.

Figure 3.9b presents a comparison between OCTRA-5G and Traditional RA for First Come First Serve (FCFS), Shortest Job First (SJF) and Priority Scheduling (PS) scheduling algorithms. The results demonstrate that OCTRA-5G outperforms traditional methods, with reduced time consumption observed across FCFS, SJF, and PS. Specifically, OCTRA-5G with FCFS scheduling exhibits superior efficiency compared to other algorithms.

Figure 3.9c displays the time consumption for FCFS, SJF, and PS scheduling algorithms. FCFS is noted for its minimal time consumption compared to SJF and PS. The SJF algorithm requires additional time to identify the next shortest job for resource allocation, while PS spends extra time sorting *microservices*. This figure highlights the differences in time consumption among FCFS, SJF, and PS.

Tables 3.5 and 3.4 present the efficiency results of OCTRA-5G for scenarios involving 10 gNBs and 30 gNBs, respectively. The relative efficiency is calculated using the Equation 3.13:

$$\eta = \frac{\text{Performance (Without OCTRA-5G) - Performance (OCTRA-5G)}}{\text{Performance (Without OCTRA-5G)}} \times 100 \qquad (3.13)$$

These tables detail the number of services per gNB and compare the time consumed with and without the OO. The efficiency of OCTRA-5G is shown to vary, with a 20.679% improvement on gNB 1 and a 95.57% improvement on gNB 10, illustrating the range of efficiency gains achieved with OCTRA-5G compared to traditional Resource Allocation methods.

Table 3.3 depicts the total time consumed by OCTRA-5G vs Traditional Resource Allocation algorithm. The results show that the use of OCTRA-5G with FCFS, PS, and SJF leads to less time consumption as compared to traditional RA on FCFS, PS, and SJF resource allocation, respectively.

Table 3.3: Time taken by all algorithms with and without OC.

Algorithm	Time
FCFS with OC	0.0068278
FCFS without OC	0.019809
SJF with OC	0.009176
SJF without OC	0.010109
PS with OC	0.013376
PS without OC	0.014907

Table 3.4: Efficiency of Resource Allocation with OO with 30 gNBs.

BS	Total Services	Time Without OC	Time With OC	Efficiency With OC (%)
1	80	0.043231	0.0093179	78.446
2	80	0.0087744	0.0045554	48.083
3	15	0.0065994	0.0009573	85.494
4	80	0.0060103	0.0007913	86.834
5	15	0.0095168	0.0030484	67.968
6	80	0.0061897	0.0014222	77.023
7	80	0.0062232	0.0007499	87.950
8	45	0.0077554	0.0006185	92.025
9	80	0.0060350	0.0004091	93.221
10	32	0.0080689	0.0003995	95.049
11	80	0.0073035	0.0007073	90.316
12	80	0.0063367	0.0007124	88.758
13	32	0.0064107	0.0003518	94.512
14	32	0.0061171	0.0003247	94.692
15	32	0.0071148	0.0006262	91.199
16	15	0.0066993	0.0006501	90.296
17	80	0.0073349	0.0003992	94.558
18	32	0.0108070	0.0004170	96.141
19	45	0.0081600	0.0007053	91.357
20	80	0.0091701	0.0006276	93.156
21	15	0.0087356	0.0006152	92.958
22	45	0.0102970	0.0007514	92.702
23	32	0.0096194	0.0004624	95.193
24	32	0.0111160	0.0005010	95.493
25	32	0.0118450	0.0006167	94.794
26	80	0.0107080	0.0006415	94.009
27	45	0.0097762	0.0006019	93.843
28	32	0.0097379	0.0003596	96.307
29	80	0.0101960	0.0006921	93.212
30	80	0.0100820	0.0004707	95.331

On analyzing the complexity of the algorithms for resource allocation the time complexity for FCFS, SJF, and PS, the FCFS has lowest complexity of O(sctr×rctr) as compared to other algorithms which have O(rctr log rctr+sctr×rctr) and O(rctr log rctr+sctr×rctr) respectively. The time complexity of these algorithms is O(1), O(rctr), and O(sctr) . The linear complexity of FCFS proves computational efficiency of the algorithm.

Table 3.5: Efficiency of Resource Allocation with OO with 10 gNBs.

BS	Total Services	Time Without OC	Time With OC	Efficiency With OC (%)
1	32	0.0062439	0.0049527	20.679
2	32	0.0031334	0.0019438	37.965
3	80	0.0025956	0.0017626	32.093
4	32	0.0019700	0.0004490	77.208
5	45	0.0012024	0.0001248	89.621
6	15	0.0014314	0.0001752	87.760
7	80	0.0010654	0.0004364	59.039
8	15	0.0010276	0.0001821	82.279
9	15	0.0008259	0.0001062	87.141
10	32	0.0034812	0.0001542	95.570

3.4 Conclusion

5G technology offers markedly enhanced data throughput and minimal latency. Mobile Edge Computing (MEC) is a crucial technology that improves real-time processing capabilities, reduces the burden on the core network, and facilitates prompt data processing, thus achieving high data rates and low latency. Traditionally, the Resource Allocation (RA) algorithm processes all service requests for resources at the Edge Node (EN), which may result in increased processing time due to potential mismatches between service requirements and available resources at the EN.

This chapter presents the OCTRA-5G Framework for task scheduling and resource allocation with Osmotic Computing (OC). The notion of OC is based on the osmosis process in chemistry. In OCTRA-5G, an Osmotic Layer (OL) is incorporated at the edge layer of each network. Each OL includes an OO, which comprises three components: (i) the Osmotic Resource Database (ORD), which records information about all lent resources, (ii) the Osmotic Service Segregator (OSS), which classifies services into *microservices* and *macroservices* and migrates *macroservices* to

Public/Private Cloud, and (iii) the Osmotic Resource Scheduler (ORS), which allocates resources to *microservices* by searching for compatible resources in the ORD. The microservice is migrated to the next OL if the required resource is unavailable in the ORD.

Validation of the results was conducted using sets of 10, 20, and 30 gNBs, demonstrating a reduction in RA time consumption and a decrease in latency ranging from 20.679% to 95.57%. The performance of OCTRA-5G was evaluated against traditional RA algorithms such as First-Come-First-Served (FCFS), Shortest Job First (SJF), and Priority Scheduling (PS). The results indicate that OCTRA-5G exhibits lower time consumption compared to traditional RA algorithms. Among the algorithms tested within the OCTRA-5G framework, FCFS proved to be more efficient than SJF and PS.

Chapter 4

OsCoMIT: Osmotic Computing-based Service Management for Intelligent Transportation Systems in 5G Networks[1]

In the rapidly evolving domain of Intelligent Transportation Systems (ITS), integrating advanced communication technologies and efficient resource management strategies is paramount. This chapter presents a novel framework based on Osmotic Computing (OC) to address the critical challenges of task segregation and resource allocation within ITS, leveraging the capabilities of 5G networks. The framework is structured into three distinct layers: the Core Network Layer, the Edge Computing Layer, and the Intelligent Transportation Layer. The framework enhances performance and reliability by employing OC at the Edge Network Layer through effective task segregation and intelligent decision-making. Additionally, the proposed Proportional Fairness algorithm ensures optimal resource allocation for microservices, further improving the efficiency and robustness of the system. Through these contributions, the proposed framework aims to significantly advance the functionality and applicability of ITS in the 5G era.

[1]The contents of this chapter are published as: Kaur, A., Saxena, S. & Kumar, R. OsCoMIT: Osmotic computing-based service management for intelligent transportation systems in 5G network. Cluster Comput 27, 5403–5421 (2024). https://doi.org/10.1007/s10586-023-04217-1

4.1 Proposed OsCoMIT Framework

To handle task scheduling and resource allocation in the Intelligent Transportation system (ITS), **Os**motic **Com**puting-based service **M**anagement for **I**ntelligent **T**ransportation systems framework is proposed. Its successful implementation for the management of Intelligent Transportation Systems (ITS) at the edge network necessitates the following components:

- *Edge Computing Infrastructure*: Essential for OC implementation, this includes edge devices such as gateways, sensors, and edge servers capable of local data processing and communication with each other and the cloud.
- *High-Speed Network Connectivity*: Crucial for real-time data exchange between edge devices and the cloud, necessitating a high-speed network connection.
- *Standardized Protocols*: Data exchange and communication must adhere to standardized protocols to ensure interoperability among various peripheral devices and cloud services.
- *Data Processing Capabilities*: Edge devices should possess sufficient capabilities for local data processing and analysis to enhance the overall system performance.

4.1.1 OsCoMIT System Model

The OsCoMIT framework is shown in Figure 4.1. It is divided into three layers: Core Network Layer (Layer 1), Edge Computing Layer (Layer 2), and Intelligent Vehicle Layer (Layer 3). Layer 1 is the public/private cloud layer. Layer 2 is the edge layer. It contains Osmotic Orchestrator (OO) which controls the osmotic process. OO has service segregation module, service mapping repository, and osmotic resource database module. Layer 3 is the sensor layer containing intelligent vehicles. The Intelligent Vehicle layer (layer 3) encompasses Intelligent Vehicles (V_e) where $e \in \{1, 2, ..., n\}$, with n representing the maximum number of devices connected to the gNB (M_j) where $j \in \{1, 2, ..., m\}$ denotes the edge-id of the network. These Intelligent Vehicles gather data from various sensors, including speed, battery, ultrasonic parking, electronic battery, torque, throttle position, accelerator, air pressure, GPS, and radar distance sensors. The gathered data is transmitted to the gNB at the edge layer (layer 2) utilizing the Message Queue Telemetry

Transport (MQTT) protocol. Each gNB is equipped with an Osmotic Orchestrator (OO). Upon arrival at the edge layer, services are directed to the service segregation module of the OO, which categorizes the services into *macroservice* and *microservice*. The *macroservices* are migrated to the core layer (layer 1), while *microservices* are routed to the Service Mapping Repository (SMR). In the SMR, services are matched with available resources in the osmotic resource database according to specific requirements based on threshold values. If a resource cannot be allocated to a service, that service is then migrated to the cloud network.

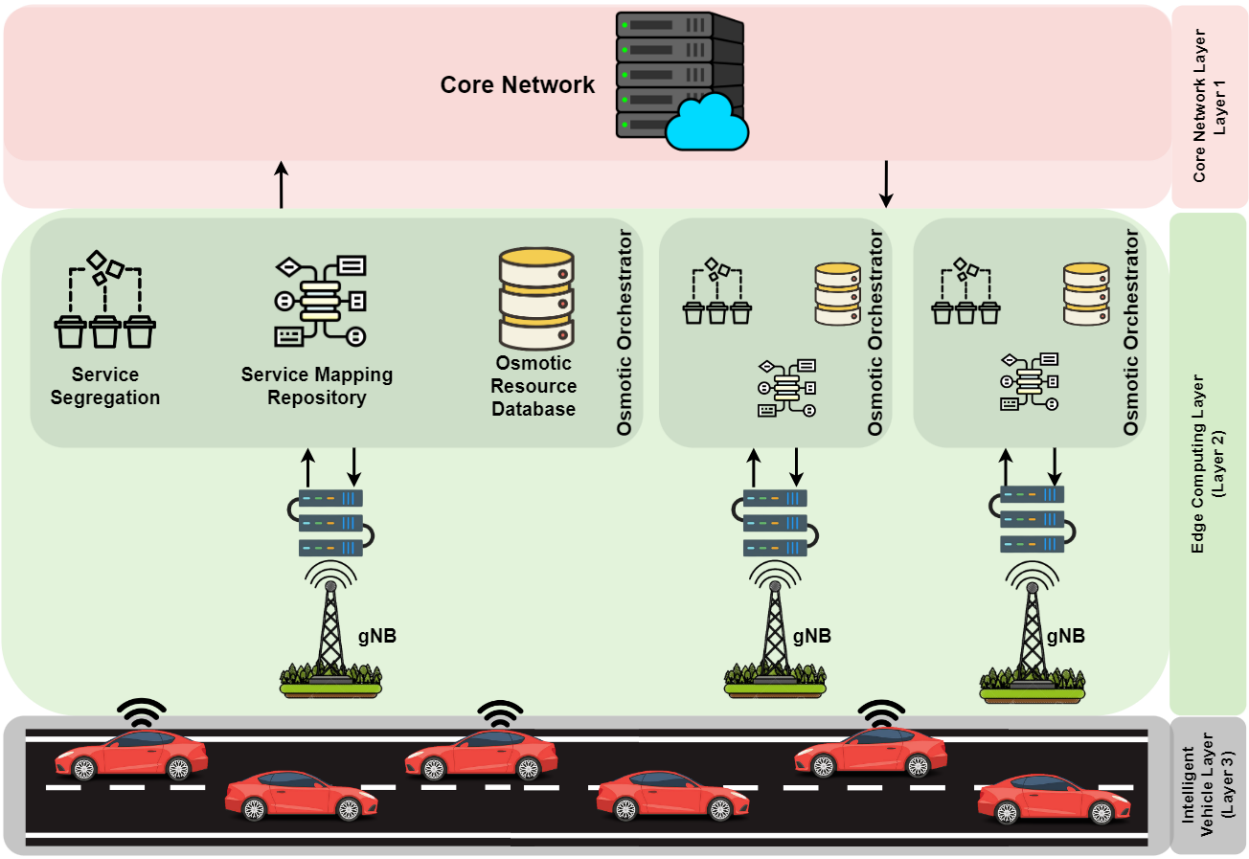


Figure 4.1: Proposed network framework for ITS using OC

Let S be the set of services requested by vehicle V_i, i = 1, 2, ... ,m from the edge of j services as shown in Equation 4.1. Here, S_{ij} is the jth service, j= 1, 2, ..., n, requested by vehicle i.

$$S = \begin{bmatrix} S_{1,1} & S_{1,2} & ... & S_{1,n} \\ S_{2,1} & S_{2,2} & ... & S_{2,n} \\ S_{3,1} & S_{3,2} & ... & S_{3,n} \\ ... & & & \\ S_{i,1} & S_{i,2} & ... & S_{i,j} \end{bmatrix} \tag{4.1}$$

Based on the threshold values received from the Osmotic Resource Database (ORD) (Algorithm 4.2), the service is termed as *microservice*(mic) or *macroservice*(mac) as shown in Equation 4.2 and Equation 4.3, respectively.

$$X_a = \text{mic if } S_{ij} \leq Th \tag{4.2}$$

$$Y_b = \text{mac if } S_{ij} \geq Th \tag{4.3}$$

4.1.2 Edge Device Identification and Service Generation at Intelligent Vehicle Layer

Intelligent vehicles generate data that encompasses services required by Intelligent Transportation Systems (ITS). Each service contains vehicle ID, the time of production, priority, and additional sensor details. The service set S, as defined in Equation 4.1, consists of *macroservices* and *microservices* generated by various sensors from different vehicles. These services can request data processing, computation, storage, *etc*. When a device on the edge network requests a data service, it sends a request to the gNB. The request includes information about the type of service (e.g., data retrieval, storage, processing, GPS services, and processing of other sensory data) needed and any relevant parameters or data (e.g., type of service requested, the size and format of the data being requested or sent). The data produced by each intelligent vehicle is denoted as shown in Equation 4.4

$$S_i(data(W, S, B, UV, UT, T, TH, A, G, F), req(C, M)) \tag{4.4}$$

The variables represent different sensors and requested resources as follows: *W* denotes the wheel speed sensor, *S* represents the speed sensor, *B* indicates the battery sensor, *UV* stands for the ultrasonic parking sensor voltage, *UT* signifies the ultrasonic parking sensor temperature, *T* is the temperature sensor, *TH* is the throttle sensor, *A* is the accelerator sensor, *G* corresponds to the GPS request, and *F* denotes the fluid level sensor. The requested resources include *C* for computation resources and *M* for memory requests. Figure 4.2 illustrates a snippet of a service request sent by an intelligent vehicle to the edge network.

```
{
  "vehicle_id": vehicle_id,
  "timestamp": timestamp,
  "request.Sensordata":
    { "wheelSpeedSensor": wwheelSpeedSensorss,
      "wheelSpeedSensorTemperature":wheelSpeedSensorTemperature,
      "speedSensor":speedSensor,
      "batterySensor":batterySensor,
      "ultrasonicParkingSensor":ultrasonicParkingSensor,
      "torqueSensor":torqueSensor,
      "throttleSensor":throttleSensor,
      "acceleratorSensor":acceleratorSensor,
      "fluidLevelSensor":fluidLevelSensor
    },
  "request.required":
  {
    "cpu_req": req_cpu,
    "ram req": req ram,
  }
}
```

Figure 4.2: Service request by the intelligent vehicle to the edge network

4.1.3 Edge Computing Layer

With the proliferation of vehicles and IoT devices, the necessity for executing sensitive data has grown significantly. This layer addresses the service requests generated by vehicles by utilizing the resources available at the network edge, thereby alleviating the load on public and private clouds

and reducing network traffic. Upon the arrival of a service request at the edge network, it is routed to the OO for processing.

Algorithm 4.1 Service Segregation

Input: S_i *// S_i is the service request sent by intelligent vehicle*
Output: *microservices, macroservices*
// Step 1: Fetching the threshold values from the resources available in ORD.
$CPU_{th}, MEM_{th} \leftarrow$ FindThreshold(Database)
$S \leftarrow S_i(\text{data}(W, S, B, UV, UT, T, TH, A, G, F), \text{req}(C, M))$
$data \leftarrow \text{data}(W, S, B, UV, UT, T, TH, A, G, F)$
$request \leftarrow \text{req}(C, M)$
for $i = 1$ **to** m **do**
    for $j = 1$ **to** n **do**
        if *data* **then**
            $d_{sn} \leftarrow \text{sensor}_{data}$
        end if
        if *request* **then**
            // Step 2: Calculate the size of requested resources.
            $C_{req} \leftarrow \lceil \frac{R_{device}}{P_{cpu}} \rceil$ *// Calculate Computational Resource*
            $R_{req} \leftarrow \sum_{d_i \in D_i} d_i + \sum_{d_{\text{int}} \in D_{\text{int}}} d_{\text{int}} + \sum_{d_o \in D_o} d_o + \sum_{d_m \in D_m} d_m$ *// Calculate Memory Resources*
            // Step 3: Segregating the services
            if $C_{req} < CPU_{th}$ **and** $R_{req} < MEM_{th}$ **then**
                $X_a \leftarrow S$ *// service is microservice*
                ResAllocation(microservice, DatabaseResource)
            else
                $Y_b \leftarrow S$ *// service is macroservice*
                Migrate to the cloud layer
            end if
        end if
    end for
end for

4.1.4 Osmotic Orchestrator (OO)

OO is responsible for managing resources and facilitating communication between edge devices and cloud resources. It comprises three primary components: (i) Service Segregation, (ii) ORD, and (iii) SMR. These components collectively maintain the resource database to enhance the efficiency of Mobile Edge Computing (MEC), allocate resources to *microservices*, and migrate *macroservices* to the cloud network. The three components are detailed as follows:

Algorithm 4.2 ORD and finding the threshold value for available resources

Input: IoT_{device} *//IoT_{device} are the IoT devices connected to the edge network*
Output: CR_{th}, MR_{th}, *Database*

function DATABASERESOURCE(Devices)
    Initialize Database
    for all device in IoT_{device} **do**
        Read ID, $Bandwidth$, CR_l, MR_l from service
        Database $\leftarrow append(ID, Bandwidth, CR_l and MR_l)$
        return Database
        function FINDTHRESHOLD(Database)
            read CR_l for device from Database
            $CPU_{th} = \max_{i=1}^{k}(\text{CR}_l)$ *//Find the threshold value for CPU resource*
            read MR_l
            $MEM_{th} = \max_{i=1}^{k}(\text{MR}_l)$ *//Find the threshold value for MEM resource*
            return CR_{th}, MEM_{th}
        end function
        Devices $\leftarrow IoT_{device}$ *//Read device information*
        Database $\leftarrow$ DATABASERESOURCE(Devices)
        $CPU_{th}, MEM_{th} \leftarrow$ FindThreshold(Database)
    end for
end function

- *Service Segregation:* When an edge server receives a request, it can either process the request locally or forward it to a cloud server based on the current resource availability on the edge. Segregating *macroservices* and *microservices* within an edge network can enhance system performance and reliability. The procedure for identifying different service types is detailed in Algorithm 4.1, which includes three primary steps. The first step involves fetching threshold values based on the available resources, as outlined in Algorithm 4.2. These threshold values represent the peak resource capacities at the edge. Using peak values ensures optimal resource utilization on the edge, as the system allocates available resources to requesting devices and avoids handling services that require more substantial computation resources or memory, thereby saving time. The second step determines the type, size, or power of the resources needed for the request. In the final step, the values from the request are compared against the threshold values from the resource database, categorizing them into *macroservices* and *microservices*. *Microservices*, with requests within the threshold limits, are allocated resources according to Algorithm 4.3. In contrast, *macroservices*, requiring more significant resources than available on the edge, are migrated to the cloud network for processing.

