UNIVERSITY OF TARTU
Faculty of Science and Technology
Institute of Computer Science
Computer Science Curriculum

Jeyhun Abbasov

Resource optimization with DRL-driven real time service placement strategy in Edge-Cloud continuum

Master's Thesis (30 ECTS)

Supervisor: Chinmaya Kumar Dehury, Ph.D.

Tartu 2023

Resource optimization with DRL-driven real time service placement strategy in Edge-Cloud continuum

Abstract: The growth of Internet of Things (IoT) devices and the need for data intensive applications has led to Edge-Fog-Cloud architecture, known as Edge-Cloud continuum. Cloud computing is utilized for handling and keeping the large amounts of data produced by IoT devices. One of the major limitations of Cloud computing is network latency. Due to these limitations, the Fog computing is introduced. Fog computing provides near real-time services and saves network resources. However, Fog computing has lower resource capacity comparing to Cloud computing. Edge computing is an extension of Fog Computing where data is processed closer to the source. In our study, we present a Deep Reinforcement Learning (DRL) real time service distribution solution without compromising the Quality of Services (QoS) in Edge-Cloud continuum. The services are offered by Fog and Cloud environments. The request that is coming from user is sliced in Edge and distributed between the Fog and Cloud environments using DRL. The proposed DRL algorithm is implemented and evaluated in terms of its success rate, distribution of service request slices, and etc. Furthermore, the study delves into the intricate dynamics of three distinct data-intensive applications, revealing insights into their performance and resource utilization within the Edge-Cloud continuum.

Keywords: Cloud Computing, Fog Computing, Edge Computing, Smart Gateway, Deep Reinforcement Learning, Service Delivery, Service Dispersal

CERCS: P170 Computer science, numerical analysis, systems, control

Ressursi optimeerimine DRL-põhise reaalajas teenuste paigutuse strateegiaga Edge-Cloud kontinuums

Lühikokkuvõte: Asjade Interneti (IoT) seadmete kasv ja vajadus andmeintensiivsete rakenduste järele on viinud Edge-Fog-Cloud arhitektuuri, mida tuntakse Edge-Cloud kontinuums. Pilvandmetöötlust kasutatakse suurte andmehulkade haldamiseks, mida IoT seadmed toodavad. Üks suuremaid pilvandmetöötluse piiranguid on võrgu latentsus. Nende piirangute tõttu on kasutusele võetud Uduandmetöötlus. Uduandmetöötlus pakub reaalajas teenuseid ja säästab võrgu ressursse. Siiski on uduandmetöötlusel võrreldes pilvandmetöötlusega madalamad ressursimahtude võimalused. Servandmetö ötlus on Uduandmetöötluse laiendus, kus andmeid töödeldakse lähemal allikale. Meie uuringus esitleme SüvaTugevusõppe (DRL) reaalajas teenuste jaotuse lahendust, mis ei ohusta teenuste kvaliteeti (QoS) Edge-Cloud kontinuums. Teenuseid pakuvad Udu- ja Pilvikeskkonnad. Kasutajalt saabuv päring lõigatakse Edge'is ja jaotatakse Udu- ja Pilvikeskkondade vahel, kasutades DRL-i. Pakutav DRL algoritm on rakendatud ja hinnatud selle edukuse määra, teenusepäringu lõikude jaotuse jms osas. Lisaks uurib uuring kolme erineva andmeintensiivse rakenduse keerulist dünaamikat, paljastades teadmisi nende jõudlusest ja ressursside kasutamisest Edge-Cloud kontinuums.

Võtmesõnad: Pilvandmetöötlus, Uduarvutus, Servandmetöötlus, Nutikas Värav, Sügav Õpetamisreinforcement, Teenuse Tarnimine, Teenuse Jaotus

CERCS:P170 Arvutiteadus, arvutusmeetodid, süsteemid, juhtimine

Contents

List of Figures

List of Tables

1 Introduction

With the rise of the Internet of Things (IoT), smart homes have become a reality, offering greater convenience and control over our daily lives [2]. IoT devices are equipped with sensors and actuators that allow us to monitor and control various aspects of our home environment, such as temperature, lighting, and security, from anywhere at any time through dedicated mobile applications or web services. This has revolutionized the way we live, work, and interact with our homes. In addition, the increasing demand for computing and storage resources from these smart home services is driving the development of cloud computing infrastructure, making it possible to store and process large amounts of data in real-time, further enhancing the capabilities of our smart homes.

Cloud computing is comprised of a vast variety of powerful physical servers that provide an almost infinite number of computation, storage, and networking capabilities to enable IoT services. In cloud computing, there are typically three categories of services: IaaS, PaaS and SaaS. IaaS (infrastructure as a service) is the most basic type of cloud computing. It provides access to a virtualized environment that includes computer resources such as servers and virtual machines, storage, networks, and operating systems. PaaS (platform as a service) provides an on demand platform for developing, testing, deploying, and managing software without having to worry about establishing or maintaining the necessary IT infrastructure such as servers, storage, networking, and databases. SaaS (software as a service) is a vendor-provided cloud-based software that may be accessed via a web browser. Customers may subscribe to it and enjoy a variety of applications like as accounting, sales, invoicing, and more. The advantages of SaaS include low cost, great performance, and dependability. Despite its advantages, cloud computing has drawbacks, such as geographical constraints and sophisticated network architecture [3]. This is where Fog Computing comes in, filling the gap between the traditional cloud computing and IoT devices. It extends cloud computing closer to the edge devices, improving mobility support, enabling location awareness, and reducing latency for better performance and user experience. [4]

Fog computing is proposed as a solution to address the previously mentioned issue. Fog computing solves the issues of low latency, location awareness and improves quality-of-service for real time and streaming applications by being deployed at the edge of the network. Examples of fog computing applications can be found in industrial automation, transportation and networks that involve sensors and actuators. In addition, the fog infrastructure accommodates different types of devices, such as user end devices, access points, edge routers, and switches. Fog computing is ideal for real-time big data analytics and provides benefits in various fields like entertainment, advertising, personal computing, etc. by enabling densely distributed data collection points. By utilizing local resources in fog computing, a user's service requirements can be met with less reliance on distant cloud resources, leading to improved performance. Implementing fog computing can help reduce service latency, improve overall Service Response Time (SRT), reduce network congestion, decrease energy consumption, and enhance system security by utilizing resources at the edge organized as fog nodes [5, 6].

Edge computing is an extension of Fog Computing where data is processed closer to the source, either on the device, sensor, or a nearby server. One of the crucial elements in an edge computing is the Smart Gateway. The Smart Gateway is responsible for managing various IoT operations, such as collecting and preprocessing data, filtering and reformatting it, uploading relevant data to the fog/cloud, monitoring IoT objects and sensors, monitoring energy consumption, ensuring data security and privacy, and overall service management. Additionally, The Smart Gateway can play a key role in efficient service dispersal process. Communication between IoT and Smart Gateway can either be direct or through base station(s). There are two types (Single-hop communication, Multi-hop communication) of Smart Gateway-based communication. In single-hop connectivity, the sensors and IoT devices are directly connected to the Smart Gateway. The Smart Gateway then collects the data and sends it to either the Fog or Cloud. This type of communication is typically used in smaller networks where the sensor nodes have limited roles, such as in smart health or healthcare applications. Quick monitoring and response is possible through direct communication with the Smart Gateway. M2M communication takes place in this scenario, and the Smart Gateway can perform data refinement, filtering, trimming, and security measures, service dispersal process, depending on the application needs and in conjunction with fog computing. The extent of this type of communication depends on the capabilities of the Smart Gateway device. In a scenario where multiple sensor networks and IoTs are connected, direct connections become unfeasible. These networks and IoTs have their own base stations and sink nodes, and the gateway collects data from these. This creates a multi-hop communication setup, where nodes are more diverse and spread out and data is more heterogeneous, requiring more processing and analysis from the gateway. Sink nodes provide an additional layer of communication and make underlying sensors and things a "black box," adding security. Security can be customized according to the IoT and WSN, and sink nodes can manage sensor networks based on their constraints. The gateway must handle heterogeneous data from various devices, IoTs, and WSNs, requiring transcoding and interoperability. This can either be achieved through an intelligent gateway or through Fog computing resources. This setup is suitable for large scale IoTs/WSNs and mobile objects, such as vehicle tracking IoTs and environmental monitoring. [7] For the purpose of keeping things simple, we have selected to use Single-hop communication in our Smart Gateway setup. As an illustrative example, Figure 1 illustrates an abstract view of Cloud-Fog-Edge architecture.

In our approach, the service workload is divided into smaller sub-services by Smart Gateway and is shared between fog and cloud servers. The fog node processes some slices of the service request and the rest is sending to the cloud for processing. The decision of which slices to be processed by the fog and cloud depends on various factors such as user proximity, user priority, service sensitivity, service priority, etc. Determining the correct distribution of the service between fog and cloud becomes increasingly complex as the service complexity grows, leading to a new research challenge.