The maximum number of CPUs (C) that a device in the edge network can support is determined using Equation 4.5. This calculation is based on the power consumption of the device (P_{device}), the available power (P_{avail}) of the device, and the power consumption per CPU (P_{cpu}).

$$C = \frac{(P_{avail} - P_{device})}{P_{cpu}} \tag{4.5}$$

In Equation 4.5, it is assumed that the power consumption per CPU and the power consumption of the device are constant and known values.

The total number of CPUs required by an intelligent vehicle from the edge network is calculated using Equation 4.6,

$$C_{req} = \lceil \frac{R_{device}}{P_{cpu}} \rceil \tag{4.6}$$

where C_{req} denotes the maximum number of CPUs required by the intelligent vehicle, R_{device} represents the resource requirement in terms of CPU capacity, and P_{CPU} is the processing power of a single CPU. For instance, if a device requests a processing power of 3500 units, and each CPU has a capacity of 1000 units, then the total number of CPUs required is:

$$C_{req} = \lceil \frac{3500}{1000} \rceil = 4$$

Total available Random Access Memory (RAM) at the edge network, R_{avail}, which represents the total RAM available for allocation to requesting devices, is calculated as shown in Equation 4.7. Here, R_{Edge} is the total amount of RAM available for allocation to other processes on the network, and R_{res} is the amount of memory already allocated.

$$R_{avail} = R_{Edge} - R_{res} \tag{4.7}$$

$$R_{dev} = min(R_{req}, R_{avail}) \tag{4.8}$$

To prevent the device from surpassing its maximum memory limit and ensure it does not exceed the available memory on the edge network, Equation 4.8 is employed. This Equation calculates the total memory requested by the device, denoted as R_{req}, using Equation 4.9.

$$R_{req} = \sum_{d_i \in D_i} d_i + \sum_{d_{\text{int}} \in D_{\text{int}}} d_{\text{int}} + \sum_{d_o \in D_o} d_o + \sum_{d_m \in D_m} d_m \tag{4.9}$$

Consider the following sets: D_i for input data, D_{int} for intermediate data, D_o for output data, and D_m for metadata. Let d_i denote the size of the data element d_i. The formula calculates the total memory requirements by adding the sizes of all data elements within each set. This approach provides an estimate of memory needs based on the sizes of different data elements involved in the service.

- *Osmotic Resource Database (ORD):* IoT devices connected to gNBs can allocate idle computational or storage resources to the edge network. Intelligent vehicles can utilize these resources when their resources are insufficient. The resource management module of the database oversees the resources contributed by IoT devices within the edge network. This database monitors these resources using various parameters. Here, the key parameters include:
  - *Device Identification:* Each device should be uniquely identified to facilitate effective management of resources allocated by the device. Identification can be accomplished through the use of device-specific IDs, MAC addresses, or IP addresses.
  - *Bandwidth:* This parameter refers to the volume of data transmitted over the network at any given moment. It determines the capacity for data transfer between edge devices and the network.
  - *Battery Life:* This denotes the operational duration of a device on a single charge. It is particularly critical for battery-operated devices such as sensors and wearables.
  - *Processing Power:* This indicates the capability of edge devices to process data and execute algorithms locally. It is crucial for applications requiring real-time processing with minimal delay, such as object recognition and natural language processing.

– *Memory:* This represents the volume of data that can be stored locally on edge devices. Sufficient memory is essential for applications that need to handle large datasets, such as machine learning algorithms.

Real-time monitoring of the resources allocated by each device is crucial. This involves tracking CPU usage, memory usage, storage, and network utilization. Effective monitoring assists in identifying available resources, detecting potential issues with devices, and implementing necessary corrective measures. Threshold values for computation and memory resources are determined using Equations 4.10 and 4.11, respectively. In these equations, CR_l and MR_l represent the amounts of CPU and memory resources allocated by IoT devices.

$$CPU_{th} = \max_{i=1}^{k}(\mathbf{CR}_l) \tag{4.10}$$

$$MEM_{th} = \max_{i=1}^{k}(\mathbf{MR}_l) \tag{4.11}$$

Algorithm 4.3 Resource Allocation

Input: microservices, Database, IoT_{device}
Output: Resources allocated to microservices
function RESALLOCATION(microservice, DatabaseResource)
    for each $S_i \in S$ **do**
        for each $D_j \in D$ **do**
            // Calculate required SR and CR for S_i based on its specifications.
            $CR_{score_i} = \left(\frac{CR_{req_i}}{CR_{avail_j}}\right)^2$
            $SR_{score_i} = \left(\frac{SR_{req_i}}{SR_{avail_j}}\right)^2$
        end for
    end for
    $Score_i = \min_j \left(CR_{score_i} + SR_{score_i}\right)$ // Find the minimum score
    Allocate resources to microservice.
end function

- *Service Mapping Repository using the proposed Proportional Fairness (PF) algorithm:* The resource allocation procedure, described in Algorithm 4.3, employs the PF approach to assign resources to *microservices*. This algorithm is executed for each detected *microservice*. It requires three input parameters: the Resource Database, edge devices, and *microservices*, and outputs the allocated resource for the respective *microservice*.

The process involves iterating through the ORD and computing a score for each edge device as per Equation 4.12. The edge device that yields the minimum score is assigned to the intelligent vehicle. The selection of the minimum score is crucial as it minimizes internal fragmentation by ensuring that the allocated resources are close to the requested amount. This approach effectively mitigates resource wastage during the allocation process. The PF algorithm is designed to minimize internal fragmentation and to distribute resources equitably based on user requirements. A lower score indicates a more effective resource allocation, which enhances both fairness and efficiency in the allocation process.

$$Score_i = \min_j \left(\left(\frac{CR_{req_i}}{CR_{avail_j}} \right)^2 + \left(\frac{SR_{req_i}}{SR_{avail_j}} \right)^2 \right) \qquad (\forall j = 1, 2, \ldots, n) \tag{4.12}$$

where, CR_{req_i} is the computation resource requested, CR_{avail_j} is the available computation resource in the database, SR_{req_i} is the requested memory resource, and SR_{avail_j} is the available memory resource in the ORD database.

Table 4.1: Summary of Key notations

Parameter	**Description**
S	Set of service request
S_j	j^{th} Service request by i^{th} intelligent vehicle
C_{max}	CPUs requested by intelligent vehicle
R_{req}	RAM requested by intelligent vehicle
C_{req}	Computation requested by intelligent vehicle
C	Maximum number of CPUs supported by edge device
P_{device}	Power consumption of the device
P_{avail}	Available power
R_{avail}	Total RAM available at the edge network
R_{Edge}	RAM available to be allocated to other processes
P_{cpu}	Power consumption per CPU
R_{res}	Amount of memory already allocated
CR	Computation Score
SR	RAM Score

Although the notations have been explained in their respective places, Table 4.1 summarizes all the mathematical notations.

Figure 4.3: Area selected for network traffic simulation in SUMO

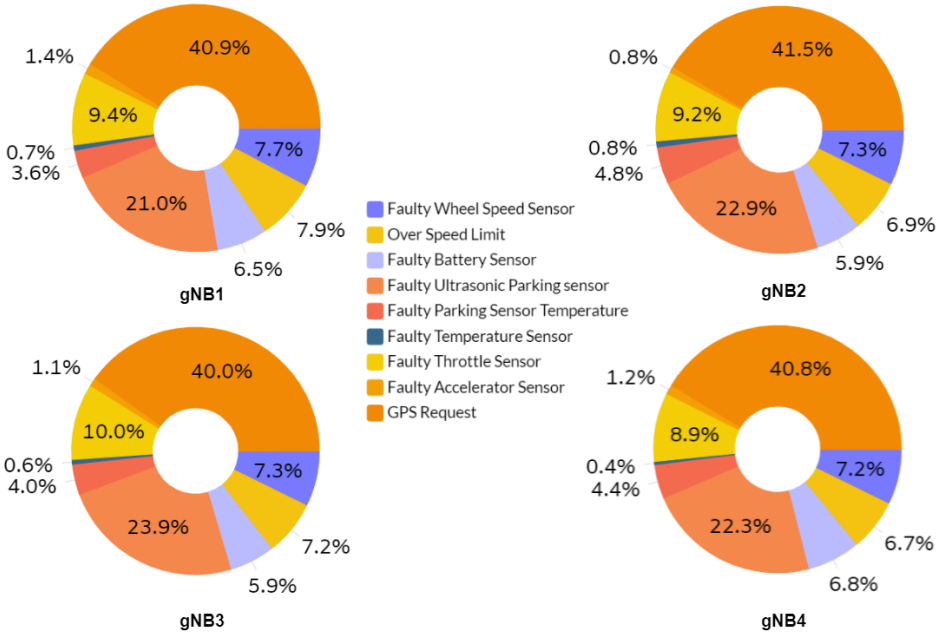


Figure 4.4: Type and number of sensor faults detected at the edge network

4.2 Performance Evaluation

The evaluation of the proposed framework utilizes SUMO and Python for simulations. These simulations are performed on a computer equipped with an Intel i5 CPU @ 1.19 GHz and 8 GB RAM. For the network simulations, a map of Delhi, India, as depicted in Figure 4.3, is used to generate the dataset for Intelligent Transportation Systems (ITS). Figure 4.5 illustrates the process of data set generation with the interaction between the different modules. SUMO (Simulation of

Table 4.2: Simulation Settings

Parameter	Description
Simulation Tools	SUMO (Simulation of Urban MObility), Python
Hardware Specifications	Intel i5 CPU @ 1.19 GHz, 8 GB RAM
Simulation Environment	Map of Delhi, India, generated using OSM (OpenStreetMap) web wizard
Dataset	ITS dataset generated with 172 vehicles equipped with ten different sensors each
Route Length	1824.15 km
Network Configuration	4 gNBs connected to intelligent vehicles and IoT devices
Sensor Types	Wheel Speed Sensor, Wheel Speed Temperature Sensor, Speed Sensor, Battery Sensor, Ultrasonic Parking Sensor Voltage, Ultrasonic Parking Temperature Sensor, Torque Sensor, Throttle Sensor, Acceleration Sensor Voltage, Fluid Level Sensor
Service Requests	Categorized into microservices and macroservices based on edge network availability
Fault Management	Faulty sensors are identified and addressed; notifications are sent for faults, and GPS data is transmitted upon request
Fault Types	Deviation from predefined operational ranges; specific faults include GPS, wheel speed, and ultrasonic parking sensor faults
Simulation Outputs	Number of microservices and macroservices detected at the edge network

Urban MObility) simulates the ITS environment with predetermined initiatives. The OSM (OpenStreetMap) web wizard facilitates modeling of the physical world. The OSM web wizard extracts the real map from OSM, followed by node processing that generates UEs (User Equipments). A network traffic generator then produces a traffic trace for each UE, creating a *.sumocfg* file, which is fed into the SUMO simulator. For network simulations, a map of Delhi, India, is used, generating the data set on ITS with a route length of 1824.15 km and a total of 172 vehicles, each equipped with ten different sensors. Table 4.2 lists the simulation settings used for the implementation of the proposed framework.

The network considered for evaluation consists of four gNBs. Each gNB is connected to a set of intelligent vehicles and IoT devices. Intelligent vehicles connected to the gNBs submit service requests to the edge network, while IoT devices provide resources to the edge network. Service

Table 4.3: Type of Sensors used.

Sr no.	Sensor Type
1	Wheel Speed Sensor
2	Wheel Speed Temperature Sensor
3	Speed Sensor
4	Battery Sensor
5	Ultrasonic Parking Sensor Voltage
6	Ultrasonic Parking Temperature Sensor
7	Torque Sensor
8	Throttle sensor
9	Acceleration Sensor Voltage
10	Fluid Level Sensor

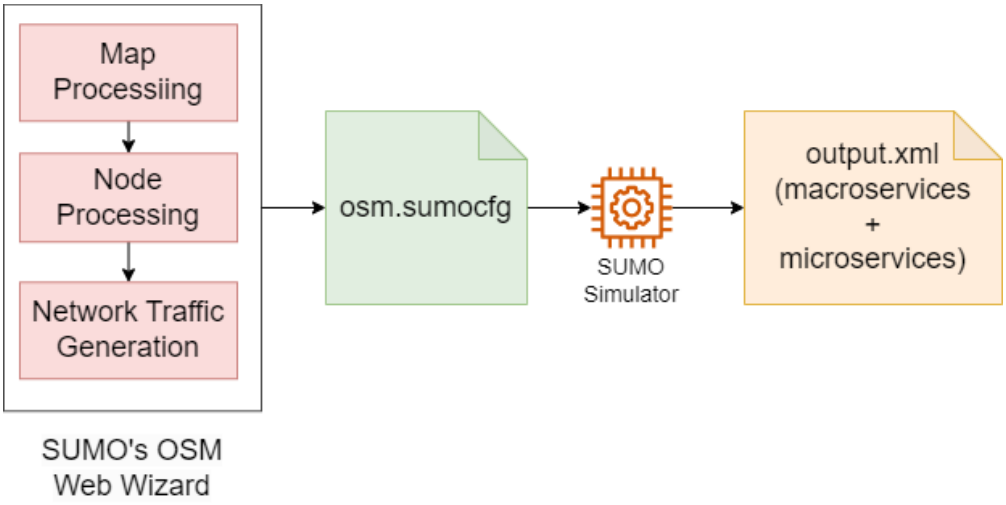


Figure 4.5: Steps to generate data using OSM Web Wizard

requests from ITS received by the edge server are categorized based on sensor data and the request type. Sensor data is managed within the edge network, and the OO processes these requests.

The types of sensors employed in the simulation are detailed in Table 4.3. Sensor data is categorized into two scenarios: faulty and working. A sensor is deemed faulty if its readings deviate from the predefined operational range; otherwise, it is considered functional. Figure 4.4 illustrates the different fault types in sensor data transmitted to the ITS within the edge network at 4 gNBs. gNB1 has 40.% GPS requests, 7.7% faulty wheel speed sensor, 21% faulty ultrasonic parking sensor. Similarly, we can see faults and requests in other gNBs. Faulty sensors are addressed at the edge by sending notifications about the faults, and GPS data is transmitted upon request from the

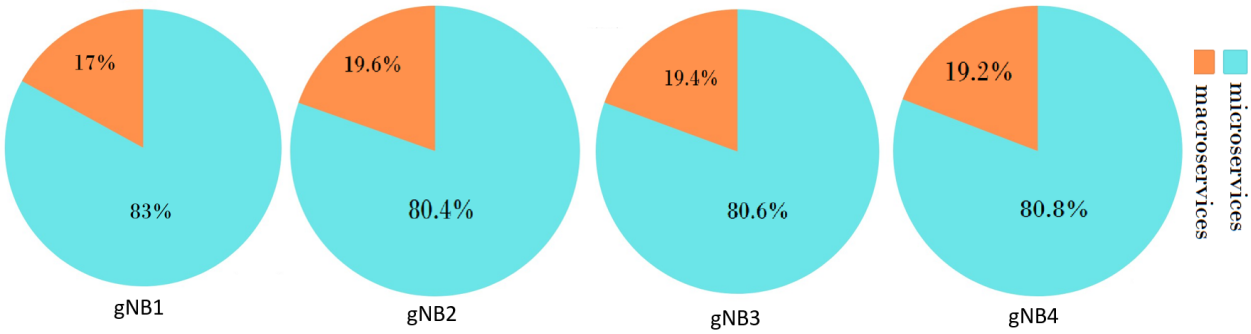
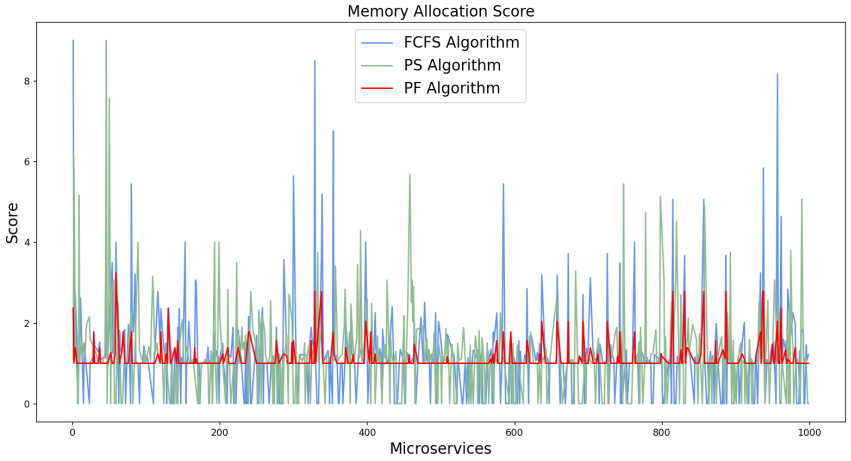


Figure 4.6: Number of *microservices* and *macroservices* detected at the edge network

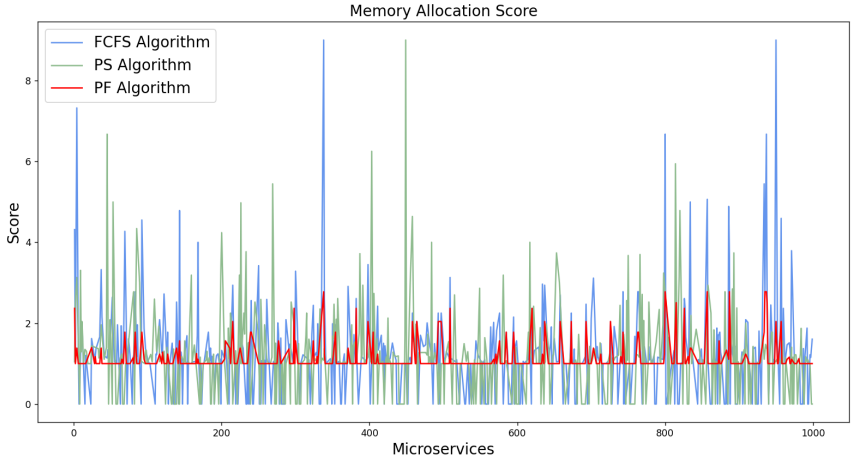
GPS sensor. Sensor data is classified as *microservice* if the requested data is available on the edge; if not, it is classified as *macroservice*.

4.2.1 Performance of the Proposed Algorithms

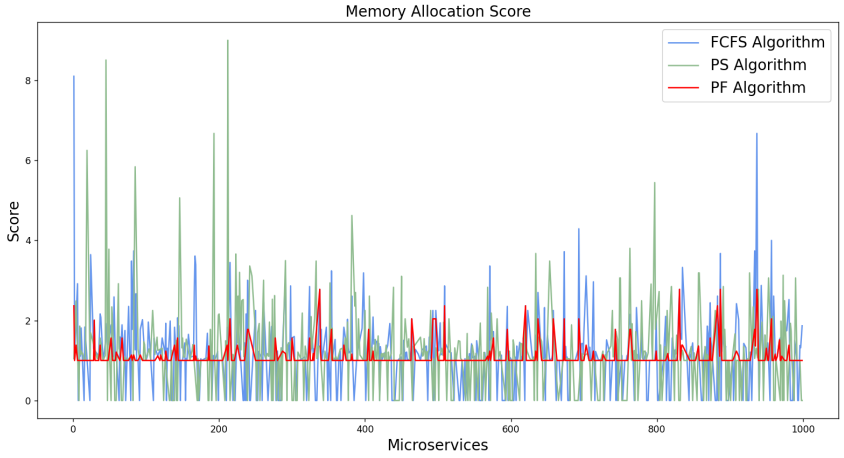
- *Service Segregation*: The service segregation algorithm is crucial in the edge network, categorizing the services into *macroservices* and *microservices*. This categorization is guided by threshold values provided by the ORD Algorithm 4.2. Figure 4.6 illustrates the distribution of segregated services across four gNBs. The *macroservices* detected at these gNBs are 17%, 19.6%, 19.4%, and 19.2%, respectively. This implies that approximately 19% of the identified services are *macroservices*. By segregating the services, the necessity of locating resources for these *macroservices* is eliminated, allowing for their direct migration to the cloud network. This approach optimizes resource allocation and enhances the overall system efficiency.



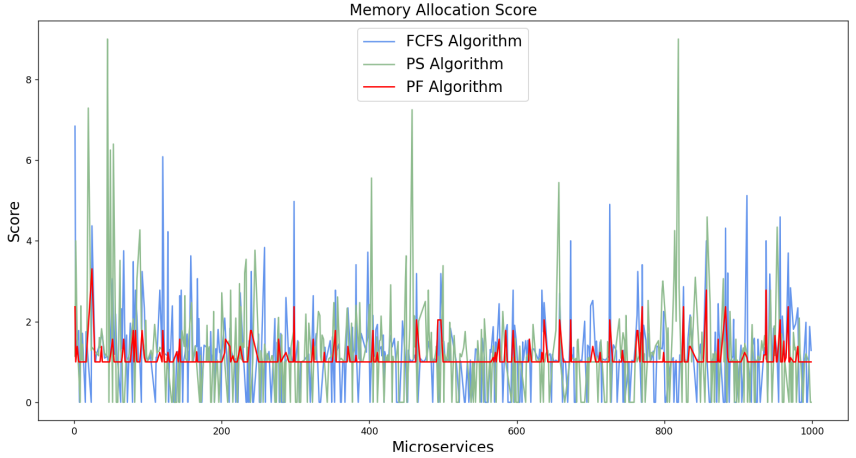
(a) gNB1



(b) gNB2

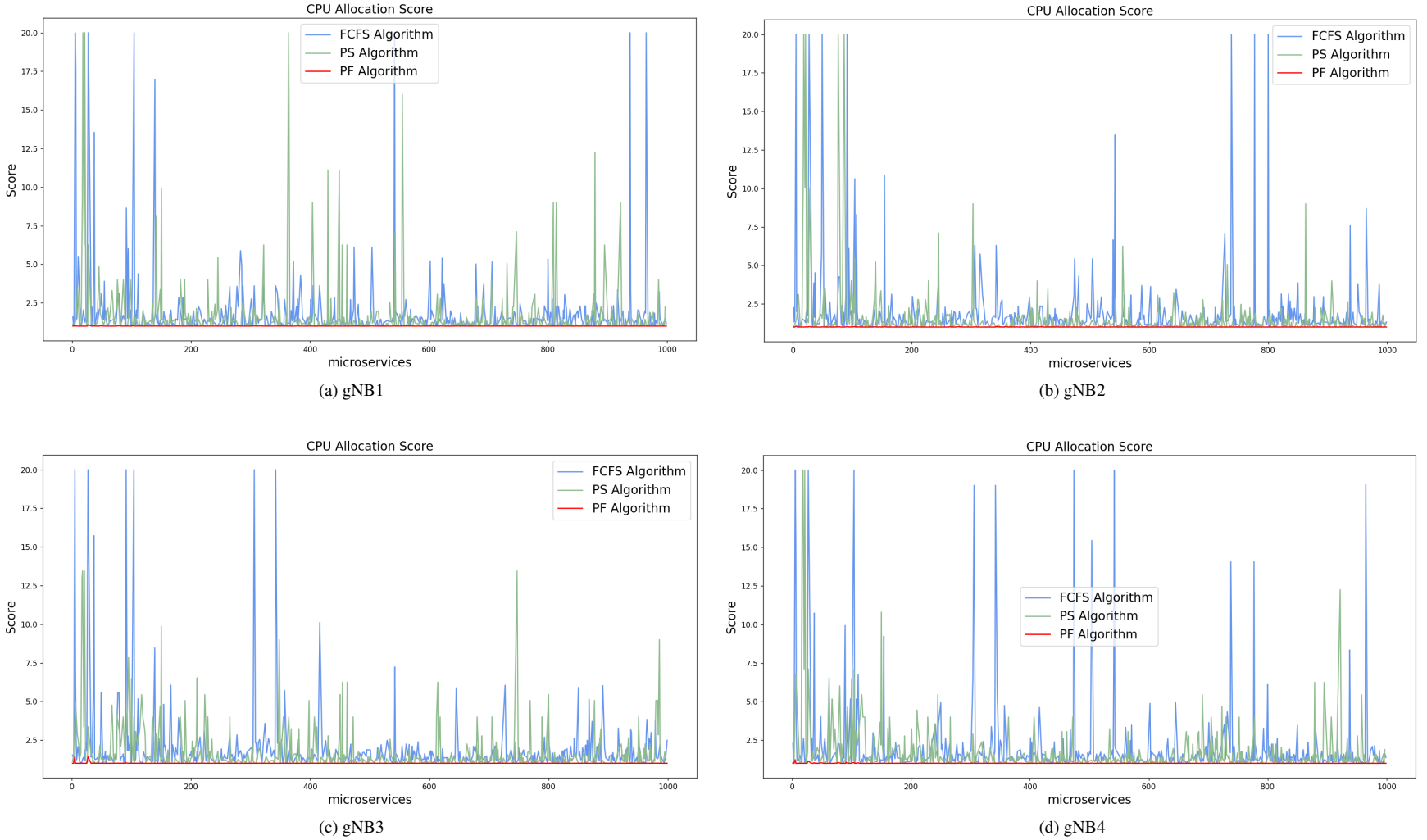


(c) gNB3



(d) gNB4

Figure 4.7: Memory Resource allocation using PF, FCFS, and PF Algorithm



(a) gNB1

(b) gNB2

(c) gNB3

(d) gNB4

Figure 4.8: Calculation Resource allocation using PF, FCFS, and PF Algorithm

- *Resource Allocation*: The PF algorithm, described in Algorithm 4.3, is utilized to allocate resources to intelligent vehicles. This approach evaluates the resource allocation score by comparing the available resources on the edge network to the requested resources. The score is computed as the ratio of the available resource amount to the required amount. A lower score indicates a closer match between the allocated resource and the requested amount, thus minimizing internal fragmentation. Conversely, a higher score suggests greater internal fragmentation. The PF method allocates two types of resources: calculation resources and memory resources. Figure 4.7 illustrates the memory resources allocated to intelligent vehicles. In this, Figures 4.7a-4.7d show the allocation in 4 gNBs. In this, the score for PF algorithm is lower than other algorithms. The PF algorithm clearly outperforms both FCFS and PS in terms of memory allocation stability and fairness. This advantage is critical for real-time applications, as it ensures predictable performance and better resource utilization. Figure 4.8 displays the calculation resources allocated using PF, First-Come-First-Serve (FCFS), and Priority Scheduling (PS) algorithms, Figure 4.8a-4.8d shows allocation in gNBs 1 to 4. The PF algorithm selects the resource with the minimum score, FCFS allocates the first available resource that meets or exceeds the requirement, and PS applies a fixed priority value based on the needs. The figures demonstrate that the PF algorithm achieves minimal internal fragmentation compared to the FCFS and PS methods. Table 4.4 presents the mean scores for calculation and memory resource allocations using PF, FCFS, and PS algorithms. The PF algorithm consistently shows the lowest mean score, indicating its superior performance in reducing fragmentation compared to FCFS and PS techniques. Table 4.5 illustrates the relative performance improvement of the proposed technique in comparison to the FCFS and PS techniques and from the table, the improvement is deduced to be up to 67% w.r.t. FCFS, and up to 74% w.r.t. PS algorithms.

Illustrative Example

Consider a simplified example involving a 5G edge computing network. At time t_1, the network comprises seven intelligent vehicles and 8 IoT devices connected to the base station gNB_1. At a subsequent time t_2, the network includes eight intelligent vehicles and 8 IoT devices connected to a different base station gNB_2.

Table 4.4: Mean score for CR and MR allocation using PF, FCFS, and PS Algorithm

gNB	Resource Type	PF	FCFS	PS
1	CR	1.000128	1.486763	1.743242
1	MR	1.136000	1.202540	1.194269
2	CR	1.007952	1.684099	1.173754
2	MR	1.131351	1.134696	1.199530
3	CR	1.000701	1.680290	1.314868
3	MR	1.092297	1.174868	1.194528
4	CR	1.000192	1.571955	1.295260
5	MR	1.149787	1.275606	1.201317

Table 4.5: Relative improvement (%) using PF w.r.t. to FCFS and PS algorithms

gNB	Resource Type	w.r.t. FCFS	w.r.t. PS
1	CR	48.6572	74.3021
1	MR	5.857	5.1293
2	CR	67.0812	16.449
2	MR	0.295	0.06026
3	CR	67.9112	31.39
3	MR	7.5593	9.35
4	CR	57.165	29.50
4	MR	10.9428	0.44816

In this scenario, IoT devices contribute resources to the edge network while intelligent vehicles submit requests for these resources. The edge network utilizes Osmotic Computing to allocate resources efficiently to intelligent vehicles. The focus is on memory resource allocation requested by the intelligent vehicles. Table 4.6 and Table 4.7 illustrate the service requests from IoT devices to the edge network at times t_1 and t_2, respectively. Table 4.8 presents the resources lent by the IoT devices to the edge network, with the threshold value derived from this table. The service type is determined by the service segregation algorithm. The columns labeled FCFS, PS, and PF denote the resources allocated to intelligent vehicles by IoT devices according to the First Come First Serve, Priority Scheduling, and Proportional Fairness algorithms, respectively. The columns FCFS IF, PS IF, and PF IF represent the internal fragmentation resulting from these algorithms. Analysis of the internal fragmentation values indicates that the PF algorithm exhibits the lowest internal fragmentation and is capable of accommodating more service requests compared to the other algorithms.

Table 4.6: Representation of Resource allocation using FCFS, PS, and PF algorithms in Osmotic Computing based edge network using toy example at time t_1

Vehicle_id	Memory Required	Priority	Threshold	Service Type	FCFS	FCFS IF	PS	PS IF	PF	PF IF
IV1	8	2	9	*microservice*	IoT2	0	IoT2	0	IoT2	0
IV2	5	3	9	*microservice*	IoT1	1	IoT3	4	IoT8	0
IV3	12	-	9	*macroservice*	-	-	-	-	-	-
IV4	4	1	9	*microservice*	IoT3	5	IoT1	2	IoT7	0
IV5	8	5	9	*microservice*	-	-	-	-	IoT3	1
IV6	7	4	9	*microservice*	IoT4	0	IoT4	0	IoT5	0
IV7	6	6	9	*microservice*	IoT5	1	IoT5	1	IoT1	0

Table 4.7: Representation of Resource allocation using FCFS, PS, and PF algorithms in Osmotic Computing based edge network using toy example at time t_2

Vehicle_id	Memory Required	Priority	Threshold	Service Type	FCFS	FCFS IF	PS	PS IF	PF	PF IF
IV1	7	4	8	*microservice*	IoT1	1	IoT5	1	IoT3	0
IV2	3	6	8	*microservice*	IoT2	1	IoT4	3	IoT6	0
IV3	4	2	8	*microservice*	IoT3	3	IoT2	0	IoT2	0
IV4	8	5	8	*microservice*	IoT5	0	-	-	IoT1	0
IV5	10	-	8	*macroservice*	-	-	-	-	-	-
IV6	5	7	8	*microservice*	IoT4	1	IoT7	2	IoT8	0
IV7	7	1	8	*microservice*	IoT7	0	IoT1	1	IoT7	0
IV8	6	3	8	*microservice*	-	-	IoT3	1	IoT4	0

Table 4.8: Representation of ORD in Osmotic Computing based edge network using toy example

IoT_id	**Resource Lent at time t_1**	**Resource Lent at time t_2**
IoT1	6	8
IoT2	8	4
IoT3	9	7
IoT4	7	6
IoT5	7	8
IoT6	3	3
IoT7	4	7
IoT8	5	5
Threshold	9	8

Table 4.9: ANOVA table for memory resource obtained from PS, FCFS, and PF algorithms

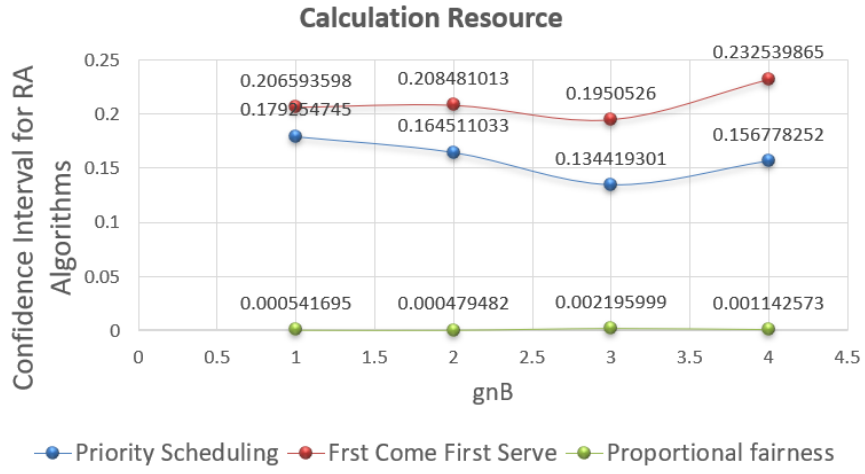
Source of variation	SS	df	MS	F	$F_{critical}$
Between Groups	151.363148	2	75.681574	106.05981	3.003094
Within Groups	871.274443	1221	0.713574	-	-
Total	921.728825	1223	-	-	-

4.2.2 Result Validation

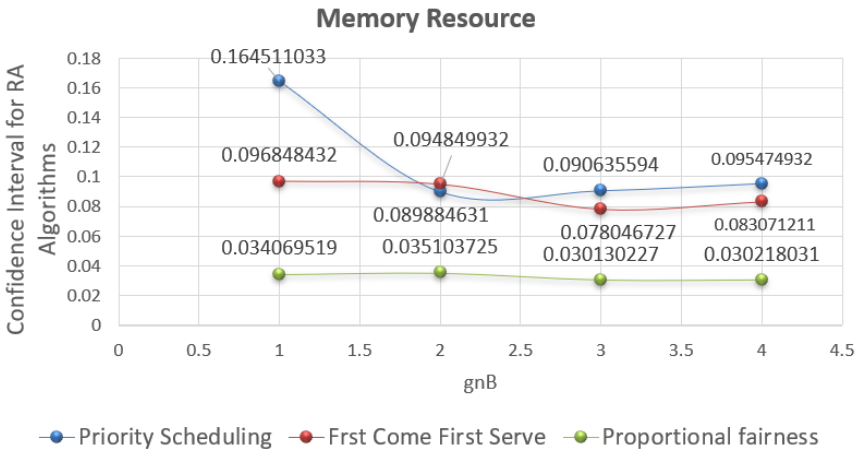
To validate the proposed technique, a one-way Analysis of Variance (ANOVA) test was conducted to determine if there is a statistically significant difference between the means of the PS, FCFS, and PF algorithms. The null hypothesis (H_0) proves that the mean performance metrics for memory and calculation resources are identical across all algorithms, i.e., $H_0 : \mu_{FCFS} = \mu_{PS} = \mu_{PF}$, where μ_{FCFS} represents the mean performance value for the First Come First Serve algorithm, μ_{PS} denotes the mean performance value for the Priority Scheduling algorithm, and μ_{PF} signifies the mean performance value for the Proportional Fairness algorithm. The alternative hypothesis (H_a) asserts that not all means are equal. The test was applied to both resources: memory and calculation. An alpha level of 0.05 was used for the analysis.

Table 4.10: ANOVA table for computation resource obtained from PS, FCFS, and PF algorithms

Source of variation	SS	df	MS	F	$F_{critical}$
Between Groups	765.314442	2	382.657221	112.276337	3.001448
Within Groups	5357.648505	1572	3.408173	-	-
Total	5612.753319	1574	-	-	-



(a)



(b)

Figure 4.9: Confidence Interval for PS, FCFS, and PF algorithms in terms of calculation and memory resource allocation

The results of the ANOVA test are detailed in Tables 4.9 and 4.10. The mean square (MS), degrees of freedom (df), and the sum of squares (SS) were calculated to derive the F statistic, which was then compared with the critical value $F_{critical}$. As shown in the tables, the computed F value exceeds $F_{critical}$, leading to the rejection of the null hypothesis in favor of the alternative hypothesis. This indicates that the means of the three algorithms are not equal. Additionally, the 95% confidence intervals for each algorithm are depicted in Figures 4.9a and 4.9b. The figures reveal that the margin of error for the Proportional Fairness algorithm is notably smaller compared to the other two algorithms for both memory and calculation resource allocation.

4.3 Conclusion

This chapter introduces OsCoMIT, a novel framework based on Osmotic Computing designed to optimize task segregation and resource allocation in Intelligent Transportation Systems (ITS) through the application of the Proportional Fairness Algorithm. The framework facilitates the handling of service requests from intelligent vehicles in the edge network, managed by the OO. The service requests are categorized into *microservices* and *macroservices*. *Microservices* are managed by OO with allocated resources, while *macroservices* are migrated to the cloud network due to the limitations in resource availability at the edge. Resources borrowed from IoT devices are allocated to *microservices* through the resource allocation algorithm. Evaluations are conducted using four gNBs, considering various amounts of lent and requested resources. Table 4.5 shows the relative improvement of the proposed technique as compared to FCFS and PS technique. The proposed framework provides several advantages. Firstly, it employs the concept of Osmotic Computing (OC) to enhance resource allocation by sharing resources from IoT devices within the edge network. This approach increases resource utilization, enhances network efficiency, and reduces resource wastage. Secondly, the framework differentiates between *microservices* and *macroservices*, which allows for optimized task scheduling and resource allocation based on specific requirements, thus minimizing resource wastage. Thirdly, it ensures that resource allocation is executed in a fair and balanced manner, addressing the needs of various *microservices* in the edge network. The results demonstrate the effectiveness of the proposed framework in enhancing task segregation and resource allocation in ITS through Osmotic Computing. The migration of *macroservices* to the cloud network and the use of the Proportional Fairness algorithm for allocating resources to *microservices* at the edge contribute to improved resource utilization and overall network performance.

Future research will extend the findings presented in this chapter. Although the focus has been on service segregation and resource allocation, latency considerations have not been addressed. In the subsequent work strategies to reduce latency within the OC framework will be explored. Additionally, mobility conditions, such as dynamic user movement and changing network topologies will also be considered.