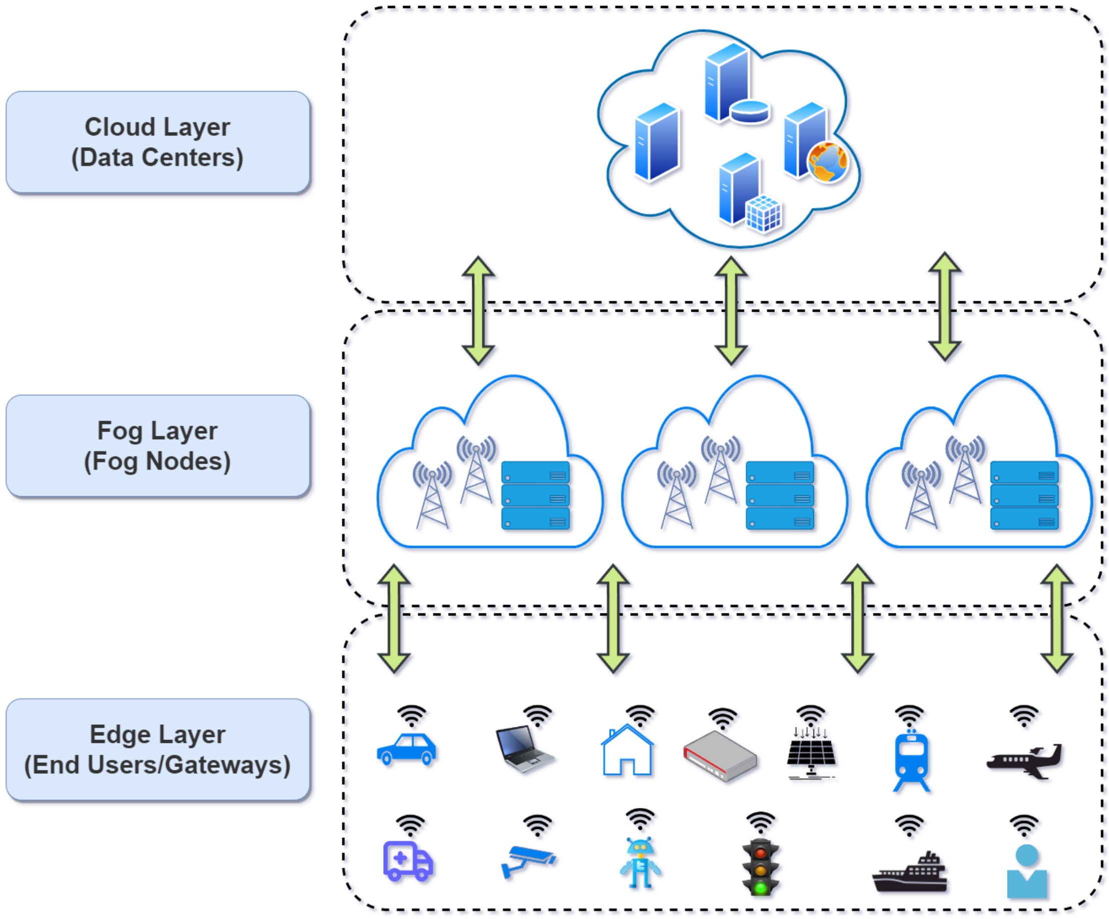
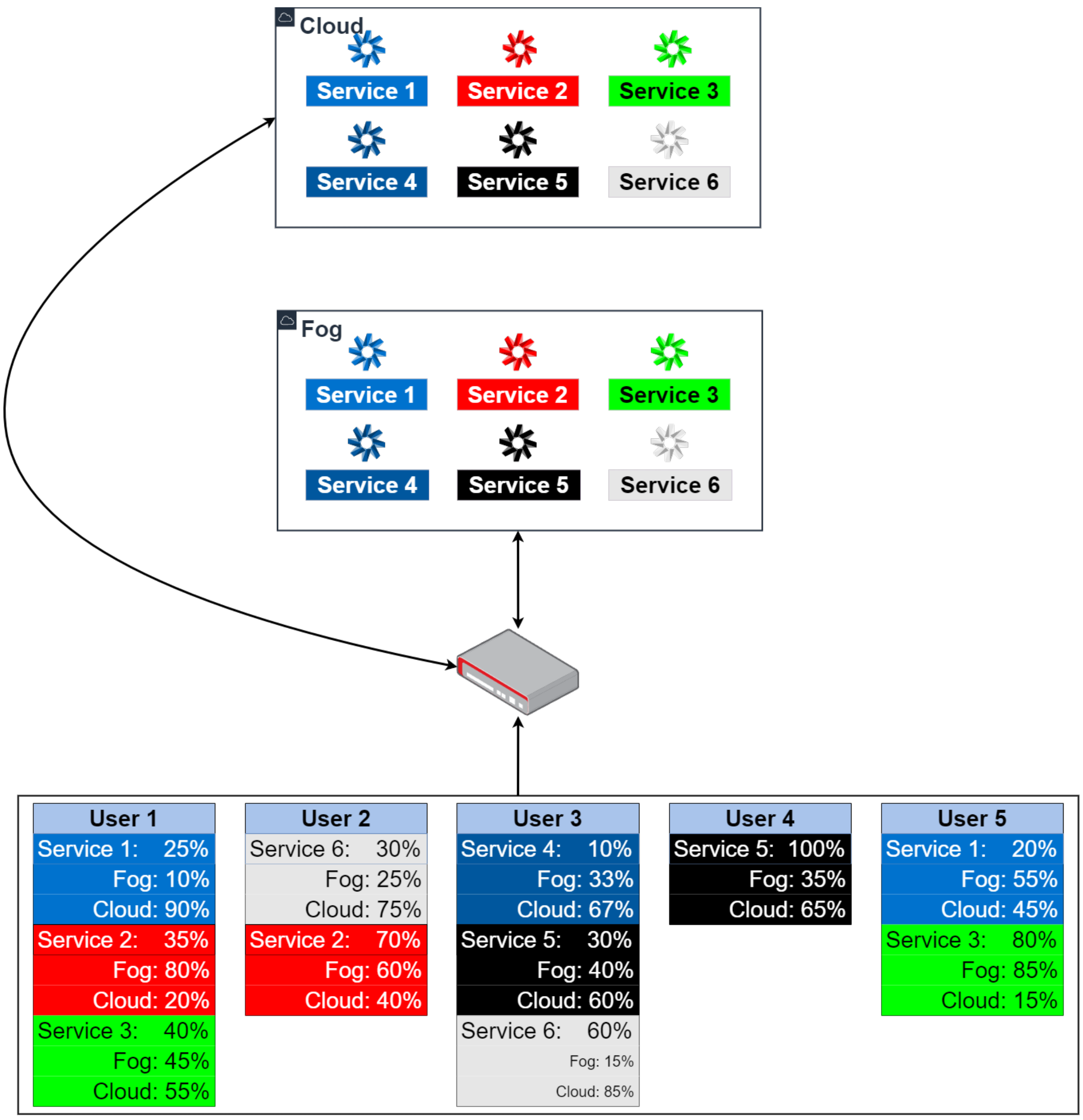


Figure 1. Abstract view of Cloud-Fog-Edge architecture.

1.1 Motivation

The allocation of services between the fog and cloud is managed by the Smart Gateway, as previously discussed, to provide services to the users through both fog and cloud environments. As seen in Figure 2, the division of services between fog and cloud for different services and users is not consistent, and an individual user may utilize a variety of services. For example, the User1 is using three services named as Service1, Service2, and Service3. The service request from User1 includes a request for 25% of Service1, a request for 35% of Service2, and a request for 40% of Service3. The 45% and 55%



User 1	User 2	User 3	User 4	User 5
Service 1: 25%	Service 6: 30%	Service 4: 10%	Service 5: 100%	Service 1: 20%
Fog: 10%	Fog: 25%	Fog: 33%	Fog: 35%	Fog: 55%
Cloud: 90%	Cloud: 75%	Cloud: 67%	Cloud: 65%	Cloud: 45%
Service 2: 35%	Service 2: 70%	Service 5: 30%		Service 3: 80%
Fog: 80%	Fog: 60%	Fog: 40%		Fog: 85%
Cloud: 20%	Cloud: 40%	Cloud: 60%		Cloud: 15%
Service 3: 40%		Service 6: 60%		
Fog: 45%		Fog: 15%		
Cloud: 55%		Cloud: 85%		

Figure 2. An example motivational scenario.

of Service3 that is requested by User1 is handled by Fog and Cloud, respectively. The distribution of service workload between fog and cloud servers can vary, depending on various factors such as the sensitivity of the service, user location, available resources, and others. For example, a critical service with low tolerance for latency cannot be assigned to the cloud, whereas a fog node experiencing resource constraints may need to delegate tasks to the cloud. Balancing the workload between fog and cloud environments becomes challenging when taking into account multiple parameters. This motivates us to investigate a new solution for distributing service workload effectively between fog and cloud computing platforms while maintaining quality of service. In this thesis, to overcome the aforementioned issue, we are adopting deep reinforcement learning.

1.2 Goal and Contributions

The goals of this paper are to enhance QoS with reduced service latency by delegating a portion of the service to the cloud, serve the maximum number of users in real-time by letting the cloud handle part of the service request, and efficiently utilize the fog node's limited resources by incorporating the cloud in the real-time service delivery process. The main contribution of this work can be summarized as follows:

- This paper outlines the problem of distributing users' services among fog and cloud environments;
- Proposed a new deep reinforcement learning (DRL) based method to distribute service requests to fog and cloud computing environments;
- Performed comprehensive experiments employing real-world data-intensive applications to assess the performance of the proposed DRL model.

1.3 Outline

This chapter provides an overview of IoT, cloud computing, fog computing, smart gateway, the significance of service distribution, and how deep reinforcement learning can help resolve the issue. Chapter 2 delves into the background of service distribution and reviews the current state of the art. In Chapter 3, we present our solution to the service distribution problem by discussing the architecture, applications, methods, and algorithms used. Chapter 4 details the experiment process and results are discussed in Chapter 5. Finally, Chapter 7 concludes the work with a summary and future direction.

2 State of the Art

In this section, we delve into the current state of the art in the field of service delivery. Our discussion begins by providing an overview of the background of service delivery in both fog and cloud computing environments in Section 2.1. This section aims to give the reader a comprehensive understanding of the context in which the current work is being carried out. Following this, in Section 2.2, we delve into the related works that have been done in the area of service dispersal methods within the cloud domain. This section aims to give the reader an understanding of the prior efforts-made in this field and the gaps that the current work aims to fill. Finally, in Section 2.3, we provide a summary of the key points covered in this section, offering a clear and concise overview of the state of the art in the field of service delivery.

2.1 Background

2.1.1 Cloud Computing

Recently, the Internet of Things (IoT) has made a significant impact on our society by converting everyday objects like wearable, transportation, and augmented reality into communicating devices, bringing both new challenges and opportunities [8]. It is evident that the existing infrastructure will not be capable of handling the vast amount of data generated as the number of devices connected to the network is expected to reach over 50 billion by 2020, according to Cisco [9]. The current Cloud infrastructure is inadequate to support a substantial amount of the existing IoT applications, due to three main reasons. First, the large volume of generated data makes it difficult to transfer it from the end-devices where it is created to the Cloud servers where it is processed. This is due to limitations in bandwidth, processing overhead, and transmission costs. Second, applications that require real-time analysis, such as online gaming and video applications, can be negatively impacted by the significant end-to-end delay from end-devices to the faraway Cloud servers [10]. Lastly, privacy and security concerns may dictate that certain sensitive data must not traverse the entire Internet, making it necessary to process it closer to its source [11].

A solution has been identified to address the problems mentioned above by avoiding network congestion, reducing communication costs, and minimizing the delay in data transfer. This new concept that leverages the advantages of the Cloud and the decentralization of service processing on edge devices is called Fog Computing [12].

2.1.2 Fog Computing

Fog Computing extends Cloud Computing by locating processing, analysis, and storage closer to the source of requests, at the edge of the network. The goal is to minimize the data sent to the Cloud and lower the time delay and computational expenses [13]. One of the main challenges of Fog Computing, a new paradigm, is the management of services, specifically the problem of determining the appropriate placement for them. Efficiently deploying services on available Fog nodes is a significant challenge facing the widespread adoption of Fog Computing. The difficulty in deploying services on Fog nodes arises from the fact that they are geographically dispersed, have limited resources, and can change dynamically, adding to the complexity of the issue. Therefore, it is crucial to determine an efficient, effective, and equitable way of allocating resources to IoT applications in order to provide end-to-end guaranteed services to the users.

Additionally, the focus may vary depending on the situation and the priorities, including aspects such as resource utilization [14], Quality of Service (QoS) [15, 16], and Quality of Experience (QoE) [17]. Over the past few years, multiple efforts have been made to address this challenge. An efficient service placement has been proposed by taking into account different characteristics, assumptions, and strategies.

2.1.3 Edge Computing

Edge computing extends Fog Computing, and refers to the practice of processing data near the edge of the network, closer to the source of the data. In this paradigm, processing takes place on the device or sensor itself or on a nearby server. The goal of edge computing is to reduce the amount of data that needs to be transmitted to the fog or cloud for processing, which reduces latency and improves performance. One of the main benefits of edge computing is its ability to support mission-critical applications that require low latency and high reliability. For example, edge computing can be used in the healthcare industry to provide real-time monitoring and analysis of patient data, enabling medical professionals to make informed decisions quickly.Edge computing architectures can vary, with some relying on edge devices like smartphones, and others using dedicated edge servers. [18]

Overall, by bringing processing and storage capabilities closer to the source of data, edge computing enables new applications and services that were not possible before, while also providing faster and more reliable processing and analysis of data.

2.2 Related Work

The placement of services that are effective in fog and cloud environments is a complex task that is influenced by several factors. The first of these is the heterogeneous and resource-constrained nature of most Fog nodes, which have limited capacities. The second factor is the dynamic nature of the environment, where resources can change over time and workload can be vary. Thirdly, the issue is made even more complicated by the widespread distribution of fog devices across a large-scale infrastructure. The problem of SPP (service placement problem) in a fog and cloud environment is a difficult task due to various specificities and constraints. Within the literature, we can identify the primary approaches employed to solve this problem as heuristic approach, machine learning based algorithms, and deep reinforcement learning based algorithms.

2.2.1 Heuristic Approaches

Wang and Li et. al [19] proposes a solution for improving task scheduling in fog computing using a hybrid heuristic algorithm. However, this study is limited to only using bandwidth and energy consumption. Zahoor et al. [20] proposes a model for managing resources in SGs using cloud-fog computing, and uses several load balancing algorithms including HABACO to optimize response time. This study is not using the deep reinforcement learning approach as our study.

Rafique et al. [21] proposes a scheduling strategy to optimize resource utilization and reduce average response time in fog computing. The proposed approach outperforms other techniques in terms of energy consumption and execution time. However, it is required to apply deep reinforcement learning techniques to improve fog-IoT resource management. Yasmeen et al. [22] proposes a three-layered cloud and fog-based model to decrease consumer load and power generation system. Resource balancing is achieved through the use of three algorithms: Round Robin, throttled, and Particle Swarm Optimization with Simulated Annealing (PSOSA). Unlike our study, this study does not take reducing latency into account.

Yadav et al. [23] proposes a hybrid algorithm that combines Genetic Algorithm and Particle Swarm Optimization to allocate services efficiently in a fog computing environment. The algorithm minimizes total makespan and energy consumption for IoT applications. Ren et al. [24] suggests a model to decrease energy consumption and efficiently manage energy resources in fog systems. The proposed architecture aims to manage resources in a way that reduces losses and ensures quality of service. However, both these studies is only based on energy consumption unlike ours.

Hosseinioun et al. [25] suggests a method for saving energy in the fog computing environment using the Dynamic Voltage and Frequency Scaling (DVFS) technique. A hybrid algorithm, IWO-CA, is used to create valid task sequences. Javanmardi et al. [26] The article describes the FPFTS fog task scheduler, which uses particle swarm optimization and fuzzy theory to optimize application loop delay and network utilization. The drawback of the study is that it requires developing a mobility-aware task scheduling algorithm. The study clearly lacks the benefit of other features like resource demand as in our study.

Xu et al. [27] proposes the LBP-ACS algorithm for task scheduling in cloud-fog environments, which considers priority constraints, energy consumption, and mixed deadlines. The algorithm reduces energy consumption and failure rates while ensuring reasonable scheduling length. The study is limited to using only energy consumption and deadlines. Djemai et al. [28] proposes a strategy for IoT services placement in Fog architecture using a Discrete Particles Swarm Optimization algorithm. The placement strategy minimizes application delay violations and considers energy consumption. This study only considers minimizing energy consumption.

2.2.2 Machine Learning based Algorithms

Rahbari and Nickray et al. [29] presents a Module Placement method by Classification and regression tree Algorithm (MPCA) for selecting the best FDs for modules. MPCA selects the best FDs based on decision parameters such as authentication, confidentiality, integrity, availability, capacity, speed, and cost. As opposed to our study, these studies are focusing on the module placement, there is no slicing method for modules.