Chapter 5

Optimizing Service Migration and Task Offloading in Osmotic Computing with Deep Q-Networks[1]

This chapter presents a novel method for decision-making using DQN-osmosis. It uses Deep Q-Networks (DQN) in OC environment to intelligently manage and allocate resources. The proposed method has two different operations: service migration and task offloading. Using reinforcement learning, it learns to make optimal decisions based on current state metrics, such as resource usage. It adapts to varying network situations and ensures effective resource allocation and enhanced performance. The mathematical model, that forms the basis of the proposed DQN-based approach is also presented. The results of the model demonstrate enhancements in the speed of task processing and enhance the overall efficiency of the system, proving the effectiveness of DQN in the Osmotic Computing paradigm. To validate the hypothesis that the DQN-based framework performs better than the other three methods, i.e., Random Agent, Q-Learning, and SARSA algorithm, the Wilcoxon signed-rank test is performed.

[1]The contents of this chapter are accepted: Kaur, A., Kumar, R., & Saxena, S., "Optimizing Service Migration and Task Offloading in Osmotic Computing with Deep Q-Networks." 16th International Conference on Ubiquitous Computing and Ambient Intelligence (UCamI 2024) - Ulster University - Belfast (Northern Ireland, United Kingdom)

5.1 Introduction

The traditional cloud computing provides substantial computational power, but it suffers from latency due to the physical distance between the users and cloud data centers. To address these latency issues, edge computing has been introduced. But edge computing has restricted computational and storage power and has scalability issues. Osmotic Computing (OC), addresses these challenges by enabling transparent deployment and migration of distributed services. These distributed services are called MicroELements (MELs) in OC [2, 103]. It ensures reduced response time and optimal resource utilization. The main aspect of achieving effective osmotic computing lies in intelligent decision-making processes that dynamically adapt to the changing workload and environment.

5.2 Elements of Reinforcement Learning

Reinforcement Learning (RL) is founded on many fundamental ideas that delineate an agent's decision-making and learning processes through interaction with the environment. The elements of Reinforcement Learning (RL) are as follows.

- Agent: It is the decision-maker or learner. To achieve the goal it takes the actions in the environment. It operates on value functions and policies.
- Environment: The agent engages with the environment. It provides the agent with observations and rewards. The environment state transition policy can be stated as shown in Equation 5.1:

$$P(s'|s,a) = \Pr(S_{t+1} = s'|S_t = s, A_t = a) \tag{5.1}$$

  where s and s' are states, and a is the action taken by the agent.

- State (S): It is the representation of the environments current situation. The agent takes action after observing the state. The future state depends on the current state and can be represented as shown in Equation 5.2:

$$s_{t+1} = f(s_t, a_t) \tag{5.2}$$

where f is the environment's state transition function.

- Action (A): The action taken by the agent in response to the current state. It can be represented as shown in Equation 5.3:

$$a_t \sim \pi(s_t) \tag{5.3}$$

- Reward (R): It is the feedback signal that tells the agent about how well it is doing at a specific time. The goal is to maximize the cumulative reward over time. It can be represented as shown in Equation 5.4:

$$r_t = R(s_t, a_t) \tag{5.4}$$

where r_t is the reward received after taking action a_t in state s_t.

- Policy: It is defined as the agents behavior by specifying the actions in each state. It can be defined as Equation 5.5.

$$\pi(a|s) = \mathbf{Pr}(A_t = a|S_t = s) \tag{5.5}$$

where $\pi(a|s)$ is the probability of taking action a in state s.

- Value Function: It is the cumulative reward an agent can achieve from the given state as shown in Equation 5.6.

$$V^{\pi}(s) = \mathbb{E}_{\pi}\left[\sum_{t=0}^{\infty} \gamma^t r_t \middle| S_0 = s\right] \tag{5.6}$$

where γ is the discount factor that determines the importance of future rewards.

- State transition model: It predicts the next state and reward given a state and action. It can be represented as given in Equation 5.7:

$$P(s', r|s, a) = \mathbf{Pr}(S_{t+1} = s', R_t = r|S_t = s, A_t = a) \tag{5.7}$$

where P is the probability distribution over the next state s' and reward r, given the current state s and action a.

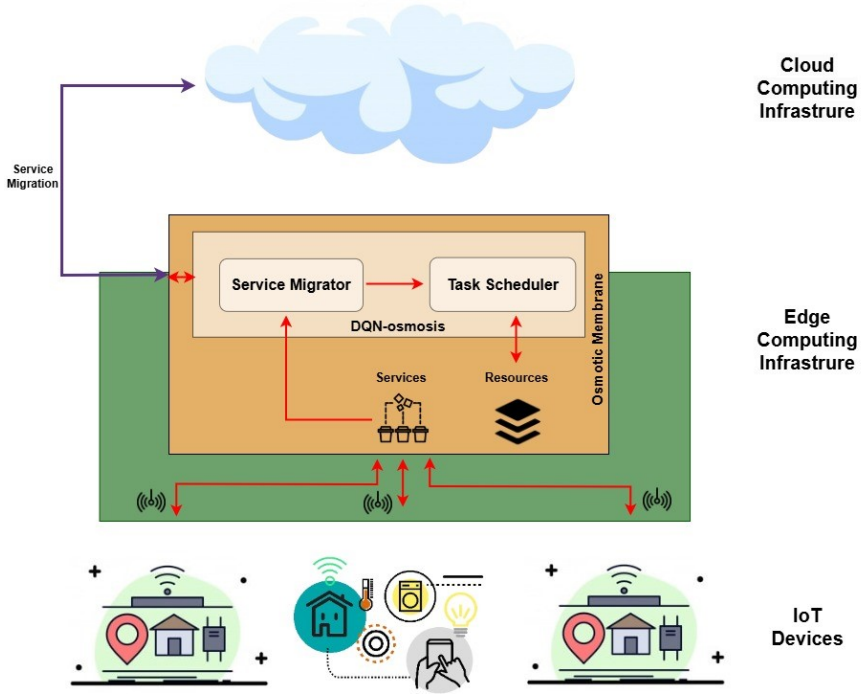


Figure 5.1: Proposed DQN-osmosis Architecture

- Discount Factor: This parameter values between zero and one. It determines the value of future rewards as compared to immediate rewards. It can be represented as Equation 5.8

$$G_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots \tag{5.8}$$

5.3 DQN-osmosis Framework

The DQN-osmosis framework is shown in Figure 5.1. The framework is divided into three layers: IoT device layer, Edge computing layer, and Cloud computing layer

- *IoT device layer*: It is the sensor data acquisition layer. The process entails gathering data from multiple IoT devices dispersed throughout the network. IoT devices create demands for computing activities, necessitating resource allocation at either the edge or in the cloud.
- *Edge computing layer*: This layer pertains to resource allocation and management. It is tasked with assessing the available resources across multiple edge devices. The major objective of this layer is to ascertain if an IoT device's computing work can be executed locally at the edge or must be delegated to the cloud. It amalgamates Osmotic Computing with the

DQN method. It assesses the available RAM, storage, and CPU capacity of edge devices prior to determining their capability to manage the computing demands of an IoT device. When edge devices possess adequate resources, tasks are assigned to them, thereby optimizing local processing efficiency. The resource allocation layer is essential for distributing the load between edge devices and the cloud. It guarantees that tasks are executed in proximity to the data source, hence minimizing latency and enhancing system performance. This layer additionally enhances energy efficiency by utilizing local resources.

- *Decision-Making and Learning Layer*: This is the fundamental layer accountable for intelligent decision-making within the framework. The DQN method estimates the Q-value function to determine optimal task distribution between edge devices and the cloud. The DQN algorithm acquires knowledge from past experiences and dynamically updates its decisions in response to the evolving conditions of IoT devices and accessible resources. The agent's action space comprises choosing one of the edge devices or migrating the task to the cloud. The algorithm obtains feedback through rewards (1 for successful edge allocations, 0 for cloud migrations), which it utilizes to refine its policy and enhance subsequent judgments. This layer facilitates adaptive resource management. It perpetually acquires knowledge from its surroundings, enhancing the efficacy of resource allocation decisions progressively. The DQN's capacity to generalize across diverse states enables the framework to scale and adjust to varying network conditions.

- *Reward Feedback Layer*: This layer provides feedback to the DQN algorithm in the form of rewards, which help the agent learn optimal policies for resource allocation. The rewards are calculated based on the success or failure of task allocations. A positive reward of 1 is given when a task is successfully processed at an edge device, while a zero reward is assigned for cloud migrations. The reward system incentivizes the DQN agent to prioritize edge allocations, which are generally more efficient in terms of both energy consumption and processing time. The reward feedback layer plays a critical role in reinforcing successful behaviors in the system. By shaping the rewards based on desired outcomes (such as minimizing cloud migrations), the DQN agent is trained to make decisions that improve overall system performance.

- *Cloud computing layer*: When edge devices lack sufficient resources to handle a task, this layer is responsible for migrating the task to cloud servers. The cloud layer provides a fallback mechanism to ensure that IoT tasks are processed, even when edge resources are exhausted. The DQN-osmosis framework interacts with this layer when edge devices fail to meet the computational requirements of IoT tasks. This layer monitors cloud availability and ensures that tasks are migrated and processed efficiently. Cloud migration incurs higher latency and energy costs, so it is only used when necessary, and the framework aims to minimize migrations. The cloud migration layer serves as a backup to maintain continuous service when edge resources are limited. It allows the system to handle overflow conditions and manage peak loads but at the cost of increased energy usage and delay. This layer contributes to overall system resilience.

5.3.1 System Model

The proposed DQN-Osmosis framework is designed to optimize resource management within the IoT-Edge-Cloud continuum, focusing on improving service migration and task offloading processes. By utilizing Deep Q-Networks (DQN), the framework dynamically adjusts to changing network workloads, ensuring efficient resource allocation and enhanced service delivery. The DQN-osmosis operates through a structured process, beginning with the initialization of its essential components. It starts by initializing the state space, action space, and replay memory. The state space represents the set of all possible states in which the system can be present. A state s is a vector in $\mathbb{R}^n$, where n is the dimensionality of the state space. The state includes information about CPU usage, RAM usage, and storage. Action space is a set that includes the agent's possible actions. An action a is a discrete value in the set $\{0, 1, \ldots, K-1\}$, where K is the total number of possible actions. Actions include migrating tasks to the cloud and offloading services to edge devices. The replay memory $\mathcal{D}$ is a buffer used to store the agent's experiences during training. An experience e_t consists of the state s_t, the action a_t, the reward r_t, the next state s_{t+1}, and a flag indicating if the episode is done. The memory has a maximum capacity N, beyond which old experiences are discarded.

Q-Value Function Approximation estimates the value of state-action pairs in environments where the state or action space is too large to handle explicitly. Neural networks are used to

approximate the Q-value function, which estimates the expected cumulative reward of taking a particular action in a given state. The Q-value function $Q(s, a; \theta)$ is approximated by a neural network parameterized by θ. The goal is to learn the optimal Q-values $Q^*(s, a)$, which represent the maximum expected reward achievable from state s by taking action a and the optimal policy can be represented as shown in Equation 5.9:

$$Q(s, a; \theta) \approx Q^*(s, a) \tag{5.9}$$

A distinct target Q-network with parameters θ^- is employed to stabilize training. This network serves as a periodic copy of the Q-network, offering steady target values throughout the training process. The remember function stores experiences in the replay memory as shown in Equation 5.11:

$$F = \{(s_t, a_t, r_t, s_{t+1})\} \tag{5.10}$$

$$\mathcal{D} \leftarrow \mathcal{D} \cup F \tag{5.11}$$

This step ensures that the agent has diverse experiences to learn from, which is crucial for effective training. Then, the agent selects actions using an ϵ-greedy policy, balancing exploration and exploitation as given in Equation 5.12:

$$a_t = \begin{cases} \text{random action} & \text{with probability } \epsilon \\ \arg\max_a Q(s_t, a; \theta) & \text{with probability } 1 - \epsilon \end{cases} \tag{5.12}$$

With probability ϵ, a random action is selected to explore the action space. With probability $1 - \epsilon$, the action that maximizes the Q-value function $Q(s_t, a; \theta)$ is chosen. This strategy helps the agent discover new strategies while still leveraging its learned knowledge.

The replay function updates the Q-values based on sampled experiences. The target value y_i is the actual reward plus the discounted future reward from the target Q-network. If the episode has ended, the target is simply the immediate reward as shown in Equation 5.13:.

$$y_i = \begin{cases} r_i & \text{if done} \\ r_i + \gamma \max_{a'} Q(s_{i+1}, a'; \theta^-) & \text{otherwise} \end{cases} \tag{5.13}$$

Here, γ is the discount factor that determines the importance of future rewards. The network weights θ are updated to minimize the difference between the predicted Q-values and the target values. This is done using gradient descent as shown in Equation 5.14:.

$$\theta \leftarrow \theta - \alpha \nabla_\theta \left(y_i - Q(s_i, a_i; \theta)\right)^2 \tag{5.14}$$

Here, α is the learning rate, and ∇_θ represents the gradient with respect to θ.

The exploration rate ϵ is decayed to reduce exploration over time and exploits learned policies more, ϵ is decayed as per the Equation 5.15:

$$\epsilon \leftarrow \max(\epsilon_{\text{min}}, \epsilon \cdot \epsilon_{\text{decay}}) \tag{5.15}$$

This ensures that the agent gradually shifts from exploration to exploitation. The loss function used to train the network is the Mean Squared Error (MSE) between the target y_t and the predicted Q-value $Q(s_t, a_t; \theta)$ as shown in Equation 5.16:

$$L(\theta) = \mathbb{E}_{(s,a,r,s',\text{done})\sim\mathcal{D}} \left[\left(y_t - Q(s_t, a_t; \theta)\right)^2\right] \tag{5.16}$$

This loss guides the training process to improve the Q-value approximations. The osmotic Decision-Making function integrates the decision-making process within the system. It reshapes the state of the neural network input to match the input dimensions expected by the neural network as shown in Equation 5.17:.

$$\text{state} \leftarrow \text{reshape}(state, [1, s]) \tag{5.17}$$

The ϵ-greedy policy, as per Equation 5.18, is used to select an action based on the current state.

$$\text{action} \leftarrow \text{action(state)} \tag{5.18}$$

Different actions, as per Equation 5.19, result in different system responses, such as migrating tasks to the cloud or offloading services to an edge device.

$$\begin{cases} \text{Migrate tasks to cloud} & \text{if a} = 0 \\ \text{Offload services to edge device} & \text{if a} = 1 \end{cases} \tag{5.19}$$

Algorithm 5.1 DQN-Based Decision Making for Osmotic Computing

Initialize replay memory $\mathcal{D}$ with capacity N
Initialize target Q-network with weights $\theta^- = \theta$
Initialize state s
for each episode **do**
    for each step in episode **do**
        Reshape the state for the neural network input: state $\leftarrow$ reshape$(state, [1, s])$
        Select an action using the act function: action $\leftarrow$ act(state)
        Execute actions based on the selected action:

$$\begin{cases} \text{Migrate tasks to cloud} & \text{if a} = 0 \\ \text{Offload services to edge device} & \text{if a} = 1 \end{cases}$$

        Execute action a and observe reward r and next state s'
        Store transition $(s, a, r, s', \text{done})$ in $\mathcal{D}$
        Set $s = s'$
        Sample random mini-batch of transitions $(s_j, a_j, r_j, s_{j+1}, \text{done})$ from $\mathcal{D}$
        for each transition in mini-batch **do**
            Compute target y_j
            if done **then**
                $y_j = r_j$
            else
                $y_j = r_j + \gamma \max_{a'} Q(s_{j+1}, a'; \theta^-)$
            end if
            Perform gradient descent step on $(y_j - Q(s_j, a_j; \theta))^2$
        end for
        Update $\epsilon = \max(\epsilon_{\text{min}}, \epsilon \cdot \epsilon_{\text{decay}})$
    end for
    Periodically update target Q-network $\theta^- = \theta$
end for

5.4 Results and Discussions

This section investigates the performance of the proposed DQN-based decision-making framework for osmotic computing by comparing it with other algorithms. A series of simulation experiments are performed using Python 3.11.1, *collections.deque* for relay memory buffer and TensorFlow for training neural networks on a server equipped with Intel code i7-12700 CPU @ 2.10 GHz, 16.0 Gb RAM.

To ensure comprehensive evaluation, the following parameters and configurations were used in the simulation environment:

- *Replay Memory Capacity:* 10,000 transitions
- *Q-Network*: Initialized with random weights and updated periodically
- *Target Q-Network*: Updated with the Q-network weights periodically
- *State Representation*: Includes system states such as resource availability, task queue lengths, and network latency
- *Action Space*: Consists of task migration to the cloud, service offload to edge devices actions
- Reward Function: Designed to minimize latency and optimize resource utilization
- *Training Episodes*: Actions are taken based on the current policy with 100 episodes, each consisting of multiple steps.

5.4.1 Results and Analysis

Algorithm 5.1 shows the proposed algorithm. To verify the proposed technique, the DQN-Osmosis method is compared with Q-Learning, Random Agent, and SARSA algorithms. The results are validated for 100 IoT devices over 100 episodes. The comparison is made based on the rewards accumulated by all the algorithms for the same data. There are two actions taken by the algorithm, i.e., migration of the IoT request to the cloud or allocation of resources at the edge layer. The rewards calculated for each decision are either zero or one. One if the resource is allocated at the edge and zero if the request is migrated to the cloud. Table 5.1 shows the comparison of all the algorithms for both rewards. Compared to other approaches, the proposed algorithm performs better.

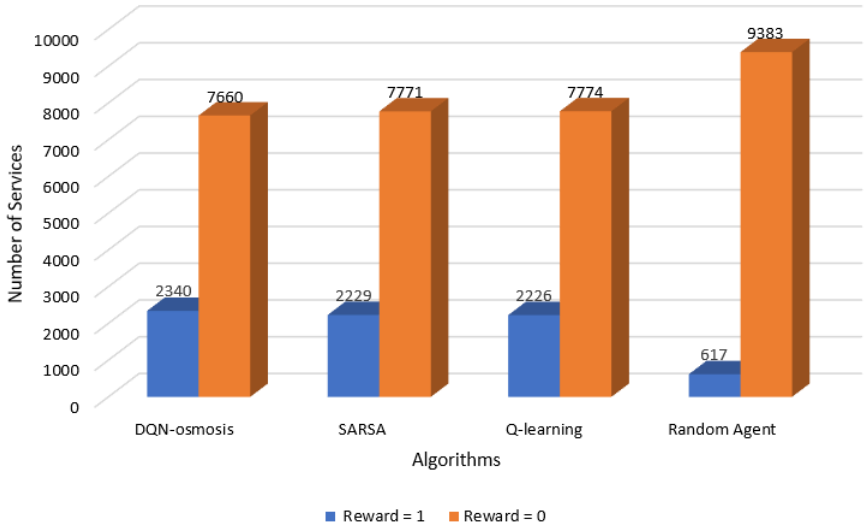


Figure 5.2: Comparison of DQN-osmosis, Q-Learning, SARSA, and Random Agent Algorithm

Figure 5.2 shows the comparison of DQN with other algorithms based on rewards accumulated over 100 episodes. It can be inferred from the figure that the proposed method consistently achieves higher cumulative rewards than the other methods for resource allocation at the edge and the lowest for cloud migrations, with DQN-Osmosis having 2340 allocations at the edge layer as compared to SARSA, Q-Learning, and Random Agent which have 2229, 2226, and 617 allocations respectively.

Table 5.1: Comparison of Rewards for Resource Allocation Algorithms

Algorithm	**Reward = 1**	**Reward = 0**
DQN-osmosis	2340	7660
SARSA	2229	7771
Q-learning	2226	7774
Random Agent	617	9383

5.4.2 Statistical Analysis based on Wilcoxon Signed-Rank Test

This section utilizes the Wilcoxon signed-rank test [104] to compare the cumulative rewards achieved by DQN-Osmosis and other algorithms. The Wilcoxon signed-rank test is a non-parametric statistical method employed to evaluate disparities between paired data. The objective is to ascertain whether there exists a statistically significant difference in performance between the two strategies. The hypothesis for the Wilcoxon signed-rank test is as follows:

- Null Hypothesis (H_0): There is no significant difference in the cumulative rewards between the DQN-Osmosis and other algorithms.
- Alternative Hypothesis (H_1): There is a significant difference in the cumulative rewards between the DQN-Osmosis and other algorithms.