Bashir et al. [30] proposes a dynamic resource allocation strategy for cloud, fog nodes, and users. The strategy uses TOPSIS (technique for order performance by similarity to ideal solution) to rank the fog nodes and logistic regression to calculate the load of individual fog nodes. This study though uses the machine learning approach for dynamic resource allocation but is limited to slicing.

Ferrández-Pastor et al. [31] proposes a method to design smart services based on the edge computing paradigm. The approach addresses interoperability and scalability issues of existing designs and implements energy management, security system, climate control, and information services subsystems on embedded devices. This study, though, makes use of the machine learning approach but fails to minimizing latency like our study.

He et al. [32] proposes a new fog computing model with functional modules can reduce problems related to dedicated computing infrastructure and slow response times in cloud computing. However, this study is limited to only using response time. Priyabhashana et al. [33] compares the use of various activation functions on a TensorFlow-based neural network model built using the Keras library. The proposed fog station was evaluated, and the experiment results showed the feasibility, efficiency, and applicability of the system. This study is focused on cpu and ram usage.

Alsaffar et al. [34] proposes an architecture for IoT service delegation and resource allocation that combines fog and cloud computing. The authors propose an algorithm to allocate resources to meet SLA and QoS requirements and optimize big data distribution. Yadav et al. [35] presents a new technique for task allocation that minimizes response time and system cost using clustering. The proposed technique uses Fuzzy C-Means clustering and Hungarian method for task allocations. However, this study is only based on minimizing response time unlike ours.

2.2.3 Deep Reinforcement Learning based Algorithms

Farhat et al. [36] proposes R-learning model that aims to decrease the cloud's load by utilizing available fog resources in different locations. The model is demonstrating its ability to adapt to user demand and efficiently use fog resources. However, this study is limited to only using user demands. Orhean et al. [37] presents a reinforcement learning algorithm for solving the scheduling problem in distributed systems. The proposed algorithm considers the heterogeneity of nodes and the dependency graph of tasks for determining a scheduling policy that minimizes execution time. Unlike our study, this study only take execution time into account.

Tang et al. [38] proposes a new architecture and algorithms for container migration to support mobility tasks with different application requirements. The proposed algorithms consider hosting mobile application tasks in containers of corresponding fog nodes, and modeling container migration strategies as multiple dimensional Markov Decision Process spaces. This study though uses the deep reinforcement learning approach but is limited due to slicing.

Gazori et al. [39] proposes a Double Deep Q-Learning (DDQL)-based scheduling algorithm for task scheduling of fog-based IoT applications. The aim is to minimize long-term service delay and computation cost under resource and deadline constraints. As opposed to our study, these studies are focusing on minimizing service delay, and deadline constraints. Alam et al. [40] proposes a deep Q-learning based autonomic management framework for computation offloading in mobile edge/fog. The framework uses a distributed edge/fog network controller to allocate resources and minimize latency of service computing while maintaining energy efficiency. However, this study is only based on energy consumption and minimizing service latency unlike ours.

Zhang et al. [41] proposes a double deep Q-learning model for energy-efficient edge scheduling (DDQ-EES). The model includes a generated network and a target network for producing Q-values, and rectified linear units (ReLU) as the activation function to avoid gradient vanishing. This study only considers energy-efficiency unlike our study. Lu et al. [42] proposes a QoE model for computation offloading that considers service latency, energy consumption, and task success rate. The study clearly lacks the benefit of other features like resource demand as in our study.

Wang et al. [43] proposes a Q-learning-based hierarchical service tree placement strategy to optimize the net utility in large networks with complex service structures. The study is limited to using only network bandwidth feature. Dehury and Srirama et al. [44, 45] a reinforcement learning-based (RLPSD) and a deep reinforcement learning-based (DRLSD-FC) service delivery mechanism that distributes users' service requests between fog and cloud environments to to minimize the workload on fog while maintaining service quality. The algorithms consider user constraints such as distance from fog, service sensitivity, and other metrics. However, the study is limited because it does not address service-placement in real-time manner as our study, and service dispersal mechanism inside of fog nodes, in our study we use smart gateway in order to do service dispersal jobs.

2.3 Summary

In this chapter, we first introduced with background information on service placement in fog and cloud environments. Then we discussed the current state-of-the-art in service dispersal methods for both environments. We also saw the different approaches used to answer the SP problem in related works section. Finally, we saw what is new in our studies compared to the current state-of-the-art. Table 1 provides an overview of the relevant literature utilized in this study. The first column displays the name of each related work, while the second column outlines the methodology employed in each case. The third, fourth, and fifth columns indicate whether or not the authors of each work took User Priority, Service/Task Priority, and Input/Task Slicing into consideration, respectively. In these columns, "Y" signifies that these factors were considered, while "N" indicates otherwise. The final column denotes the specific resource parameters that were taken into account by each author. Here, "B" corresponds to network bandwidth, "C" to CPU, "PS" to processor speed, "M" to memory/RAM, "E" to energy/power consumption, and "S" to storage. In the next chapter, we will see our approach to the problem of service placement in fog and cloud environments.

Table 1. Summary of Related Work

Paper	**Method**	**User Priority**	**Service/Task Priority**	**Input/Task Slicing**	**Resource**
[19]	Heuristic Algorithm	N	N	N	B, E
[20]	Heuristic Algorithm	N	N	N	PS, M
[21]	Hybrid Heuristic	N	N	N	E
[22]	Heuristic Algorithm	N	N	Y	E
[23]	Genetic Heuristic	N	N	Y	E
[24]	Hybrid Heuristic	N	N	N	B, E
[25]	Hybrid Heuristic	N	N	N	E
[26]	Hybrid Heuristic	N	Y	N	B, E
[27]	Heuristic Algorithm	N	Y	N	E
[28]	Swarm Optimization	N	N	N	E
[29]	Regression Tree	N	N	N	C, E, M, S
[30]	Logistic Regression	N	N	N	B, C, M
[31]	Machine Learning	N	N	N	E
[32]	Machine Learning	N	N	N	C, M
[33]	Deep Neural Networks	N	N	N	C, E
[34]	Linearized Decision Tree	N	N	Y	C
[35]	Machine Learning	N	N	N	C
[36]	R-learning	N	N	N	B
[37]	R-learning	N	N	N	B, S
[38]	Deep R-learning	N	Y	N	B, E
[39]	Deep R-learning	N	N	N	C, E, M, S
[40]	Deep R-learning	N	N	N	E
[41]	Double Deep R-learning	N	N	N	E
[42]	Deep R-learning	N	N	N	B, C, E
[43]	Deep R-learning	N	N	N	B
[44]	Deep R-learning	Y	Y	Y	C, M
[45]	Deep R-learning	Y	Y	Y	C, M

"Y" means feature considered, "N" means not. "B" - network bandwidth, "C" - CPU, "PS" - processor speed, "M" - memory/RAM, "E" - energy/power consumption, and "S" - storage.

3 Methodology

Multiple methods can be used to solve the same issue. In this chapter, we outline our approach to addressing the challenge of distributing service requests between fog and cloud environment. We start our discussion with architecture of the system in Section 3.1, followed by modelling the users in Section 3.2, and modeling the services and mathematical representation and analysis of key parameters Section 3.3. Section 3.4 delves into the deep reinforcement learning, covering topics like state space, action space, and reward calculation. Finally, we wrap up this chapter with a summary in Section 3.5.

3.1 System Architecture

In Figure 2, the fog and cloud environments offer multiple services to users. Users can obtain services by submitting a Service Request (SR) to the smart gateway. The Service Request can be partitioned, which is an assumption that can be demonstrated with various real world examples involving data intensive applications generated by IoT devices, distributed among the fog and cloud environments. The smart gateway collaborates with both fog and cloud environments to meet service resource demands.

3.1.1 Smart Gateway

The Smart Gateway, represented by G, plays a vital role in ensuring the efficient distribution of user requests within the fog and cloud computing environments. G's primary task is to break down user requests into smaller, manageable slices and allocate these slices across the fog and cloud environments. Each slice is a logical subdivision of the service request or input data, allowing for efficient processing and improved overall performance. For instance, if a user uploads a 250MB file, G can divide this file into five slices with varying sizes, each containing [10MB, 25MB, 30MB, 70MB, 115MB] of the data. This way, the processing of the file is distributed and can be handled more efficiently.

To allocate these slices effectively, G considers several essential factors that influence the performance and user experience. These factors include user priority, service sensitivity, service priority, resource demand, and more. For example, service priority takes into account the importance of the request, with higher-priority services being assigned more resources; service sensitivity is another key factor, as some services may require more stringent privacy or security measures. By evaluating these factors, G is able to distribute the slices effectively, ensuring optimal performance and user satisfaction in the fog and cloud computing environments.

In our system, Fog Node and Cloud node are designed to offer a set of m number of services, denoted by $S = \{s_1, s_2, s_3, ..., s_m\}$, where each service is represented as s_k, with $0 < k \leq m$.

3.1.2 Fog and Cloud Environments

The Fog Node, represented by F, is a crucial component within the fog and cloud computing environment, outfitted with a limited amount of resources such as CPU and RAM. The main function of the fog node is to offer these services to as many nearby users as possible. By functioning as an intermediary between smart gateway and the cloud, the fog node helps to reduce the load on the cloud environment.

On the other hand, the cloud node (represented as C) has a large collection of top-notch servers that are all connected together. This setup gives it almost limitless processing power and storage capacity. Unlike the fog node, the cloud node is designed to handle more resource-intensive tasks, supporting a multitude of users and services simultaneously. While the F is responsible for receiving and processing requests from smart gateway, some tasks are may delegated to the cloud node when required. This could occur when the fog node is unable to handle the demand or when specific tasks necessitate the capabilities of the cloud node. By utilizing the strengths of both the fog node and the cloud node, the fog and cloud computing environments can ensure optimal performance, efficient resource allocation, and a seamless user experience.

3.2 User Modelling

The fog and cloud computing environments offer a diverse range of services to cater to the various needs of all connected users. We use the notation $U = \{u_1, u_2, u_3, ..., u_n\}$ to represent the group of all users who are connected to the Smart Gateway. Each user is represented by u_i, where $0 < i \leq n$. Users have the flexibility to send requests for multiple service types, depending on their specific requirements. The boolean variable $u_i(s_k)$ is used to indicate if user u_i is currently availing the service s_k. $u_i(s_k)$ can be represented mathematically as following [45]:

$$u_i(s_k) = \begin{cases} 1 \text{ - if user } u_i \text{ is availing the service } s_k \\ 0 \text{ - otherwise} \end{cases} \tag{1}$$

This variable helps the Smart Gateway, or G, track the active services for each user, allowing for optimal service delivery. As users interact with the fog and cloud computing environments, two major parameters play a crucial role: distance priority and latency priority.

3.2.1 User Distance Priority

The geographic distance between a user and the G impacts the overall latency and performance of the system. As the distance increases, the time taken for data to travel between the user and the gateway may also increase, potentially leading to a slower user experience. A user is positioned within the maximum range of the G, with each user and the G having associated location parameters. The distance between a user and the Smart Gateway can be computed using the Haversine formula as follows:

$$d_i = 2r \cdot \arcsin\left(\sqrt{\sin^2\left(\frac{p_g - p_i}{2}\right) + \cos(p_i) \cdot \cos(p_g) \cdot \sin^2\left(\frac{q_g - q_i}{2}\right)}\right) \tag{2}$$

In this formula, r represents the radius of the sphere, (p_i, q_i) represents the latitude and longitude of user u_i, and (p_g, q_g) corresponds to the latitude and longitude of the Smart Gateway. The distance of a user has to be less than the maximum range of the Smart Gateway [45]. The maximum range of the Smart Gateway is represented by D. By considering the D and the user's distance (d_i), the distance priority of a user can be determined as follows:

$$P_i^d = \frac{d_i}{D} \tag{3}$$

All users who are equidistant from the G will have the same priority based on distance, with those farther away given higher priority.

3.2.2 User Latency Priority

Communication latency refers to the duration required for a user's request to be sent to the Smart Gateway and for the subsequent response to be returned. This latency can be influenced by a variety of factors, such as network congestion or constraints in the user's device. Users closer to the Smart Gateway may not necessarily experience lower latency. Let l_i represent the Round Trip Delay (RTD) time for user u_i. Based on this value, the latency-based priority can be calculated as:

$$P_i^l = \frac{l_i}{\max(l_i, 0 < i \leq n)} \tag{4}$$

The value of P_i^l is relative and depends on the latency values for all connected users. Users with higher values of P_i^l are assigned the highest priority.

3.2.3 User Priority

We explored the methods for calculating user distance and latency priorities in the previous subsections. Taking into account these priorities, as shown in Equation 3 and Equation 4, we can determine the combined priority of a user using the following approach:

$$P_i = P_i^d + P_i^l \tag{5}$$

By considering both distance and latency priorities, we can develop a more comprehensive understanding of each user's needs within the fog and cloud computing environment. This overall priority calculation allows for more effective service request allocation and ensures that users with more urgent or demanding requirements are given the necessary attention.

3.3 Service Modelling

Here, we delve into service modeling and its various aspects. When users send a service request, several properties of the services must be considered, such as the size of the service request and the demand for the service request, etc. A service request (SR) from a user is divided into $\hat{s}_i^k$ service request slices, with the number of slices being greater than 0. If the value $\hat{s}_i^k = 0$, it signifies that the user is not utilizing the service. On the other hand, the value $\hat{s}_i^k = 1$ indicates that the service request is unsliced and will be processed solely by the Fog Node (FN).

Each SR is also associated with a size, which represents the magnitude of the SR or the size of the input data. Let $u_i(\hat{s}_k)$ denote the size of the SR of type s_k from user u_i. The size of the SR slices can vary, meaning that $u_i(\hat{s}_{k,j_1})$ may not equal $u_i(\hat{s}_{k,j_2})$ if j_1 and j_2 are different, where $0 < j_1, j_2 < \hat{s}_i^k$, and $\forall u_i \in U$. Additionally, the sizes of SRs differ, and the size of a specific SR slice is distinct from that of another SR slice. As a result, $u_1(\hat{s}_k, j)$ is not equal to $u_2(\hat{s}_k, j)$, where $\forall u_1, u_2 \in U$, and $i_1 \neq i_2$.

3.3.1 Service Demand

The demand for service s_k from all connected users at the Smart Gateway is represented by $SD(s_k)$. This value is calculated using the total number of users availing the service and the cumulative size of all SRs from all users. Mathematically, the service demand (SD) takes into account the boolean variable $u_i(s_k)$, which indicates if a user is availing the service s_k, and $u_i(\hat{s}_k)$, which represents the size of the service requests of type s_k from user u_i.

$$SD(s_k) = \sum_{u_i \in U} [u_i(\hat{s}_k) \cdot u_i(s_k)] \tag{6}$$

By considering these factors, the Smart Gateway can better understand each user's specific needs, allowing for more efficient resource allocation and improved overall system performance.

3.3.2 Service Resource Demand

Service Resource Demand is an important aspect that illustrates the required resources for a service to function properly and meet user expectations. The resource demand for service requests (SR) is represented by $u_i(R_k(x))$. Resource demand can be categorized into two distinct types: $x = \{CPU, MEM\}$. These categories help in evaluating the performance and efficiency of the system across different resource dimensions. CPU demand is an indicator of the processing power being utilized, while MEM, or memory demand, demonstrates the consumption of the available memory capacity.

The resource demand can be determined by adding together the minimum (or base) resource demand - $u_i(R_k^{base}(x))$ and the auxiliary resource demand - $u_i(R_k^{aux}(x))$. The auxiliary resource demand accounts for the extra resources needed because of the varying size of input data. Mathematically,

$$u_i(R_k(x)) = u_i(R_k^{base}(x)) + u_i(R_k^{aux}(x)) \tag{7}$$

Additionally, when dividing the service request into slices, it is necessary to determine the resource demand for each individual slice. The resource demand for a specific slice can be calculated by combining base resource demand and dividing the size of that slice by the total size of the service request and then multiplying the result by the auxiliary resource demand. Mathematically,

$$u_i(R_{k,j}(x)) = u_i(R_k^{base}(x)) + u_i(R_k^{aux}(x)) \cdot \frac{u_i(\hat{s}_{k,j})}{u_i(\hat{s}_k)} \tag{8}$$

3.3.3 Service Sensitivity Priority

The sensitivity level of a particular service s_k, is represented by P_k^s, with a value ranging between 0 and 1. The Smart Gateway is responsible for determining the sensitivity of each service. A higher P_k^s value indicates a more sensitive service. For instance, when evaluating services, the processing of sensitive medical data has a higher sensitivity value compared to a service that manages user-created playlists of songs.

In this context, sensitivity may relate to factors such as security, privacy, and the critical nature of the service. Services with a higher sensitivity level may require additional precautions, resource allocation, or prioritization to ensure that they are handled properly and securely within the fog and cloud computing environments. By taking into account the sensitivity of each service, the Smart Gateway can effectively allocate resources and manage the services in a way that addresses the varying requirements and expectations of users, ultimately enhancing overall system performance and user satisfaction.