Calculation of Results: The Wilcoxon signed-rank test was performed using the cumulative rewards collected for the DQN-Osmosis and other methods. The p-value is calculated which determines whether sufficient evidence exists to reject the null hypothesis. It is a statistical measure (less than 0.05) that is used to determine the strength of evidence against the null hypothesis.

Table 5.2: Wilcoxon Test Result

Comparison with DQN-Osmosis		
Sr no.	**Algorithm**	**P-Value**
1	Random	$<$ 1e-15
2	Q-Learning	8.715576e-15
3	SARSA	1.209506e-14

With a significance level of $\alpha = 0.05$, the p-value is considerably lower than α, prompting the rejection of the null hypothesis. This signifies a statistically significant disparity in performance between DQN-Osmosis and other algorithms.

5.5 Conclusion

This chapter presents a novel approach for intelligent decision-making in osmotic computing using Deep Q-Networks (DQN). The proposed method addresses the challenges of service migration and task offloading in dynamic and heterogeneous environments, aiming to enhance response times and optimize resource utilization. A comprehensive mathematical model that integrates reinforcement learning with the principles of osmotic computing is developed. This model effectively captures the complexities of managing resources and deploying services across both cloud and edge environments. The implementation and extensive simulations demonstrated that the DQN-based approach significantly outperforms other algorithms, as can be seen by the p-values in all the

comparisons. The results confirmed the statistical significance of the performance improvements, thereby reinforcing the robustness and reliability of the proposed approach. Future work will focus on extending the proposed approach to handle more complex and larger-scale environments. Additionally, we aim to explore other performance metrics like bandwidth.

Chapter 6

$\mu-osmotic$: A Microservice Management Framework for Intelligent Transportation System [1]

This chapter introduces the $\mu-osmotic$ framework, a novel approach designed to address the challenges of dynamic resource allocation in Intelligent Transportation System. As these devices generate vast amounts of data, the need for efficient computational paradigms at the network edge becomes critical. The proposed $\mu-osmotic$ framework leverages Osmotic Computing principles and the Advantage Actor-Critic (A2C) algorithm to optimize the distribution of service requests between edge and cloud resources. By dynamically managing resources based on real-time factors such as CPU usage, memory consumption, and energy efficiency, $\mu-osmotic$ enhances service performance, reduces latency, and ensures effective resource utilization. Through comprehensive evaluation and comparison with other algorithms, the framework demonstrates significant improvements, making it a valuable contribution to the field of resource management.

[1]The contents of this chapter are communicated in IEEE Transactions on Intelligent Transportation Systems: Kaur, A., Kumar, R., & Saxena, S. (2024). " $\mu-osmotic$: Service Management in IoT Microservice Architecture Leveraging Osmotic Computing and A2C"

6.1 Proposed $\mu-osmotic$ Framework

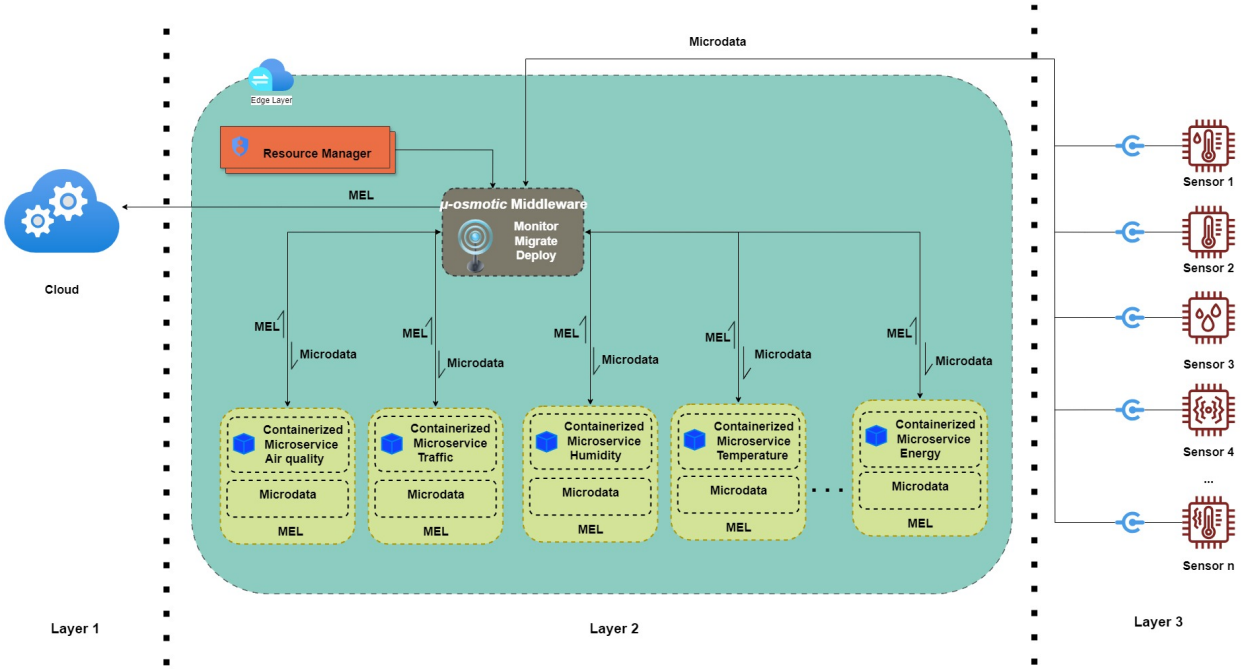


Figure 6.1: $\mu-osmotic$ Framework

The $\mu-osmotic$ framework consists of three layers: Cloud, Edge, and IoT, as shown in Figure 6.1. Containerized *microservices* for various sensor data are deployed at the edge layer. The resource manager at the edge layer manages the resources available at the edge layer. The middleware, deployed at the edge layer, controls the osmotic behavior of the framework. Figure 6.2 shows the middleware. The main elements of the middleware are monitoring, deploy, and migration. It dynamically adjusts the microservice placement based on the performance metrics and the resources available, leveraging a container engine for deployment and management. $\mu-osmotic$ monitors the node for incoming requests, monitors the edge resources, maintains a database for the gathered metrics of CPU, Memory, network traffic, and disk I/O, and has a migration decision engine.

6.1.1 Layer 1: Cloud Layer

Layer 1 is the cloud layer. The services received at the edge layer are migrated to the cloud layer along with the microservices when the resources at the edge are insufficient. At the edge layer,

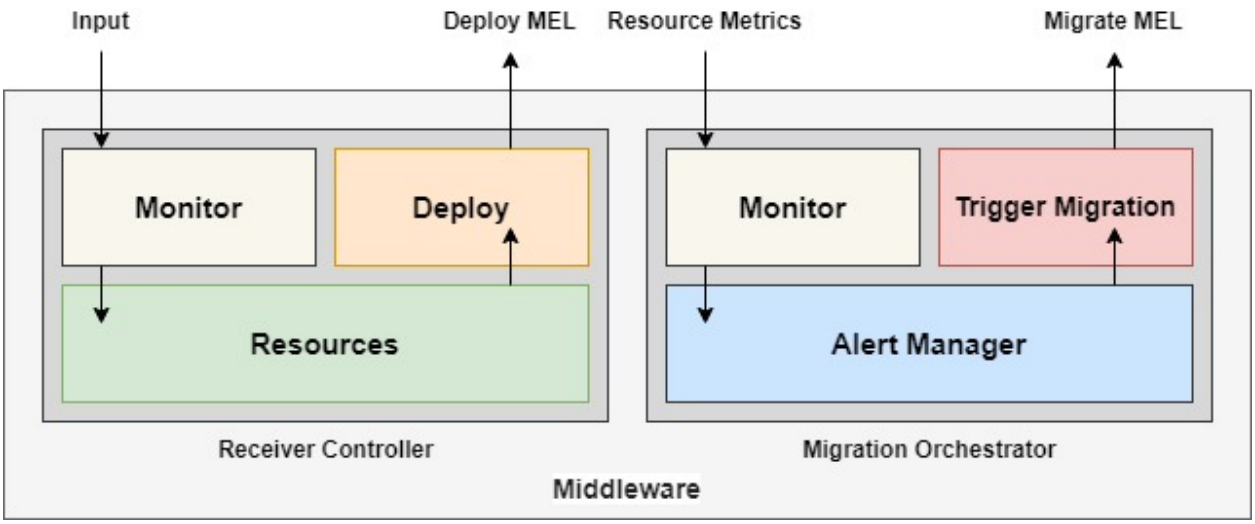


Figure 6.2: $\mu-osmotic$ middleware

the microdata received from the sensors has to be processed with the microservice. It is migrated to the cloud in the form of MicroELements (MELs) by encapsulating the microdata along with the microservice. This ensures that in the presence of any constraints at the edge, the functionality and the performance of the system is maintained by offloading the data to the cloud. The processed data is then sent back to the cloud for further actions allowing seamless service delivery and efficient resource utilization.

6.1.2 Layer 2: Edge Layer

Layer 2, the edge layer, represents the core of the system in terms of primary data processing. Most data processing tasks are performed here, in the vicinity of data sources, thus reducing latency and improving response times. It consists of several containerized *microservices* that process specific types of *microdata* received from the sensors in Layer 3. One of the main goals of the edge layer is local processing, which saves bandwidth and reduces energy consumption. In the absence of sufficient resources at the edge layer, the data is encapsulated and migrated to the cloud in the form of MELs. The middleware at the edge layer includes monitoring, resource allocation, and migration for optimal performance and resource utilization.

- *Resource Manager*: This component of the middleware monitors the use and the reservation of resources at the edge layer. It ensures that every microservice has enough storage and

computational resources at the edge layer. In the absence of enough resources, it coordinates with the cloud layer by encapsulating the microservice and microdata and migrating the resulting MEL to the cloud.

- *Middleware*: The middleware migrates, monitors, and deploys the microservices. It guarantees that *microservices* are running well as they have to be migrated to the cloud based on resource availability and performance requirements.
- *Containerized Microservices*: The *microservices* are containerized ensuring efficient and isolated execution. The microservices are designed to handle specific data sent by the sensors. The *microservices* used in the framework are air quality, humidity, traffic, temperature, and energy consumption.

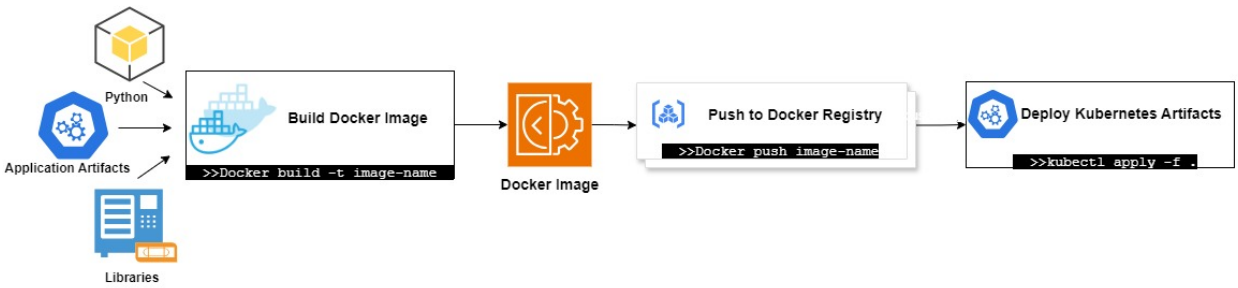


Figure 6.3: The process of implementing microservices in Containers

- *Microservices with Containers*: *Microservice* architecture is a design pattern that structures an application as a collection of loosely coupled services. *Containers* are an ideal platform for deploying *microservices* due to their isolation, portability, and ease of management. In $\mu-osmotic$, five *microservices* have been deployed for various sensor types. To implement the *microservices*, a *Dockerfile* for each microservice is created to specify the environment and dependencies required to run it, followed by building *Docker* images from these *Dockerfiles*. Then, the images can be loaded into the Google Container Registry, which enables recovery, scalability, and simple development. Then, *Kubernetes* is used to manage the deployment, scaling, and operation of containerized *microservices*. This involves defining *Kubernetes* objects: Pods, Services, and Deployments for each microservice. Three pods for each microservice have been deployed. This process sets up networking to allow communication between *microservices*, utilizing the service discovery mechanisms provided by the orchestration platform.

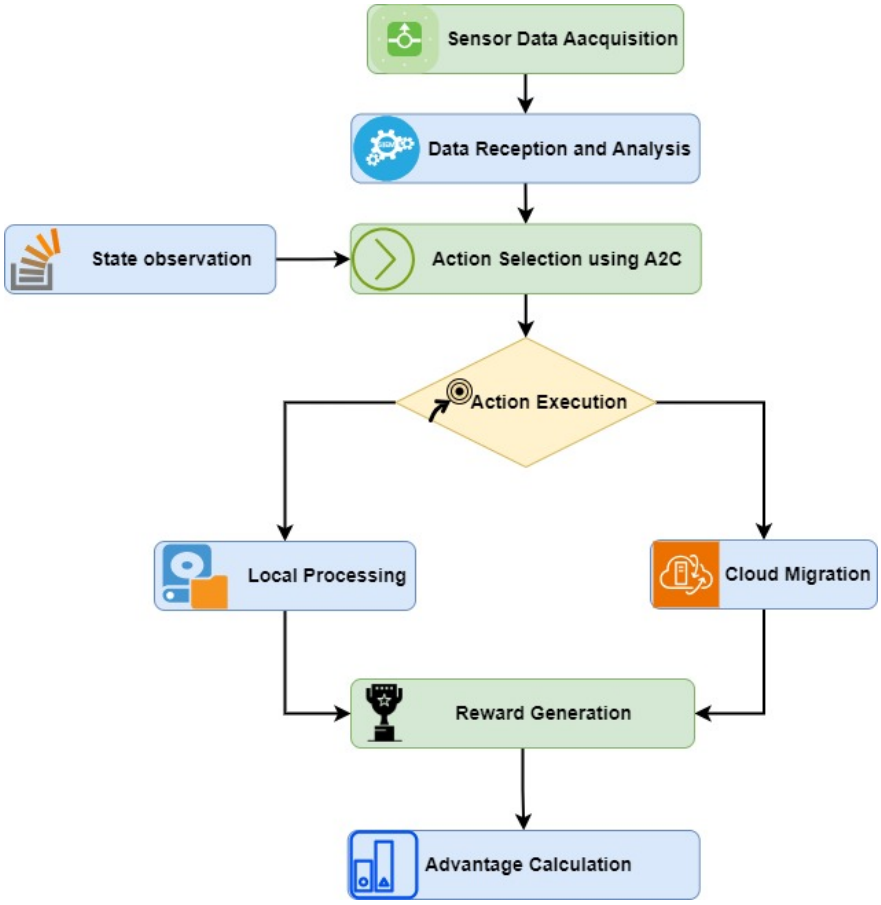


Figure 6.4: Data Processing Workflow

For monitoring and logging solutions of *microservices* and microdata, resource allocation, and migration of MELs, Prometheus and Grafana have been integrated. The process is shown in Figure 6.3.

- *Data Processing Workflow in* $\mu-osmotic$: The data processing workflow includes the following steps:

  i. *Sensor Data Acquisition:* The edge device collects sensor data from various sources.

  ii. *Data Reception and Analysis:* The data receiver component receives the sensor data and extracts its type (e.g., temperature, humidity).

  iii. *State Observation:* The system collects the current state of the edge device, including CPU usage, memory usage, and energy consumption. The state is represented by a vector s_t.

iv. *Action Selection:* It takes the state (s_t) and sensor data type as input and predicts an Action (a) based on the learned policy as given in Equation 6.1:

$$a \sim \pi_\theta(a|s_t) \tag{6.1}$$

Here, π_θ is the probability distribution that defines the likelihood of taking a particular action at the given current state. The value of the current state is estimated by the critic using Equation 6.2:

$$V(s_t) = f_\phi(s_t) \tag{6.2}$$

In this equation, f_ϕ is the value function approximator parameterized by ϕ.

v. *Action Execution:*

- Local Processing ($a = 0$): If the operation is a local processing, the corresponding microservice is selected based on the type of data of the sensor i.e., a microservice designed to handle a specific type of sensor data. The selected microservice on the edge device then processes the data.
- Cloud Migration ($a = 1$): If the operation is cloud migration, the data and the corresponding microservice are forwarded to the cloud service for processing.

vi. *Reward Function Design:* The reward function is crucial in guiding the A2C agent to learn the optimal placement decision. A reward function is proposed that considers both processing efficiency and energy consumption as shown in Equation 6.3:

$$r_t = w_1 \cdot \text{Processing Time } - w_2 \cdot \text{Energy Consumption} \tag{6.3}$$

Here, w_1 and w_2 are positive weight factors determining the relative importance of processing speed and energy usage. The processing time of local processing is estimated based on historical data or analysis, while the energy consumption depends on the selected action as shown in Equation 6.4 and 6.5:

$$\text{Energy Consumption (local)} = E_l(s_t) \tag{6.4}$$

$$\text{Energy Consumption (cloud)} = E_c + E_t(\text{data\_size}) \tag{6.5}$$

- $E_l(s_t)$: The energy consumption of local processing based on the current state (s_t) (e.g., using a CPU power consumption model).
- E_c: Fixed energy cost associated with cloud service invocation
- $E_t(\text{data\_size})$: The energy consumption of transmitting data to the cloud based on data size. (i.e., estimation based on bandwidth and network latency)

The reward function presented incentivizes the A2C agent to learn efficient data placement by minimizing energy consumption when the resources at the edge are available.

vii. *Advantage Calculation:* The advantage function A_t is used to update the policy and value function given as Equation 6.6:

$$A_t = r_t + \gamma V(s_{t+1}) - V(s_t) \tag{6.6}$$

where γ is the discount factor. This advantage function incentivizes the A2C agent to make efficient data placement decisions by minimizing energy consumption when resources at the edge are available.

6.1.3 Layer 3: ITS Layer

Layer 3 is the sensor layer deployed in the field and collects the real-time data. Each deployed sensor monitors the environment and collects the data. The sensors are connected to the system and transmit microdata continuously to the edge layer for further processing. The collected data is then forwarded to the respective microservice for further processing like monitoring, analysis, and decision-making. This layer is an important component that collects data enabling real-time insights and responses to the changing environmental conditions.

6.2 $\mu-osmotic$ System Modeling

This section details $\mu-osmotic$ for dynamic resource management at the edge. The focus is on state representation and Action Space design, which are crucial for effective agent learning and decision-making. Let T_{task} represent a task from an IoT device. The scheduling process having four steps can be formulated as given in Equation 6.7 to 6.11:

- *Resource Check:*

$$R^{\text{edge}}_{k,\text{available}} \geq R_{k,\text{required}}(T_{\text{task}}) \quad \forall k \in \{1, 2, ..., n\} \tag{6.7}$$

- *Scheduling Decision:*

$$\begin{cases} \text{If } R^{\text{edge}}_{k,\text{available}} \geq R_{k,\text{required}}(T_{\text{task}}), & \text{then schedule } T_{\text{task}} \text{ on edge} \\ \text{If } R^{\text{edge}}_{k,\text{available}} < R_{k,\text{required}}(T_{\text{task}}), & \text{then migrate } T_{\text{task}} \text{ and } M_j \\ \text{to cloud} \end{cases} \tag{6.8}$$

- Migration Process:

$$T_{\text{migrated}} = T_{\text{task}} \cup M_j \rightarrow \text{cloud} \tag{6.9}$$

- *Resource Allocation:*

$$R^{\text{edge}}_{k,\text{available}} = R^{\text{edge}}_{k,\text{available}} - R_{k,\text{required}}(T_{\text{scheduled}}) \tag{6.10}$$

$$R^{\text{cloud}}_{k,\text{available}} = R^{\text{cloud}}_{k,\text{available}} - R_{k,\text{required}}(T_{\text{migrated}}) \tag{6.11}$$

Table 6.1 shows the variables and assumptions used in this chapter.