3.3.4 Service Priority

Service priority pertains to the significance assigned to a service when distributing resources among the service slices. This concept is distinct from service sensitivity priority, as it encompasses factors such as the distance and latency of connected users, as well as the relative service demand. The priority level for a service of type s_k, represented by P_k, can be determined using the following approach:

$$P_k = 0.5 \cdot \sum_{u_i \in U} [P_i \cdot u_i(\hat{s}_k)] + 0.5 \cdot \frac{SD(s_k)}{\sum_{s_j \in S} SD(s_j)} \tag{9}$$

The initial component of the calculation involves summing the priority of all users availing the service s_k, along with the size of the service requests. This user priority takes into account both the user's distance from the G and their communication latency. By considering these factors, the system can better address each user's specific needs and expectations. The second component of the service priority calculation represents the relative service demand. This is determined by comparing the demand of the current service request to the total service demand received by the G. By incorporating the relative service demand into the service priority calculation, the system can better allocate resources and manage services in a manner that is responsive to varying levels of demand, ultimately enhancing overall system performance and user satisfaction. Lastly, multiplying by 0.5 both sides of equation means that importance factors of both sides are equal.

3.4 DRL-based Service Placement

We discussed the challenge of effectively distributing services across fog and cloud computing environments in the previous section. Moving forward, this part will concentrate primarily on the suggested strategy, which involves adapting the DRL technique to include more practical parameters that cater to real-world situations and how this technique can outperform other numerous strategies, including heuristic methods, RL techniques.

The heuristic approach involves using problem-solving methods based on experience, intuition, and trial-and-error to find solutions or make decisions in complex situations. Heuristics can help in generating quick, approximate solutions in scenarios where obtaining an optimal solution is computationally expensive or time-consuming. In the context of service distribution across fog and cloud computing environments, heuristic methods can be used to allocate resources, manage workload, and balance network traffic. These approaches often rely on predefined rules, historical data, or expert knowledge to guide the decision-making process. However, one of the major drawbacks of heuristic methods is their inability to adapt to dynamic changes in the environment, which is common in fog and cloud computing scenarios. Heuristic solutions may not always be optimal, as they can sometimes lead to suboptimal allocations or under utilization of resources.

Additionally, heuristic approaches might struggle with scalability, as the complexity of the problem increases with the growth of the infrastructure. Heuristic algorithms can also suffer from a lack of flexibility, as updating or modifying their underlying rules can be challenging and time-consuming. In contrast to machine learning-based approaches like RL and DRL, heuristic methods may not be able to learn and improve their performance over time, limiting their long-term effectiveness. While heuristic approaches can provide some benefits for service distribution in fog and cloud computing environments, their limitations, such as adaptability, scalability, and flexibility, can lead to suboptimal performance compared to more advanced techniques like RL and DRL.

Reinforcement Learning (RL) is a sub field of machine learning that focuses on training agents to make decisions by interacting with an environment, with the aim of maximizing a cumulative reward signal. An RL agent learns through a trial-and-error process by continuously exploring the environment, taking actions, and receiving feedback in the form of rewards or penalties. The central idea behind RL is to find an optimal policy, which is a mapping from states to actions that maximizes the expected cumulative reward over time. In RL, the agent's learning process is typically modeled as a Markov Decision Process (MDP), which comprises a set of states, actions, transition probabilities, and reward functions. The Qfunction also known as the state-action value function, is a key concept in RL. It estimates the expected cumulative reward an agent will receive after taking a specific action in a given state and following an optimal policy thereafter.

RL methods can adapt to dynamic environments and learn from experiences, making them suitable for various applications, including distributing services across fog and cloud computing environments. Q-learning, a popular RL algorithm, works by iteratively updating the Q-function values based on the agent's experiences, eventually converging to the optimal Q-function. The Q-function can be represented as a table (in tabular Q-learning). However, one of the primary challenges of RL methods is, as the state space and action space expand, the maintenance of the large tabular data becomes a significant limitation. One of the main issues with RL methods is that managing large amounts of tabular data becomes harder as the state and action spaces grow. To overcome this, we can combine RL with other machine learning methods, like deep learning, to make better and more powerful algorithms such as DRL.

Instead of using a large matrix to store state-action pairs like Q-learning, deep learning is used to identify features through Artificial Neural Networks. This approach is used in many areas like image and video analysis and medical image prognosis. It's also applied in reinforcement learning by combining it with deep learning. This method helps the agent predict the best action to take by using a deep learning model with the current state as input rather than a specific state-action pair. Unlike Q-learning, which uses a Q-function to decide the agent's action, deep learning with reinforcement learning predicts an action based on all possible action prediction values.

The DRL-based service placement model is depicted in Figure 3, providing a visual representation of the architecture and components involved in the placement of services using deep reinforcement learning technique.

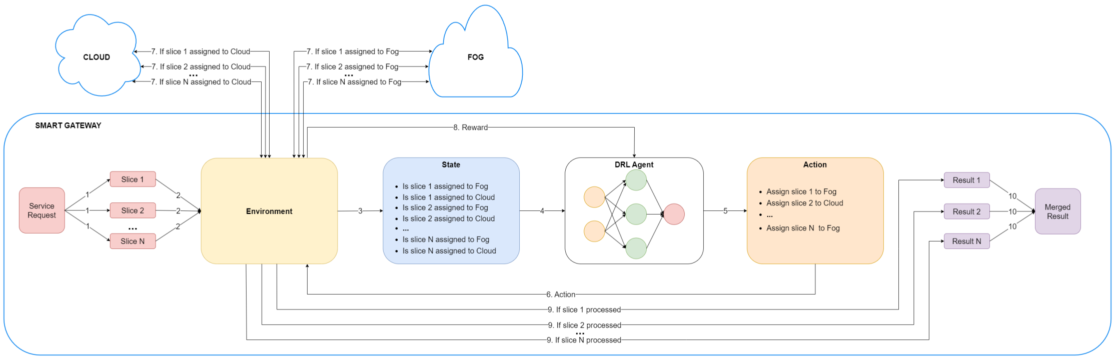

CLOUD
7. If slice 1 assigned to Cloud
7. If slice 2 assigned to Cloud
...
7. If slice N assigned to Cloud
7. If slice 1 assigned to Fog
7. If slice 2 assigned to Fog
...
7. If slice N assigned to Fog
FOG
SMART GATEWAY
8. Reward
Service Request
1
Slice 1
Slice 2
...
Slice N
2
Environment
3
State
Is slice 1 assigned to Fog
Is slice 1 assigned to Cloud
Is slice 2 assigned to Fog
Is slice 2 assigned to Cloud
...
Is slice N assigned to Fog
Is slice N assigned to Cloud
4
DRL Agent
5
Action
Assign slice 1 to Fog
Assign slice 2 to Cloud
...
Assign slice N to Fog
Result 1
Result 2
Result N
10
Merged Result
6. Action
9. If slice 1 processed
9. If slice 2 processed
...
9. If slice N processed

Figure 3. DRL-based service placement model

In our study, we created a sequential model with following parameters and configurations for DRL-based service placement problem. The input layer consisted of nodes corresponding to the number of states. For the dense layer, we utilized twice the number of nodes that present in the input layer. In this layer, the sigmoid activation function was employed, ensuring that the output always falls within the range of (0, 1). The output layer matched the number of nodes in the input layer. In this layer, a linear activation function was utilized, enabling the input values to remain the same. To optimize the model, we employed the Adam optimizer. This optimizer takes advantage of both the adaptive gradient algorithm and the root mean square propagation algorithm. The loss function utilized was the Mean Squared Error (MSE), which aids in minimizing prediction errors. The mathematical model employed for this problem was the Markov Decision Problem (MDP). MDP is represented by a four-tuple, including the state space, action space, state transition probability function, and reward function. As for the framework, we utilized PyTorch. PyTorch stands out due to its dynamic computational graph construction, efficient GPU utilization through optimized tensor operations, and its flexibility and performance in deep learning tasks.

3.4.1 State Representation

The state space includes every possible arrangement of the states of all SRs. Let $\hat{S} = \{\hat{s}_1, \hat{s}_2, \hat{s}_3, ...\}$, be the state space. A state of SR's type s_k signifies the environment in which it operates. Mathematically,

$$\hat{s}_i^k =< f(\hat{s}_i^k), c(\hat{s}_i^k) > \tag{10}$$

$$f(\hat{s}_i^k) + c(\hat{s}_i^k) = 1 \tag{11}$$

Here, $\hat{s}_i^k$ denotes the state of a SR s_k from user u_i. $f(\hat{s}_i^k)$ and $c(\hat{s}_i^k)$ are two boolean variables, indicating whether the SR is allocated to the fog or cloud environment, respectively. The value 1 represents the assignment of the SR to a specific environment, while 0 represents the opposite. Equation 11 must be satisfied for a valid service request assignment. For example, $\hat{s}_3^1 =< 1, 0 >$ implies that SR s_1 from u_3 is sent to the fog computing environment, which is this SR's state. Let $\hat{s}_{i,j}^k$ be the state of slice j of SR s_k from user u_i. Each SR slice's state indicates its assignment. Mathematically,

$$\hat{s}_{i,j}^k = \begin{cases} < 0, 0 > & \text{- the slice is not yet assigned to the neither fog nor cloud.} \\ < 1, 0 > & \text{- the slice is assigned to be processed in the fog environment.} \\ < 0, 1 > & \text{- the slice is assigned to be processed in the cloud environment.} \\ < 1, 1 > & \text{- invalid state} \end{cases} \tag{12}$$

By merging the states of all slices, we can determine the state of a SR $\hat{s}_{i,j}^k$ using the following method:

$$\hat{s}_{i,j}^k =< \hat{s}_{i,1}^k, \hat{s}_{i,2}^k, ..., \hat{s}_{i,\hat{s}}^k > \tag{13}$$

Additionally, the entire SR's state can be expressed as:

$$\hat{s}_i^k =< \hat{s}_1^k, \hat{s}_2^k, ..., \hat{s}_n^k > \tag{14}$$

Finally, the final state is added to the current state space. The final state represents that all the slices of every service request are allocated to one of the computing (fog or cloud) environment.