6.2.1 Action Representation

$\mu-osmotic$ employs an actor-critic architecture in which the actor policy $\pi(s, a)$associates states (s) with action probabilities (a), while the critic $V(s)$ evaluates the state-value function. During training, $\mu-osmotic$ enhances the policy by utilizing the benefit function.

In the realm of dynamic resource management at the edge, the agent's state (s) is a vector that denotes the current resource use of the edge device. This may encompass CPU utilization (u_{cpu}), memory utilization (u_{mem}), and energy consumption (u_{energy}) as delineated in Equation 6.12:

Table 6.1: List of Variables

Variable	Description
D_i	IoT device i
M_j	microservice j
R_k	Resource k
C_{edge}	Computational capacity of the edge
C_{cloud}	Computational capacity of the cloud
$R_{k,\text{available}}$	Available units of resource k
$R_{k,\text{required}}$	Required units of resource k by a microservice
T_{task}	Task from an IoT device
$T_{\text{scheduled}}$	Scheduled task
T_{migrated}	Migrated task to the cloud

$$s = [u_{\text{cpu}}, u_{\text{mem}}, u_{\text{energy}}] \tag{6.12}$$

In $\mu-osmotic$, the Action Space (A) defines the set of actions that can be taken in any given state. It is important because it heavily influences how the agent learns and what policy results.

Definition 1 (Action Space in $\mu-osmotic$)**.** *The Action Space, denoted by A, is a finite set of discrete actions available to the agent in a given state. In the context of dynamic resource management at the edge, the Action Space is defined in Equation* 6.13:

$$A = \{0, 1\} \tag{6.13}$$

Where, Action 0 ($a = 0$): This action signifies local sensor data processing on the edge device. Action 1 ($a = 1$): This action indicates cloud migration, where the data is sent to the cloud for processing.

The actor policy $\pi(s, a)$ outputs the probability of selecting each action given the current state as shown in Equation 6.14:

$$\pi(s, a) = P(a \mid s) \tag{6.14}$$

The Advantage function optimizes the policy, where the Advantage A_t is defined as the difference between the actual reward and the expected value of the state as given in Equation 6.15:

$$A_t = r_t + \gamma V(s_{t+1}) - V(s_t) \tag{6.15}$$

In this, r_t denotes the reward obtained after executing an action in state s_t, whereas $V(s_{t+1})$ and $V(s_t)$ represent the value estimations for the subsequent and current states, respectively, with γ signifying the discount factor.

The objective function ($L(\theta, \phi)$) seeks to maximize the anticipated return through the updating of the actor and the critic, as given in Equation 6.16:

$$L(\theta, \phi) = \mathbb{E}t \left[\log \pi\theta(a_t|s_t) \cdot A_t - \frac{1}{2} \cdot (R_t - V_\phi(s_t))^2 \right] \tag{6.16}$$

Where, $\log \pi_\theta(a_t|s_t)$ is the log probability of taking action a_t in state s_t according to the policy π_θ, A_t is the advantage calculated using Equation 6.15, and $(R_t - V_\phi(s_t))^2$ is the mean squared error between the target return R_t and the value estimate $V_\phi(s_t)$ from the critic.

The objective function ensures that the actor is updated and takes actions that lead to higher rewards. The critic is updated to provide more accurate value estimates. The data processing workflow is shown in Figure 6.4

6.2.2 State Representation

Definition 2 (State Representation in $\mu-osmotic$)**.** *The state representation s_t summarizes all relevant information about the environment at time t. It is a comprehensive summary that includes features such as current resource usage, data size, and other factors that affect decision-making. The state representation is an important part of the actor-critic architecture that enables the actor to predict the best course of action in the given situation and the critic to estimate the value of the state. Thus optimizing the performance over time.*

State representation plays an important role in enabling the agents to learn effective policy-making. The state (s) represents the current observable environment and provides the information that is essential for the most appropriate action for the agent.

In the edge resource management scenario, the state representation consists of normalized resource utilization levels:

- *CPU Usage*: This represents the percentage of CPU resources currently used by the edge device. It is determined using the `get_cpu_usage()` function.

- *Memory Usage*: This represents the percentage of memory resources currently used on the edge device, determined by the `get_memory_usage()` function.
- *Energy Usage*: This represents the total energy consumption normalized to a range of 0 to 1. It is calculated by the `get_energy_usage(data_size)` function, which considers the data size being processed and the device's energy consumption characteristics.

The state vector at timestep t (s_t) is defined as Equation 6.17:

$$s_t = \left[\frac{\text{get\_cpu\_usage(t)}}{100}, \frac{\text{get\_memory\_usage(t)}}{100}, \frac{\text{get\_energy\_usage(data\_size)}}{100} \right] \tag{6.17}$$

The state representation provides the middleware with a precise and informative view of current resource utilization on the edge device. The agent, based on this information, decides whether to process the incoming sensor data at the edge (*Action 0*) or to migrate to the cloud (*Action 1*). The critic estimates the value of this state to guide the policy updates.

6.2.3 Reward Function Design

The reward function plays an important role and shapes the agents behavior by providing feedback to the agent. It indicates the effectiveness of their chosen actions in achieving the desired goal. It aims to balance efficient data processing with minimal energy consumption at the edge layer.

Definition 3 (Reward Function in $\mu-osmotic$)**.** *The reward function r_t provides feedback to the agent at each timestep t by quantifying the immediate benefit or penalty of taking a particular Action a_t in state s_t. It is designed to guide the agent toward achieving long-term goals by incentivizing desirable behaviors and discouraging undesirable ones. This reward signal influences both the actor and the critic.*

Reward function based on the chosen action:

- *Local Processing (Action 0)*:
  - A fixed penalty (-0.1) discourages excessive local processing.

  - The reward (Equation 6.18) is further adjusted by subtracting the actual local energy consumption (E_{local}) obtained from the `get_energy_usage_local()` function (Equation 6.19). This encourages the agent to favor local processing when it's energy-efficient.

$$r_t = -0.1 - E_{\text{local}} \tag{6.18}$$

$$E_{\text{local}} = \text{get\_energy\_usage\_local()} \tag{6.19}$$

- *Cloud Migration (Action 1)*:
  - A fixed reward (0.1) encourages the agent to consider cloud migration when necessary.
  - The reward (Equation 6.21) is further adjusted by subtracting the cloud energy consumption (E_{cloud}) calculated based on the data size and a scaling factor 0.1 (Equation 6.20). This discourages unnecessary cloud usage by penalizing high energy consumption.

$$E_{\text{cloud}} = 0.1 \times \frac{\text{data\_size}}{1024} \tag{6.20}$$

$$r_t = 0.1 - E_{\text{cloud}} \tag{6.21}$$

The reward function plays an important role by influencing the actor's policy decisions and the critic's value estimation. The specific design elements influence the agent's learning.

- *Fixed Penalty/Reward*: The fixed penalty for local processing and reward for cloud migration encourage the agent to explore both options. This prevents it from getting stuck in a local optimum (always choosing one action over the other).
- *Energy-aware Adjustments*: Subtracting the actual energy consumption in both scenarios encourages the agent to consider the energy impact of its decisions. The critic uses this reward to estimate the value of each state. The actor adjusts its policy to favor energy-efficient actions.

By effectively combining these elements, the reward function steers the middleware towards learning an optimal policy for dynamic resource management at the edge. The agent balances efficient data processing with minimal energy consumption, making informed decisions based on the current resource utilization and data size.

6.2.4 Observation Space

Definition 4 (Observation Space in $\mu-osmotic$)**.** *The Observation Space is defined as the set of states that can be observed by the agent from an environment s_t. Each s_t in this set tells the environment's current condition based on relevant features such as resource utilization, data size, system performance metrics, etc. It is this Observation Space that defines the inputs to both value and policy networks, enabling the agent to make the best decision by evaluating the state it is in.*

Thus, the Observation Space is defined as a three-dimensional vector as given in Equation 6.22:

$$O \subset \mathbb{R}^3 \tag{6.22}$$

where $\mathbb{R}$ denotes the set of real numbers. Each dimension of this observation vector represents a specific resource utilization metric, given as:

Dimension 1: CPU Usage (cpu_usage): This dimension shows the actual usage of the CPU by the edge device in percentage. This information can be obtained using a system function like `get_cpu_usage()`. The value is generally between 0 and 1 (for example, 0.75 means 75% of CPU is used). Mathematically it can be represented as given in Equation 6.23:

$$\textit{cpu\_usage} = \frac{\text{get\_cpu\_usage}(t)}{100} \tag{6.23}$$

Dimension 2: Memory Usage (memory_usage): This dimension represents the percentage of memory resources currently in use on the edge device. Similar to CPU usage, it can be found using functions such as `get_memory_usage()` and is normalized to a range between 0 and 1. This can be represented mathematically as given in Equation 6.24:

$$memory\_usage = \frac{get\_memory\_usage(t)}{100} \tag{6.24}$$

Dimension 3: Energy Usage (energy_usage): This dimension represents the total energy consumption of the edge device, normalized to a range between 0 and 1. Determining this value can be more complex and involve functions such as `get_energy_usage(data_size)`. This function may take into account the size of the data processed and the energy consumption charac-

teristics of the device to estimate the energy consumption of a specific task. Mathematically it can be represented as given in Equation 6.25:

$$energy\_usage = \frac{get\_energy\_usage(data\_size)}{100} \tag{6.25}$$

By combining these three dimensions, complete state representation (s_t) at a given timestep (t) is obtained as shown in Equation 6.26:

$$s_t = [cpu\_usage, memory\_usage, energy\_usage] \tag{6.26}$$

where, *cpu_usage*, *memory_usage*, and *energy_usage* are calculated using Equations 6.23, 6.24, and 6.25, respectively. The state representation (s_t) provides the agent with a concise yet informative view of the current resource utilization on the edge device. Based on this information, the actor can decide on the best action, and the critic can evaluate the value of the state. Algorithm 6.1 describes the A2C algorithm [105]. Table 6.2 shows the mathematical notation used throughout the section:

6.3 Implementation of $\mu-osmotic$

The IoT testbed for environmental monitoring sensors is set up for synthetic data using Python. The sensors considered for Intelligent Transportation System are: air quality, humidity, temperature, traffic, and energy. The environment sends continuous data to the edge, and the data at the edge is handled in real-time.

The sensor data is transmitted to multiple reinforcement learning (RL) algorithm endpoints, specifically $\mu-osmotic$, Deep Q-Network (DQN) [106], Soft Actor-Critic (SAC) [107], and Proximal Policy Optimization (PPO) [108]. The URLs are initialized for each receiver and five sensor types: humidity, traffic, temperature, energy, and air quality. The data generated by the sensors is packed as a JSON object. This JSON data is transmitted via HTTP POST requests to each RL receiver.

The proposed algorithm (Algorithm 6.2) optimized resource utilization between local devices and cloud migration, for service processing and decision-making.

Table 6.2: Parameters used in the Advantage Actor-Critic (A2C) algorithm and their description.

Parameter	Description
θ	Parameters of the policy network (actor) that determines the actions taken by the agent.
ϕ	Parameters of the value function network (critic) that estimates the expected return for a given state.
τ	Trajectory: A sequence of states, actions, rewards, and log probabilities generated by executing the current policy in the environment.
s_t	State at time step t, representing the environments current condition or observation.
a_t	Action taken by the agent at time step t.
r_t	Reward received by the agent after taking Action a_t in state s_t.
R_t	Discounted cumulative reward (return) from time step t to the end of the trajectory τ.
$V_\phi(s_t)$	Value function estimate of the expected return from state s_t under the policy π_θ.
$\hat{A}_t$	Advantage estimate: The difference between the actual return R_t and the value function estimate $V_\phi(s_t)$.
γ	Discount factor that weights future rewards in computing returns and advantages.
c_1	Coefficient balancing the importance of the value function loss $\mathcal{L}^{\text{value}}(\phi)$ in the total loss function.
α	Learning rate determining the step size of parameter updates during gradient descent.

The model is trained in this custom environment. The training phase entails the ongoing modification of the state according to real-time resource utilization indicators and the model's actions. This setup enables the model to learn an optimal policy for deciding between local processing and cloud migration.

A Flask API is developed to handle incoming sensor data. Upon receiving data, the API parses the sensor type and data. The current state is calculated using the helper functions, and the model predicts whether to process the data locally or migrate it to the cloud. If the model chooses local processing, the data is forwarded to the appropriate microservice based on the sensor type. Otherwise, the data is sent to the cloud. This decision-making process aims to optimize overall energy consumption and resource utilization, thereby enhancing the efficiency of the edge-cloud architecture.

Algorithm 6.1 Advantage Actor-Critic (A2C)

Initialize policy network (actor) parameters θ and value function network (critic) parameters ϕ
for each iteration **do**
    Collect trajectories

$$\tau = \{s_0, a_0, r_0, s_1, a_1, r_1, \ldots, s_T\}$$

    by running policy π_θ in the environment
    for each timestep t **do**
        Compute the discounted cumulative rewards (returns):

$$R_t = \sum_{k=t}^{T} \gamma^{k-t} r_k$$

        Compute the value function $V_\phi(s_t)$ using the critic network
        Compute the advantage estimates $\hat{A}_t$ using the value function:

$$\hat{A}_t = R_t - V_\phi(s_t)$$

    end for
    Compute the policy loss:

$$\mathcal{L}^{\text{policy}}(\theta) = -\hat{\mathbb{E}}_t \left[\log \pi_\theta(a_t|s_t) \hat{A}_t \right]$$

    Compute the value function loss:

$$\mathcal{L}^{\text{value}}(\phi) = \hat{\mathbb{E}}_t \left[(V_\phi(s_t) - R_t)^2 \right]$$

    Compute the total loss:

$$\mathcal{L}(\theta, \phi) = \mathcal{L}^{\text{policy}}(\theta) + c_1 \mathcal{L}^{\text{value}}(\phi)$$

    Update the policy network parameters:

$$\theta \leftarrow \theta - \alpha \nabla_\theta \mathcal{L}^{\text{policy}}(\theta)$$

    Update the value function network parameters: $\phi \leftarrow \phi - \alpha \nabla_\phi \mathcal{L}^{\text{value}}(\phi)$
end for

It ensures a robust system capable of adapting to varying resource availability and workload demands, making it an effective solution for dynamic edge-cloud environments. Similar to this algorithm, processing has been done for other algorithms. The following section details the performance evaluation of the $\mu-osmotic$ framework in comparison to other algorithms.

Algorithm 6.2 Service Processing and Decision Making using $\mu-osmotic$

Initialize:
  Define Prometheus counters.
  Initialize the agent and custom environment.
Functions:
  GETCPUUSAGE: Calculate CPU usage percentage.
  GETMEMORYUSAGE: Calculate memory usage percentage.
  GETENERGYUSAGELOCAL: Measure local energy usage.
  GETENERGYUSAGECLOUD: Estimate cloud energy usage based on data size.
  GETENERGYUSAGE: Calculate total energy usage.
Custom Environment:
  Define Observation Space and Action Space.
  Initialize State with CPU, memory, and energy usage.
Step Function:
  STEP(ACTION)
if Action == 0 **then**
    Perform local processing.
    Calculate reward as $-0.1 -$ GETENERGYUSAGELOCAL().
else
    Migrate service to the cloud.
    Calculate reward as $0.1 -$ GETENERGYUSAGECLOUD($data\_size$).
end if
  Update State with GETCPUUSAGE()/100, GETMEMORYUSAGE()/100, GETENERGYUSAGE($data\_size$)/100.
  Return new State, reward, termination status.
Custom Callback:
  Initialize TensorBoard and CSV logging.
  On each step, log elapsed time, reward, CPU usage, memory usage, local energy usage, and Action.
Initialize the Model:
  Setup environment and monitoring.
  Train the model with specified timesteps and custom callback for logging.
Flask API:
  Define route to receive sensor data.
  RECEIVEDATA(REQUEST)
  Parse incoming data for sensor type and data.
  Decision Making:
  Calculate current State using GETCPUUSAGE()/100, GETMEMORYUSAGE()/100, GETENERGYUSAGE($len(str(sensor\_data))$)/100.
  Predict Action using the model.
if Action == 0 **then**
    Increment local processing counter.
    Forward data to appropriate microservice.
else
    Increment cloud migration counter.
    Trigger migration to cloud service.
end if
  Define route to expose Prometheus metrics.
  Start Flask application to handle requests and process data.
End the Model:

6.4 Results and Discussion

In this section, the performance evaluation of the $\mu-osmotic$ framework is comprehensively investigated by comparing the framework with other algorithms. A series of simulation experiments are comprehended in Kubernetes environment [109] along with Docker [110] for microservice architecture, Python, PyTorch, Prometheus, Tensorboard, Grafana, collections.deque for relay memory buffer and TensorFlow for training neural networks on a server equipped with Intel core i7-12700 CPU @ 2.10 GHz, 16.0 Gb RAM.

6.4.1 Evaluation Metrics

To evaluate the effectiveness of the $\mu-osmotic$ framework, the following metrics are used:

- *Processing Time:* The average time required to process sensor data. This metric reflects the overall efficiency of the system. It is defined as given in Equation 6.27:

$$\text{Processing Time} = \frac{1}{N}\sum_{i=1}^{N} T_i \tag{6.27}$$

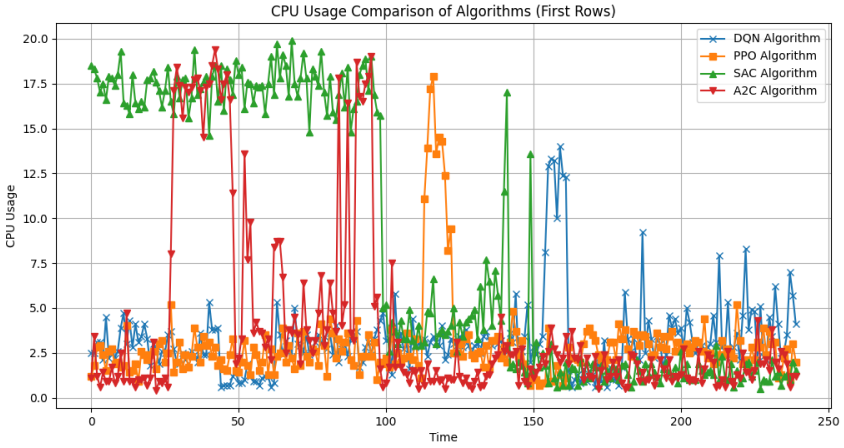
  where N is the total number of sensor data instances, and T_i is the processing time for the i-th instance.

- *Energy Consumption:* The system's total energy consumption, including local processing and cloud migration costs is given by using Equation 6.28:

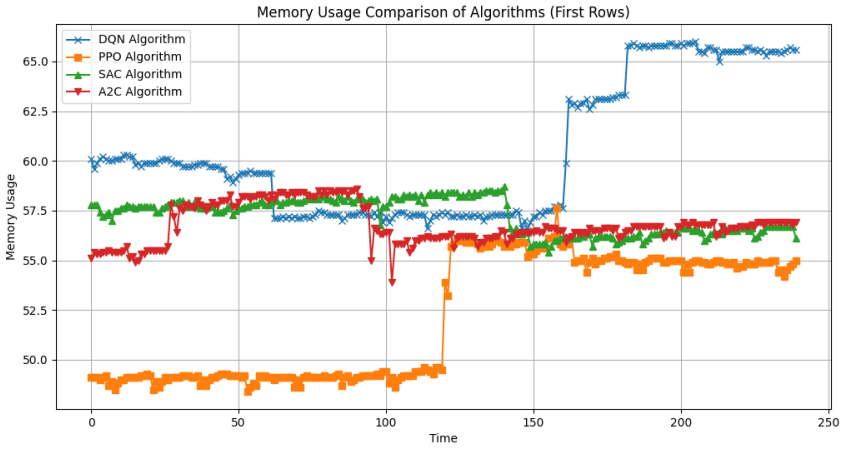
$$\text{Energy Consumption} = \sum_{i=1}^{N} E_{\text{local},i} + \sum_{j=1}^{M} E_{\text{cloud},j} \tag{6.28}$$

  where $E_{\text{local},i}$ is the energy consumption for local processing of the i-th instance, and $E_{\text{cloud},j}$ is the energy consumption for cloud processing of the j-th instance.