3.4.2 Action Selection

The action space encompasses all potential actions the agent can take, given its present state. Denoted by $\hat{A}$, the action space characterizes the range of possible decisions available to the agent. By considering all possible actions within the action space, the agent can make informed decisions that optimize its objectives while assigning the slices of services to the fog and cloud computing environments.

Lastly, the agent can not move from final state to other states.

3.4.3 Reward Function

When examining Figure 3, it is clear that by executing a specific action within the current state, the agent receives a reward value from the environment. This reward value serves as an indicator of how successful the action was in given state. In our study, a key objective for the agent is to effectively distribute the service requests between the fog and cloud environments. Mathematically, reward function for each slice can be computed as following equation:

$$f(s_{i,j}^k) + c(s_{i,j}^k) = 1 \tag{15}$$

$$\hat{R}(s_{i,j}^k) = f(s_{i,j}^k) \cdot \hat{R}^f(s_{i,j}^k) + c(s_{i,j}^k) \cdot \hat{R}^c(s_{i,j}^k) \tag{16}$$

$$\hat{R}(s_{i,j}^k) = \begin{cases} -\hat{R}(s_{i,j}^k), & \text{penalty if } T_{exec}(s_{i,j}^k) > T_{deadline}(s_{i,j}^k) \\ \hat{R}(s_{i,j}^k), & \text{otherwise} \end{cases} \tag{17}$$

where, $f(s_{i,j}^k)$ and $c(s_{i,j}^k)$ represent slice is assigned on fog and cloud, respectively. Additionally, $\hat{R}^f(s_{i,j}^k)$ and $\hat{R}^c(s_{i,j}^k)$ are reward values when slice is assigned to fog and cloud, respectively. $T_{exec}(s_{i,j}^k)$ represents the duration it takes to execute the assigned slice, while $T_{deadline}(s_{i,j}^k)$ represents the specified time limit for completing the assigned slice. In our scenario if $T_{exec}(s_{i,j}^k)$ exceeds $T_{deadline}(s_{i,j}^k)$, the agent receives a negative penalty. $T_{deadline}(s_{i,j}^k)$ is determined by Smart Gateway. For example, the deadline for processing a video file may be calculated as 1 second per frame, while for processing an audio file, the deadline could be established based on the duration of the audio file itself. Mathematically, these can be calculated using following equations:

$$\hat{R}^f(s_{i,j}^k) = w1 \cdot P_i + w2 \cdot P_k^s + w3 \cdot P_k + w4 \cdot L_{comm}^f(s_{i,j}^k) + w5 \cdot L_{comp}^f(s_{i,j}^k) \tag{18}$$

$$\hat{R}^c(s_{i,j}^k) = w1 \cdot P_i + w2 \cdot P_k^s + w3 \cdot P_k + w4 \cdot L_{comm}^c(s_{i,j}^k) + w5 \cdot L_{comp}^c(s_{i,j}^k) \tag{19}$$

where, $w1$, $w2$, $w3$, $w4$ and $w5$ are constant parameters that used to adjust the significance of each factor relative to others. Here, the value of those constants are satisfied the constraint $w1 + w2 + w3 + w4 + w5 = 1$. P_i represents priority of user u_i, P_k^s represents sensitivity value of service s_k, and P_k represents priority of service s_k. $L_{comm}^f(s_{i,j}^k)$ and $L_{comp}^f(s_{i,j}^k)$ represent communication latency on fog and computation latency on fog, respectively. Similarly, $L_{comm}^c(s_{i,j}^k)$ and $L_{comp}^c(s_{i,j}^k)$ represent communication latency on cloud and computation latency on cloud, respectively. The communication latency for fog, and for cloud can be calculated using following equations:

$$L_{comm}^f(s_{i,j}^k) = \frac{l_i}{l_f} \tag{20}$$

$$L_{comm}^c(s_{i,j}^k) = \frac{l_i}{l_c} \tag{21}$$

where, l_i represents the Round Trip Delay (RTD) time from user u_i to Smart Gateway, and l_f represents RTD from Smart Gateway to fog node. Similarly, l_c represents RTD from Smart Gateway to cloud environment. The computation latency for fog, and for cloud can be calculated using following equations:

$$L^f_{comp}(s^k_{i,j}) = \frac{u_i(\hat{s}_{k},_{j}\,)}{u_i(\hat{s}_k)} + \frac{u_i(R_{k,j}(x))}{R^f(x)} \tag{22}$$

$$L^c_{comp}(s^k_{i,j}) = \frac{u_i(\hat{s}_{k},_{j}\,)}{u_i(\hat{s}_k)} + \frac{u_i(R_{k,j}(x))}{R^c(x)} \tag{23}$$

where, $u_i(\hat{s}_{k},_{j}\,)$ represents slice size of service s_k, $u_i(\hat{s}_k)$ represents input size of service s_k, $u_i(R_{k,j}(x))$ represents resource demand for slice j, $R^f(x)$ and $R^c(x)$ represents available resources on fog and on cloud, respectively. To determine the reward of a service request (SR), the sum of rewards from all the slices can be computed. Mathematically,

$$\hat{R}(s^k_i) = \sum_{s_j \in S} f(s^k_{i,j}) \cdot \hat{R}^f(s^k_{i,j}) + c(s^k_{i,j}) \cdot \hat{R}^c(s^k_{i,j}) \tag{24}$$

The goal of our agent is to divide service requests and distribute the slices efficiently between the fog and cloud environments without compromising the QoS. When assigning a SR slice on an environment, the following constraints have to be considered:

$$d_i < D, \forall u_i \in U \tag{25}$$

$$1 \leq \sum_{0<k\leq m} u_i(s_k) \leq m, \forall u_i \in U \tag{26}$$

$$\sum_{0<j<\hat{s}^k_i} u_i(\hat{s}_{k},_{j}\,) = u_i(\hat{s}_k), \forall u_i \in U \tag{27}$$

$$\sum_{0<j<\hat{s}^k_i} u_i(R_{k,j}(x)) = u_i(R_k(x)), \forall u_i \in U \tag{28}$$

$$\sum_{0<j<\hat{s}^k_i} f(s^k_{i,j}) \cdot u_i(R_{k,j}(x)) \leq R^f(x), \forall u_i \in U \tag{29}$$

$$\sum_{0<j<\hat{s}^k_i} c(s^k_{i,j}) \cdot u_i(R_{k,j}(x)) \leq R^c(x), \forall u_i \in U \tag{30}$$

The constraints can be summarized as below:

- Equation 25 ensures that the user needs to be within the maximum communication range of the fog.
- Equation 26 ensures that number of requested services should be less than the total number of services offered by environments.
- Equation 27 ensures that total size of SR must be equal to sum of size all of SR slices.
- Equation 28 ensures that total resource requirement of SR must be equal to sum of resource requirement of SR slices.
- Equation 29 ensures that sum of all resource requirement of SR slices that assigned to fog must be less or equal than available resource on Fog.
- Equation 30 ensures that sum of all resource requirement of SR slices that assigned to cloud must be less or equal than available resource on Cloud.

Table 2, shows the list of notations that are used in our study.