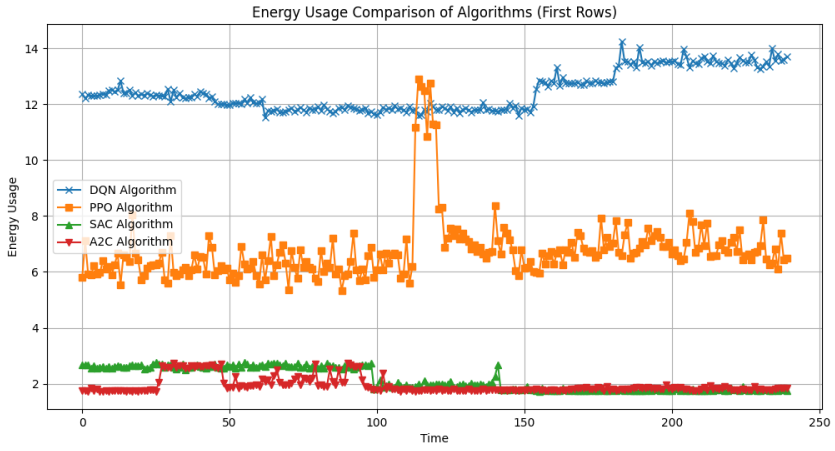
- *Resource Utilization:* The average CPU and memory usage on edge devices. This metric indicates how efficiently the system uses edge resources. It is defined as shown in Equation



(a) Based on CPU Consumption



(b) Based on Memory Consumption



(c) Based on Energy Consumption

Figure 6.5: Comparison of Algorithms based on various parameters

6.29 and 6.30:

$$\text{CPU Utilization} = \frac{1}{T}\int_0^T \text{CPU}(t)\,dt \tag{6.29}$$

$$\text{Memory Utilization} = \frac{1}{T}\int_0^T \text{Memory}(t)\,dt \tag{6.30}$$

where T is the total observation time, CPU(t) and Memory(t) are the CPU and memory usage at time t, respectively.

Table 6.3: Analysis of various algorithms

	DQN	**PPO**	**SAC**	$\mu-osmotic$
Avg Energy	12.459167	7.104788	1.898384	1.839643
Avg CPU	3.229583	1.747162	2.659136	2.149954
Avg Memory	60.423333	61.628793	58.210538	56.895807
Peak Energy	14.240000	14.130000	2.774000	2.748000
Peak CPU	14.000000	17.900000	19.900000	19.400000
Peak Memory	66.000000	65.900000	65.300000	58.900000
Std Energy	0.704144	0.781989	0.231389	0.152157
Std CPU	2.132808	1.717465	4.333035	2.822172
Std Memory	3.397371	6.033759	2.916405	0.825086

The algorithms $\mu-osmotic$, DQN, SAC, and PPO are evaluated based on energy, CPU, and memory usage. $\mu-osmotic$ performs best as the most energy-efficient algorithm with average energy consumption of $1.84J$ and peak energy usage of $2.75J$. It performs best in memory usage with an average memory usage of 56.9 % closely followed by SAC with 58.21%. For CPU usage, it is found that the PPO algorithm performs better with an average CPU usage of 1.75%. whereas, $\mu-osmotic$ uses lightly better with 2.15%. It performs 30% better than the next better-performing SAC algorithm in terms of energy consumption. For memory usage, $\mu-osmotic$ performs 5.84 % better than DQN, 7.68 % better than PPO, and 2.26 % better than the SAC algorithm. Overall, $\mu-osmotic$ performs the best in terms of resource efficiency, consuming the least energy, CPU, and memory on average, and also maintains the least peak values. Figure 6.5 shows the comparison of algorithms based on various metrics. Figure 6.5a shows a comparison of CPU consumption, Figure 6.5b shows a comparison of memory consumption among all algorithms, and Figure 6.5c shows the comparison of energy consumption. From the results, it can be deduced that $\mu-osmotic$ performs better for memory and energy, consuming fewer resources, but, in case of CPU, the PPO

algorithm performs better than $\mu-osmotic$, but, it performs better than other algorithms. Table 6.3 shows the results of the $\mu-osmotic$, SAC, DQN, and PPO algorithms.

The time complexity of the proposed algorithm, O(T*n+m*n), demonstrates its scalability for small to medium workloads, where T represents the number of training timesteps, m denotes the number of incoming API requests, and n corresponds to the size of the data being processed. The space complexity, O(P+S), where P is the model size and S accounts for logged metrics, is manageable given sufficient memory resources. This complexity ensures efficient real-time decision-making, particularly in scenarios involving edge and IoT devices, where moderate data sizes and request rates are typical. However, the linear dependency on n and m highlights potential performance challenges for handling large-scale, high-frequency workloads. To address this, optimizations such as data pre-processing, lightweight models, and parallel processing could be integrated, ensuring the algorithm's adaptability for broader applications in dynamic environments.

6.5 Conclusion

In this chapter, a novel framework for dynamic resource allocation between edge and cloud environments in Intelligent Transportation Systems (ITS) is presented. $\mu-osmotic$ is designed to optimize IoT data processing by making intelligent decisions on the processing of data based on resource availability, CPU usage, memory usage, and energy consumption. The framework utilizes a microservice architecture to achieve flexibility and modularity in deploying services, ensuring effective resource utilization and low latency. This architecture allows for seamless integration of edge and cloud resources, dynamically orchestrating service deployments to meet fluctuating demands and resource availability. The system intelligently migrates MicroELements (MELs) to the cloud layer when necessary, maintaining service quality and reliability. The performance of $\mu-osmotic$ is evaluated and compared with other established algorithms, including SAC, PPO, and DQN. The results demonstrated that $\mu-osmotic$ outperforms the other algorithms in terms of energy and memory consumption. It performs 30.0%, 19.3%, and 2.26% better than SAC in terms of CPU, energy, and memory, respectively It was found that in terms of CPU, PPO performs better, but is closely followed by $\mu-osmotic$. Specifically, $\mu-osmotic$ showed strong performance

across various metrics, including energy efficiency and memory usage, making it a balanced and advantageous choice for optimizing resource allocation in edge-cloud environments.

Chapter 7

Conclusion and Future Work

7.1 Conclusion

The emergence of new computing paradigms driven by the rapid growth in the number of IoT devices that supports real-time, low latency applications has necessitated the need for efficient and scalable resource management systems. The work presented focuses on utilizing Osmotic Computing as a foundational approach that addresses the limitations of traditional cloud and edge computing approaches. By bridging the IoT-Edge-Cloud continuum, Osmotic Computing supports resource orchestration to manage resources at fault-tolerant architecture in modern networks that includes 5G technologies and beyond. It explores frameworks and algorithms for optimizing different task scheduling, resource management, service migration, and decision-making processes for heterogeneous environments contributing to the field of Osmotic Computing. The research has validated the effectiveness of Osmotic Computing in multiple real-world scenarios, such as 5G networks, Intelligent Transportation Systems (ITS), and IoT data processing.

The OCTRA-5G Framework, leverages Osmotic Computing (OC) for efficient task scheduling and resource allocation in 5G networks. By incorporating an Osmotic Layer (OL) at the edge layer, the framework enhances real-time processing, reduces the burden on the core network. The OL consists of key components, including the Osmotic Resource Database (ORD), the Osmotic Service Segregator (OSS), and the Osmotic Resource Scheduler (ORS), which collectively optimize resource allocation by segregating services and facilitating seamless migration when nec-

essary. Sharma *et al.* [65] follows the prnciple of osmosis and presents a fitness-based osmosis algorithm that provides support for osmotic computing by utilizing teh resources at the fog layer. The proposed algorithm segregates the services into microservices and macroservices, but it does not consider whether the resources required to execute those microservices are available at the fog layer or not. The OCTRA-5G framework considers the availability of resources at the edge layer. If the required resource is not available at the edge layer then the microservice is migrated to the cloud layer. The performance of OCTRA-5G was validated through simulations with varying numbers of gNBs, showing a significant reduction in resource allocation time and latency, with improvements ranging from 20.679% to 95.57%. Comparative analysis against traditional algorithms such as FCFS, SJF, and PS demonstrated the superior efficiency of OCTRA-5G, with FCFS emerging as the most effective within the proposed framework. These findings confirm that OCTRA-5G significantly enhances task scheduling and resource allocation efficiency in 5G environments.

The proposed OsCoMIT framework extends the application of Osmotic Computing to Intelligent Transportation Systems (ITS), where the management of computational resources for intelligent vehicles becomes increasingly critical. The proposed Proportional Fairness (PF) algorithm, used for resource allocation in OsCoMIT, outperforms traditional algorithms like FCFS and PS in terms of system utility, memory, and computation resource management. Statistical validation using ANOVA further corroborates the significant performance improvements achieved by OsCoMIT, making it a promising solution for ITS in 5G networks.

Deep Q-Networks (DQN) addresses the challenges of service migration and task offloading in dynamic and heterogeneous environments, aiming to enhance response times and optimize resource utilization. A comprehensive mathematical model that integrates reinforcement learning with the principles of osmotic computing is developed. This model effectively captures the complexities of managing resources and deploying services across both cloud and edge environments. The implementation and extensive simulations demonstrated that the DQN-based approach significantly outperforms other algorithms, as can be seen by the p-values in all the comparisons. Furthermore, we conducted a statistical analysis using the Wilcoxon signed-rank test to validate the effectiveness of our proposed method. The results confirmed the statistical significance of the performance improvements, thereby reinforcing the robustness and reliability of the proposed approach.

The $\mu-osmotic$ framework presented a dynamic resource allocation solution between edge and cloud environments, employing the Advantage Actor-Critic (A2C) algorithm. This framework optimized IoT data processing by making intelligent decisions on whether to process requests at the edge or migrate them to the cloud, based on resource availability and energy consumption. Through detailed simulations, $\mu-osmotic$ consistently outperformed established algorithms like Soft Actor-Critic (SAC), Proximal Policy Optimization (PPO), and DQN in terms of energy efficiency, CPU usage, and memory utilization. Its modular microservice architecture enabled seamless integration of edge and cloud resources, providing a robust solution for scalable IoT deployments.

The research presented in this thesis offers a comprehensive exploration of Osmotic Computing and its potential to revolutionize task scheduling, resource allocation, and service orchestration in complex, heterogeneous environments. The OCTRA-5G, OsCoMIT, DQN-Osmosis, and $\mu-osmotic$ frameworks have shown substantial improvements in performance metrics such as latency reduction, energy efficiency, resource utilization, and system scalability, validating the efficacy of Osmotic Computing in real-world applications. The frameworks proposed throughout this work can be deployed across various sectors, such as 5G networks, ITS, and IoT ecosystems, contributing to advancements in network efficiency, resource optimization, and service quality. The integration of advanced reinforcement learning techniques such as DQN and A2C into these frameworks has further enhanced their adaptability and decision-making capabilities, addressing the dynamic nature of modern computing environments.

7.2 Future Work

While the proposed frameworks focus on improving resource allocation, future work will explore techniques to further minimize latency in highly dynamic environments, particularly for time-sensitive applications. Extending the frameworks to handle larger-scale, more complex environments with a broader range of devices and services is an essential next step. The integration of IoT devices and edge networks brings new challenges in terms of security and data privacy. Future research will address how Osmotic Computing frameworks can incorporate robust security mechanisms to ensure safe and secure resource sharing. The frameworks does not consider the type of task and urgency for the allocation of resources. In future, we plan to consider these points and

propose a framework that considers the task type, ensures the timely completion of services, and adjusts to the fluctuating mobility conditions. Additionally, we aim to explore other performance metrics like bandwidth. Finally, the deployment and testing of these frameworks in real-world networks, beyond simulations, would provide invaluable insights into their practical feasibility, effectiveness, and potential challenges in real-time operations.

References

[1] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE internet of things journal*, vol. 3, no. 5, pp. 637–646, 2016.

[2] M. Villari, M. Fazio, S. Dustdar, O. Rana, and R. Ranjan, "Osmotic computing: A new paradigm for edge/cloud integration," *IEEE Cloud Computing*, vol. 3, no. 6, pp. 76–83, 2016.

[3] L. Carnevale, A. Celesti, A. Galletta, S. Dustdar, and M. Villari, "Osmotic computing as a distributed multi-agent system: The body area network scenario," *Internet of Things*, vol. 5, pp. 130–139, 2019.

[4] A. Y. Ding and M. Janssen, "Opportunities for applications using 5g networks: Requirements, challenges, and outlook," in *Proceedings of the Seventh International Conference on Telecommunications and Remote Sensing*, pp. 27–34, 2018.

[5] IDC, "Worldwide internet of things spending guide, 2023," 2023. Accessed: 2024-05-28.

[6] Statista, "Internet of things (iot) connected devices worldwide 2021-2030," 2024. Accessed: 2024-05-28.

[7] S. Hamdan, M. Ayyash, and S. Almajali, "Edge-computing architectures for internet of things applications: A survey," *Sensors*, vol. 20, no. 22, p. 6441, 2020.

[8] N. I. of Standards and T. (NIST), "Computer security resource center." Accessed on [2024-05-29].

[9] Y. Jadeja and K. Modi, "Cloud computing-concepts, architecture and challenges," in *2012 international conference on computing, electronics and electrical technologies (ICCEET)*, pp. 877–880, IEEE, 2012.

[10] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The case for vm-based cloudlets in mobile computing," *IEEE pervasive Computing*, vol. 8, no. 4, pp. 14–23, 2009.

[11] J. Dilley, B. Maggs, J. Parikh, H. Prokop, R. Sitaraman, and B. Weihl, "Globally distributed content delivery," *IEEE Internet Computing*, vol. 6, no. 5, pp. 50–58, 2002.

[12] B. D. Noble, M. Satyanarayanan, D. Narayanan, J. E. Tilton, J. Flinn, and K. R. Walker, "Agile application-aware adaptation for mobility," *ACM SIGOPS Operating Systems Review*, vol. 31, no. 5, pp. 276–287, 1997.

[13] M. Satyanarayanan, "The emergence of edge computing," *Computer*, vol. 50, no. 1, pp. 30–39, 2017.

[14] K. Ashton *et al.*, "That 'internet of things' thing," *RFID journal*, vol. 22, no. 7, pp. 97–114, 2009.

[15] H. Sundmaeker, P. Guillemin, P. Friess, S. Woelfflé, *et al.*, "Vision and challenges for realising the internet of things," *Cluster of European research projects on the internet of things, European Commision*, vol. 3, no. 3, pp. 34–36, 2010.

[16] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of things (iot): A vision, architectural elements, and future directions," *Future generation computer systems*, vol. 29, no. 7, pp. 1645–1660, 2013.

[17] A. Bhardwaj and C. R. Krishna, "Virtualization in cloud computing: Moving from hypervisor to containerization—a survey," *Arabian Journal for Science and Engineering*, vol. 46, no. 9, pp. 8585–8601, 2021.

[18] H. Shukur, S. Zeebaree, R. Zebari, D. Zeebaree, O. Ahmed, and A. Salih, "Cloud computing virtualization of resources allocation for distributed systems," *Journal of Applied Science and Technology Trends*, vol. 1, no. 2, pp. 98–105, 2020.

[19] K. Inoue, S. Arakawa, and M. Murata, "Network resource planning for evolvability in software-defined infrastructure," *Computer Communications*, vol. 151, pp. 247–256, 2020.

[20] M. Villari, M. Fazio, S. Dustdar, O. Rana, L. Chen, and R. Ranjan, "Software defined membrane: Policy-driven edge and internet of things security," *IEEE Cloud Computing*, vol. 4, no. 4, pp. 92–99, 2017.

[21] M. Villari, M. Fazio, S. Dustdar, O. Rana, D. N. Jha, and R. Ranjan, "Osmosis: The osmotic computing platform for microelements in the cloud, edge, and internet of things," *Computer*, vol. 52, no. 8, pp. 14–26, 2019.

[22] L. Carnevale, A. Celesti, A. Galletta, S. Dustdar, and M. Villari, "From the cloud to edge and iot: A smart orchestration architecture for enabling osmotic computing," pp. 419–424, 2018.

[23] A. Buzachis, A. Galletta, L. Carnevale, A. Celesti, M. Fazio, and M. Villari, "Towards osmotic computing: Analyzing overlay network solutions to optimize the deployment of container-based microservices in fog, edge and iot environments," in *2018 IEEE 2nd International Conference on Fog and Edge Computing (ICFEC)*, pp. 1–10, IEEE, 2018.

[24] M. Villari, A. Galletta, A. Celesti, L. Carnevale, and M. Fazio, "Osmotic computing: software defined membranes meet private/federated blockchains," in *2018 IEEE Symposium on Computers and Communications (ISCC)*, pp. 01292–01297, IEEE, 2018.

[25] M. Nardelli, S. Nastic, S. Dustdar, M. Villari, and R. Ranjan, "Osmotic flow: Osmotic computing+ iot workflow," *IEEE Cloud Computing*, vol. 4, no. 2, pp. 68–75, 2017.

[26] M. Maksimović, "The role of osmotic computing in internet of things," pp. 1–4, 2018.

[27] J. Thönes, "Microservices," *IEEE software*, vol. 32, no. 1, pp. 116–116, 2015.

[28] X. Larrucea, I. Santamaria, R. Colomo-Palacios, and C. Ebert, "Microservices," *IEEE Software*, vol. 35, no. 3, pp. 96–100, 2018.

[29] A. Souza, N. Cacho, A. Noor, P. P. Jayaraman, A. Romanovsky, and R. Ranjan, "Osmotic monitoring of microservices between the edge and cloud," in *2018 IEEE 20th International Conference on High Performance Computing and Communications; IEEE 16th International Conference on Smart City; IEEE 4th International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*, pp. 758–765, IEEE, 2018.

[30] K. Wüst and A. Gervais, "Do you need a blockchain?," in *2018 crypto valley conference on blockchain technology (CVCBT)*, pp. 45–54, IEEE, 2018.

[31] T. T. A. Dinh, R. Liu, M. Zhang, G. Chen, B. C. Ooi, and J. Wang, "Untangling blockchain: A data processing view of blockchain systems," *IEEE Transactions on Knowledge and Data Engineering*, vol. 30, no. 7, pp. 1366–1385, 2018.

[32] T. Ahram, A. Sargolzaei, S. Sargolzaei, J. Daniels, and B. Amaba, "Blockchain technology innovations," in *2017 IEEE Technology & Engineering Management Conference (TEMSCON)*, pp. 137–141, 2017.

[33] M. Belotti, N. Božić, G. Pujolle, and S. Secci, "A vademecum on blockchain technologies: When, which, and how," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 4, pp. 3796–3838, 2019.

[34] J. Yli-Huumo, D. Ko, S. Choi, S. Park, and K. Smolander, "Where is current research on blockchain technology?—a systematic review," *PloS one*, vol. 11, no. 10, p. e0163477, 2016.

[35] A. Buzachis and M. Villari, "Basic principles of osmotic computing: secure and dependable microelements (mels) orchestration leveraging blockchain facilities," in *2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion)*, pp. 47–52, IEEE, 2018.

[36] V. Sharma, A. Kataria, J. McAllister, A. Weedon, and R. Bennett, "Detecting unpredictable adversaries in the industrial network with blockchain," in *Physical layer security for 6G networks*, Institution of Engineering and Technology (IET), 2024.

[37] V. Hassija, S. Batra, V. Chamola, T. Anand, P. Goyal, N. Goyal, and M. Guizani, "A blockchain and deep neural networks-based secure framework for enhanced crop protection," *Ad Hoc Networks*, vol. 119, p. 102537, 2021.

[38] S. Rasool, A. Saleem, M. Iqbal, T. Dagiuklas, A. K. Bashir, S. Mumtaz, and S. Al Otaibi, "Blockchain-enabled reliable osmotic computing for cloud of things: Applications and challenges," *IEEE Internet of Things Magazine*, vol. 3, no. 2, pp. 63–67, 2020.

[39] A. Ruggeri, A. Celesti, M. Fazio, and M. Villari, "An innovative blockchain-based orchestrator for osmotic computing," *Journal of Grid Computing*, vol. 20, pp. 1–17, 2022.

[40] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[41] P. Gupta, A. Sharma, and R. Jindal, "Scalable machine-learning algorithms for big data analytics: a comprehensive review," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 6, no. 6, pp. 194–214, 2016.

[42] Y. Guo, Y. Liu, A. Oerlemans, S. Lao, S. Wu, and M. S. Lew, "Deep learning for visual understanding: A review," *Neurocomputing*, vol. 187, pp. 27–48, 2016.

[43] A. Morshed, P. P. Jayaraman, T. Sellis, D. Georgakopoulos, M. Villari, and R. Ranjan, "Deep osmosis: Holistic distributed deep learning in osmotic computing," *IEEE Cloud Computing*, vol. 4, no. 6, pp. 22–32, 2017.

[44] A. Pacheco, P. Cano, E. Flores, E. Trujillo, and P. Marquez, "A smart classroom based on deep learning and osmotic iot computing," in *2018 Congreso internacional de innovación y tendencias en ingeniería (CONIITI)*, pp. 1–5, IEEE, 2018.

[45] L. Gurgel, A. Souza, N. Cacho, and F. Lopes, "Deep learning distribution model using osmotic computing," in *2023 IEEE International Smart Cities Conference (ISC2)*, pp. 1–7, IEEE, 2023.

[46] E. Oyekanlu, "Distributed osmotic computing approach to implementation of explainable predictive deep learning at industrial iot network edges with real-time adaptive wavelet graphs," in *2018 IEEE First International Conference on Artificial Intelligence and Knowledge Engineering (AIKE)*, pp. 179–188, IEEE, 2018.

[47] Z. N. Absardi and R. Javidan, "Iot traffic management using deep learning based on osmotic cloud to edge computing," *Telecommunication Systems*, pp. 1–17, 2024.

[48] J. Goyal, K. Singla, Akashdeep, and S. Singh, "A survey of wireless communication technologies from 1g to 5g," in *Second International Conference on Computer Networks and Communication Technologies: ICCNCT 2019*, pp. 613–624, Springer, 2020.

[49] G. Peng, S. Wang, Y. Huang, T. Huang, and Y. Liu, "Enabling deterministic tasks with multi-access edge computing in 5g networks," *IEEE Communications Magazine*, vol. 60, no. 8, pp. 36–42, 2022.

[50] R. K. Gupta and R. Misra, "Machine learning-based slice allocation algorithms in 5g networks," in *2019 International Conference on Advances in Computing, Communication and Control (ICAC3)*, pp. 1–4, 2019.

[51] G. Choudhary and V. Sharma, "A survey on the security and the evolution of osmotic and catalytic computing for 5g networks," *5G enabled secure wireless networks*, pp. 69–102, 2019.

[52] Monia, N. Sharma, and R. Dhir, "Fog computing: An overview of iot applications with security issues and challenges," in *2021 9th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions) (ICRITO)*, pp. 1–8, 2021.

[53] J. Zhao, H. Xu, H. Liu, J. Wu, Y. Zheng, and D. Wu, "Detection and tracking of pedestrians and vehicles using roadside lidar sensors," *Transportation research part C: emerging technologies*, vol. 100, pp. 68–87, 2019.

[54] N. Buch, S. A. Velastin, and J. Orwell, "A review of computer vision techniques for the analysis of urban traffic," *IEEE Transactions on intelligent transportation systems*, vol. 12, no. 3, pp. 920–939, 2011.

[55] B. Filocamo, A. Galletta, M. Fazio, J. A. Ruiz, M. Á. Sotelo, and M. Villari, "An innovative osmotic computing framework for self adapting city traffic in autonomous vehicle environment," in *2018 IEEE Symposium on Computers and Communications (ISCC)*, IEEE, 2018.

[56] V. Kapoor, D. Saxena, V. Raychoudhury, and S. Kumar, "Real time building and maintaining causal congestion graph for intelligent traffic management," in *2018 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*, pp. 770–775, 2018.

[57] L. Liu, M. Zhao, M. Yu, M. A. Jan, D. Lan, and A. Taherkordi, "Mobility-aware multi-hop task offloading for autonomous driving in vehicular edge computing and networks," *IEEE Transactions on Intelligent Transportation Systems*, vol. 24, no. 2, pp. 2169–2182, 2023.

[58] V. Sharma, D. N. K. Jayakody, and M. Qaraqe, "Osmotic computing-based service migration and resource scheduling in mobile augmented reality networks (marn)," *Future Generation Computer Systems*, vol. 102, pp. 723–737, 2020.

[59] V. Sharma, T. G. Tan, S. Singh, and P. K. Sharma, "Optimal and privacy-aware resource management in artificial intelligence of things using osmotic computing," *IEEE Transactions on Industrial Informatics*, vol. 18, no. 5, pp. 3377–3386, 2021.

[60] E. Carlini, L. Carnevale, M. Coppola, P. Dazzi, G. Mencagli, D. Talia, and M. Villari, "An osmotic ecosystem for data streaming applications in smart cities," in *Proceedings of the 1st Workshop on Flexible Resource and Application Management on the Edge*, pp. 27–31, 2020.

[61] B. Neha, S. K. Panda, and P. K. Sahu, "An efficient task mapping algorithm for osmotic computing-based ecosystem," *International Journal of Information Technology*, pp. 1–6, 2021.

[62] V. Sharma, I. You, R. Kumar, and P. Kim, "Computational offloading for efficient trust management in pervasive online social networks using osmotic computing," *IEEE Access*, vol. 5, pp. 5084–5103, 2017.

[63] K. Kaur, S. Garg, G. Kaddoum, S. H. Ahmed, and D. N. K. Jayakody, "En-osco: Energy-aware osmotic computing framework using hyper-heuristics," in *Proceedings of the ACM MobiHoc workshop on pervasive systems in the IoT Era*, pp. 19–24, 2019.

[64] C. Sicari, A. Galletta, A. Celesti, M. Fazio, and M. Villari, "An osmotic computing enabled domain naming system (oce-dns) for distributed service relocation between cloud and edge," *Computers & Electrical Engineering*, vol. 96, p. 107578, 2021.

[65] V. Sharma, K. Srinivasan, D. N. K. Jayakody, O. Rana, and R. Kumar, "Managing service-heterogeneity using osmotic computing," *arXiv preprint arXiv:1704.04213*, 2017.

[66] M. Gamal, R. Rizk, H. Mahdi, and B. E. Elnaghi, "Osmotic bio-inspired load balancing algorithm in cloud computing," *IEEE Access*, vol. 7, pp. 42735–42744, 2019.

[67] A. Longo, A. De Matteis, and M. Zappatore, "Urban pollution monitoring based on mobile crowd sensing: An osmotic computing approach," in *2018 IEEE 4th International Conference on Collaboration and Internet Computing (CIC)*, pp. 380–387, IEEE, 2018.

[68] A. Buzachis, D. Boruta, M. Villari, and J. Spillner, "Modeling and emulation of an osmotic computing ecosystem using osmotictoolkit," in *Proceedings of the 2021 Australasian Computer Science Week Multiconference*, pp. 1–9, 2021.

[69] T. Rausch, S. Dustdar, and R. Ranjan, "Osmotic message-oriented middleware for the internet of things," *IEEE Cloud Computing*, vol. 5, no. 2, pp. 17–25, 2018.

[70] A. Buzachis, A. Galletta, A. Celesti, L. Carnevale, and M. Villari, "Towards osmotic computing: a blue-green strategy for the fast re-deployment of microservices," in *2019 IEEE Symposium on Computers and Communications (ISCC)*, pp. 1–6, IEEE, 2019.

[71] A. Longo, M. Zappatore, and A. De Matteis, "An osmotic computing infrastructure for urban pollution monitoring," *Software: Practice and Experience*, vol. 50, no. 5, pp. 533–557, 2020.

[72] T. Szydlo, A. Szabala, N. Kordiumov, K. Siuzdak, L. Wolski, K. Alwasel, F. Habeeb, and R. Ranjan, "Iotsim-osmosis-res: Towards autonomic renewable energy-aware osmotic computing," *Software: Practice and Experience*, vol. 52, no. 7, pp. 1698–1716, 2022.

[73] A. Galletta, M. Fazio, A. Celesti, and M. Villari, "On the applicability of secret share algorithms for osmotic computing," in *2020 IEEE Symposium on Computers and Communications (ISCC)*, pp. 1–6, IEEE, 2020.

[74] A. Galletta, C. Sicari, A. Celesti, and M. Villari, "Oce-dns: an innovative osmotic computing enabled domain name system," in *2021 IEEE/ACM 21st International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*, pp. 642–648, IEEE, 2021.

[75] M. Fazio, A. Buzachis, A. Galletta, A. Celesti, J. Wan, A. Longo, and M. Villari, "A map-reduce approach for the dijkstra algorithm in sdn over osmotic computing systems," *International Journal of Parallel Programming*, vol. 49, pp. 347–375, 2021.

[76] G. Morabito, C. Sicari, A. Ruggeri, A. Celesti, and L. Carnevale, "Secure-by-design serverless workflows on the edge–cloud continuum through the osmotic computing paradigm," *Internet of Things*, vol. 22, p. 100737, 2023.

[77] A. Swain, S. K. Panda, and S. K. Sathua, "An efficient physiological record system using osmotic computing paradigm," in *2023 IEEE 20th India Council International Conference (INDICON)*, pp. 1030–1035, IEEE, 2023.

[78] P. B. Reddy and C. Sudhakar, "An osmotic approach-based dynamic deadline-aware task offloading in edge–fog–cloud computing environment," *The Journal of Supercomputing*, vol. 79, no. 18, pp. 20938–20960, 2023.

[79] L. L. Mlotshwa, S. M. Makura, N. M. Karie, and V. R. Kebande, "Opportunistic security architecture for osmotic computing paradigm in dynamic iot-edge's resource diffusion," in *Proceedings of the 2nd International Conference on Intelligent and Innovative Computing Applications*, pp. 1–7, 2020.

[80] A. Souza, Z. Wen, N. Cacho, A. Romanovsky, P. James, and R. Ranjan, "Using osmotic services composition for dynamic load balancing of smart city applications," in *2018 IEEE 11th Conference on Service-Oriented Computing and Applications (SOCA)*, pp. 145–152, IEEE, 2018.

[81] S. Hati and D. De, "Obsc: osmotic blockchain based framework for smart city environment," in *2020 Fifth International Conference on Research in Computational Intelligence and Communication Networks (ICRCICN)*, pp. 143–148, IEEE, 2020.

[82] A. Souza, N. Cacho, T. Batista, and R. Ranjan, "Sapparchi: an osmotic platform to execute scalable applications on smart city environments," in *2022 IEEE 15th International Conference on Cloud Computing (CLOUD)*, pp. 289–298, IEEE, 2022.

[83] A. Buzachis, M. Fazio, A. Celesti, and M. Villari, "Osmotic flow deployment leveraging faas capabilities," in *Internet and Distributed Computing Systems: 12th International Conference, IDCS 2019, Naples, Italy, October 10–12, 2019, Proceedings 12*, pp. 391–401, Springer, 2019.

[84] B. Neha, S. K. Panda, P. K. Sahu, and D. Taniar, "Energy and latency-balanced osmotic-offloading algorithm for healthcare systems," *Internet of Things*, vol. 26, p. 101176, 2024.

[85] S. Forti, I. Lera, C. Guerrero, and A. Brogi, "Osmotic management of distributed complex systems: a declarative decentralised approach," *Journal of Software: Evolution and Process*, vol. 34, no. 10, p. e2405, 2022.

[86] A. Souza, N. Cacho, and T. Batista, "Fogmotic: Applying osmotic data services to improve database operations on smartcity environments," in *2022 IEEE International Smart Cities Conference (ISC2)*, pp. 1–7, IEEE, 2022.

[87] G. Loseto, F. Scioscia, M. Ruta, F. Gramegna, S. Ieva, C. Fasciano, I. Bilenchi, and D. Loconte, "Osmotic cloud-edge intelligence for iot-based cyber-physical systems," *Sensors*, vol. 22, no. 6, p. 2166, 2022.

[88] E. Oyekanlu, "Osmotic collaborative computing for machine learning and cybersecurity applications in industrial iot networks and cyber physical systems with gaussian mixture models," in *2018 IEEE 4th International Conference on Collaboration and Internet Computing (CIC)*, pp. 326–335, IEEE, 2018.

[89] V. Sharma, J. Kim, S. Kwon, I. You, and F.-Y. Leu, "An overview of 802.21 a-2012 and its incorporation into iot-fog networks using osmotic framework," in *IoT as a Service: Third International Conference, IoTaaS 2017, Taichung, Taiwan, September 20–22, 2017, Proceedings 3*, pp. 64–72, Springer, 2018.

[90] K. Okafor, F. Ugwoke, and A. Obayi, "Evaluation of virtualized osmotic cloud network using discrete event branch-and-bound heuristics," in *2017 IEEE 3rd International Conference on Electro-Technology for National Development (NIGERCON)*, pp. 425–437, IEEE, 2017.

[91] R. Douhara, Y.-F. Hsu, T. Yoshihisa, K. Matsuda, and M. Matsuoka, "Kubernetes-based workload allocation optimizer for minimizing power consumption of computing system with neural network," in *2020 International Conference on Computational Science and Computational Intelligence (CSCI)*, pp. 1269–1275, IEEE, 2020.

[92] J. Spillner, P. Gkikopoulos, A. Buzachis, and M. Villari, "Rule-based resource matchmaking for composite application deployments across iot-fog-cloud continuums," in *2020 IEEE/ACM 13th International Conference on Utility and Cloud Computing (UCC)*, pp. 336–341, IEEE, 2020.

[93] R. Tapwal, P. K. Deb, S. Misra, and S. K. Pal, "Shadows: Blockchain virtualization for interoperable computations in iiot environments," *IEEE Transactions on Computers*, vol. 72, no. 3, pp. 868–879, 2022.

[94] O. Debauche, S. Mahmoudi, and A. Guttadauria, "A new edge computing architecture for iot and multimedia data management. information 2022, 13, 89," *mdpi. com*, 2022.

[95] R. Xu, W. Jin, and D. H. Kim, "Knowledge-based edge computing framework based on coap and http for enabling heterogeneous connectivity," *Personal and Ubiquitous Computing*, vol. 26, no. 2, pp. 329–344, 2022.

[96] B. Heiden, T. Knabe, V. Alieksieiev, and B. Tonino-Heiden, "Production orgitonization: Some principles of the central/decentral dichotomy and a witness application example," in *Future of Information and Communication Conference*, pp. 517–529, Springer, 2022.

[97] R. G. Prakash, R. Shankar, and S. Duraisamy, "Fupa: future utilization prediction algorithm based load balancing scheme for optimal vm migration in cloud computing," in *2020 Fourth international conference on inventive systems and control (ICISC)*, pp. 638–644, IEEE, 2020.

[98] F. G. Mármol and M. Q. Kuhnen, "Reputation-based web service orchestration in cloud computing: A survey," *Concurrency and Computation: Practice and Experience*, vol. 27, no. 9, pp. 2390–2412, 2015.

[99] Z. Wen, R. Yang, P. Garraghan, T. Lin, J. Xu, and M. Rovatsos, "Fog orchestration for internet of things services," *IEEE Internet Computing*, vol. 21, no. 2, pp. 16–24, 2017.

[100] N. Sharma, D. Sharma, P. Singh, P. Jain, and S. Kumawat, "5g nr total transmit power — maximum cell transmit power — reference signal power," 2019.

[101] S. Malik and R. Atri, "5G New Radio Technology Introduction and its Throughput Capabilities," Tech. Rep. RR-1776-NYCEDC, RAND Corporation, Santa Monica, CA, USA, June 24 2018. [Online].

[102] Z. Qin, Z. Cheng, C. Lin, Z. Lu, and L. Wang, "Optimal workload allocation for edge computing network using application prediction," *Wireless Communications and Mobile Computing*, vol. 2021, 2021.

[103] A. Kaur, R. Kumar, and S. Saxena, "Osmotic computing and related challenges: a survey," in *2020 Sixth International Conference on Parallel, Distributed and Grid Computing (PDGC)*, pp. 378–383, IEEE, 2020.

[104] R. F. Woolson, "Wilcoxon signed-rank test," *Encyclopedia of Biostatistics*, vol. 8, 2005.

[105] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[106] Y. Wang, L. Chen, H. Zhou, X. Zhou, Z. Zheng, Q. Zeng, L. Jiang, and L. Lu, "Flexible transmission network expansion planning based on dqn algorithm," *Energies*, vol. 14, no. 7, p. 1944, 2021.

[107] T. Haarnoja, A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel, *et al.*, "Soft actor-critic algorithms and applications," *arXiv preprint arXiv:1812.05905*, 2018.

[108] Z. Wang, Z. Hu, Y. Yang, and Y. Yin, "Research on ppo algorithm in solving auv path planning problems," in *2021 2nd International Seminar on Artificial Intelligence, Networking and Information Technology (AINIT)*, pp. 73–79, IEEE, 2021.

[109] R. Muddinagiri, S. Ambavane, and S. Bayas, "Self-hosted kubernetes: Deploying docker containers locally with minikube," in *2019 international conference on innovative trends and advances in engineering and technology (ICITAET)*, pp. 239–243, IEEE, 2019.

[110] I. Docker, "Docker," *lınea].[Junio de 2017]. Disponible en: https://www. docker. com/what-docker*, 2020.

List of Publications

Science Citation Index Journal

1. Kaur, A., Kumar, R., & Saxena, S. "OCTRA-5G: Osmotic computing-based task scheduling and resource allocation framework for 5G." Concurrency and Computation: Practice and Experience, 34(28), 1-18(2024). https://doi.org/10.1002/cpe.7369.

2. Kaur, A., Saxena, S. & Kumar, R., "OsCoMIT: Osmotic computing-based service management for intelligent transportation systems in 5G network." Cluster Computing, 27, 5403–5421 (2024). https://doi.org/10.1007/s10586-023-04217-1.

3. Kaur, A., Kumar, R., & Saxena, S. "$\mu-osmotic$: A Microservice Management Framework for Intelligent Transportation System", IEEE Transactions on Intelligent Transportation Systems (communicated), (2024).

International Conference

1. Kaur, A., Kumar, R., & Saxena, S., "Osmotic computing and related challenges: A survey," Sixth International Conference on Parallel, Distributed and Grid Computing (PDGC), pp. 378–383, IEEE, 2020, JUIT, Waknaghat, India.

2. Kaur, A., Kumar, R., & Saxena, S., "Optimizing Service Migration and Task Offloading in Osmotic Computing with Deep Q-Networks" 16th International Conference on Ubiquitous Computing and Ambient Intelligence November 27th to November 29th, 2024 - Ulster University - Belfast (Northern Ireland, United Kingdom) (Accepted)