Table 2. List of Notation

Notation	**Description**
G	Smart Gateway
F	Fog environment
C	Cloud environment
l_f	Round Trip Delay (RTD) time from Smart Gateway to Fog
l_c	Round Trip Delay (RTD) time from Smart Gateway to Cloud
$R^f(x)$	Available resources on Fog
$R^c(x)$	Available resources on Cloud
U	Set of users
n	Total number of users
u_i	User i
$u_i(s_k)$	Boolean variable, represents if user u_i is currently availing service s_k
d_i	Distance between user u_i and Smart Gateway
P_i^d	Distance priority of user u_i
l_i	Round Trip Delay (RTD) time from user u_i to Smart Gateway
P_i^l	Latency priority of user u_i
P_i	Priority of user u_i
S	Set of services that offered by environments
m	Total number of services
s_k	Service type k
$\hat{s}_i^k$	Number of slices of Service Request type s_k
$u_i(\hat{s}_k)$	Size of SR of type s_k from user u_i
$u_i(\hat{s}_k,_j)$	Size of slice of SR of type s_k from user u_i
$SD(s_k)$	Demand of service type s_k
$u_i(R_k(x))$	Resource demand of service type s_k from user u_i
P_k^s	Sensitivity value of service type s_k
P_k	Priority of service type s_k
$\hat{S}$	State space
$\hat{A}$	Action space
$\hat{R}(s_i^k)$	Sum of rewards from all the slices
$L_{comm}^f(s_{i,j}^k)$	Communication latency reward on Fog
$L_{comm}^c(s_{i,j}^k)$	Communication latency reward on Cloud
$L_{comp}^f(s_{i,j}^k)$	Computation latency reward on Fog
$L_{comp}^c(s_{i,j}^k)$	Computation latency reward on Cloud
$f(s_{i,j}^k)$	Represents that slice is assigned on Fog
$c(s_{i,j}^k)$	Represents that slice is assigned on Cloud

3.5 Summary

In this chapter, we began by presenting the system architecture. We then proceeded to introduce user modeling and service modeling. Following that, we presented the formulation of the DRL-based service placement that we used in our study. We extensively discussed the reasons for opting for a DRL-based solution and provided a thorough explanation of how it operates. The chapter concluded with a detailed discussion on the state space, action space, and reward function. In the upcoming chapter, we will delve into the execution and methodology of the experiment.

4 Experiment

This section describes implementation of DRL model and experimentation in detail. The experimental settings is first described in Section 4.1, including environment setup, hyper-parameter configurations, and performance metrics. Further, we discuss experiment applications in Section 4.2, Deep learning based object classification application in Subsection 4.2.1, Text-audio synchronisation or forced alignment application in Subsection 4.2.2, and Speech-to-text conversion in Section 4.2.3, with all the details. In Section 4.3, we discuss various performance metrics to evaulate our DRL algorithm. Lastly, we conclude this chapter by providing a summary in Section 4.4.

4.1 Experimental Settings

4.1.1 Environment setup

In our system, we have three distinct computing environments: Smart Gateway, Fog, and Cloud environments. Each of these environments have their individual capabilities, all of these environments are subject to various constraints including CPU limits, RAM limits and etc. By imposing CPU limits, we ensure that the processing capabilities of the Smart Gateway, Fog, and Cloud environments are effectively managed, preventing any single component from monopolizing system resources. This promotes fair resource allocation and prevents performance degradation across the entire system. Similarly, RAM limits play a vital role in maintaining stability and preventing resource exhaustion. By defining appropriate boundaries for memory usage within each environment, we ensure that the system operates smoothly, avoiding potential bottlenecks and maintaining a consistent level of performance.

We employed Raspberry PI 4 device as a Smart Gateway in Edge layer. In Fog layer, we have 5 Raspberry PI 4 devices that installed Docker Swarm. One of the devices used as swarm manager, other nodes used as worker nodes. Also, we have a cloud server that represents cloud node in cloud layer. Figure 4 illustrates implementation of Cloud-Fog-Edge architecture.

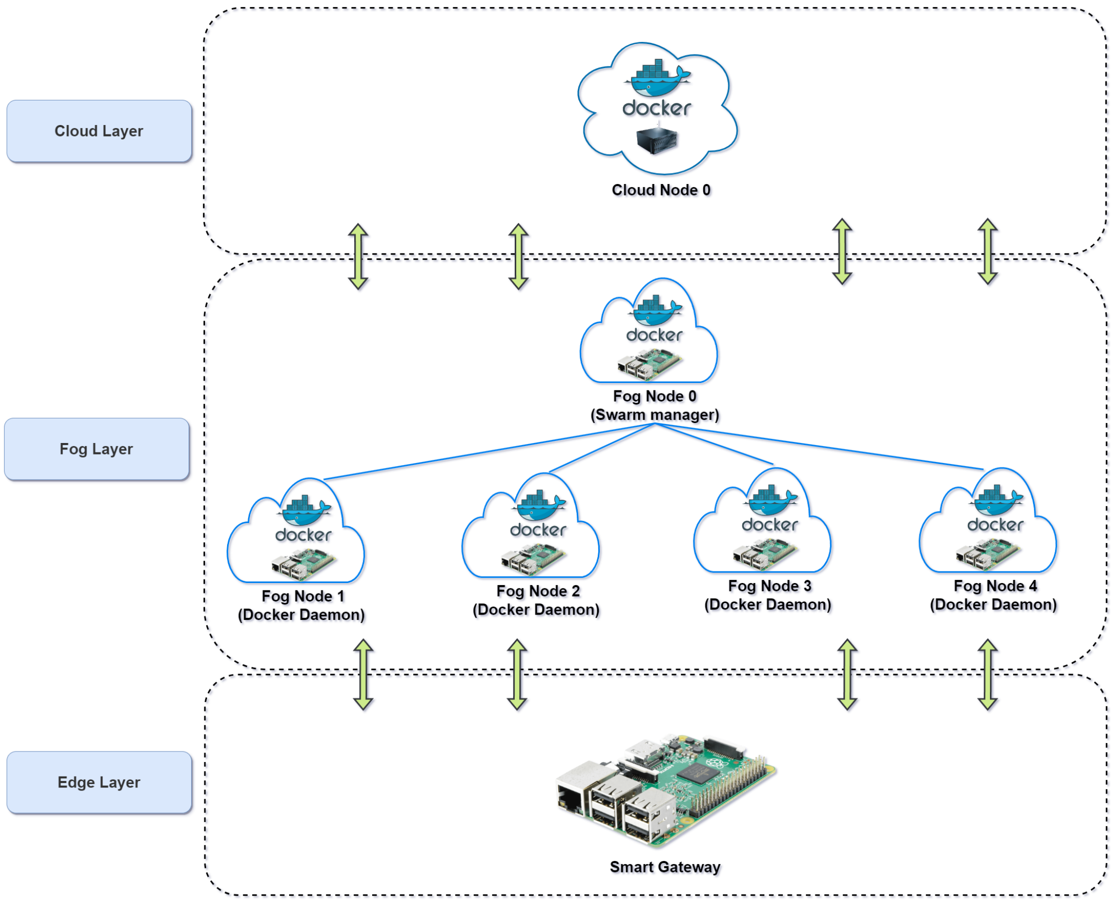


Figure 4. Implementation of Cloud-Fog-Edge architecture.

4.1.2 Hyper-parameter configurations

The hyperparameters for training DRL model are crucial settings that affect the performance and behavior of the model during training. We consider several hyperparameters for training DRL-based service placement model such as discount factor, mini-batch size, replay buffer size, epsilon max, epsilon min, epsilon decay, learning rate, and etc. The discount factor determines the importance of future rewards compared to immediate rewards. In our case, the discount factor is set to 0.99, implying that future rewards are valued highly. The mini-batch size parameter specifies the number of samples used in each iteration of the model training. A mini-batch size of 64 is chosen, indicating that 64 samples are processed together before updating the model. The replay buffer stores past experiences, allowing the model to learn from a mixture of recent and older data. The replay buffer size is set to 5000, indicating that the buffer can hold up to 5000 experiences. Epsilon is the exploration factor that determines the trade-off between exploration and exploitation during training. Epsilon starts at a maximum value of 1 and decays gradually until it reaches a minimum value of 0.01. This ensures a balance between exploration and exploitation over time. The epsilon decay determines the rate at which epsilon decays. In our case, the decay factor is set to 0.99, indicating a slow and gradual decay.

The learning rate determines the step size for adjusting the model's parameters during training. A learning rate of 0.001 is chosen, indicating small and gradual updates. The parameters (no. of nodes in input layer, no. of nodes in hidden layers, and no. of nodes in output layer) define the architecture of the neural network used in the DRL model. The number of layers is determined based on the number of states. The optimizer specifies the algorithm used to update the model's parameters based on the computed gradients. Here, the Adam optimizer is employed, which is a popular choice for deep learning models. The loss function determines the measure of dissimilarity between the predicted and target values during training. Mean Squared Error (MSE) is utilized as the loss function in our case.

We utilized Python v3.10 as a programming language, and PyTorch v2.0 as a framework for implementation of the DRL algorithm. Table 3 provides a summary of the hyperparameters configurations for the training DRL model.

Table 3. Hyper-parameters for DRL model training

Parameter	Value
General	
Discount factor (γ)	0.99
Mini-batch size	64
Replay buffer size	5000
Epsilon max (ϵ_{max})	1
Epsilon min (ϵ_{min})	0.01
Epsilon decay factor	0.99
Neural network parameters	
Learning rate (α)	0.001
No. of nodes in input layer	No. of states
No. of nodes in hidden layer	2 * No. of states
No. of nodes in output layer	No. of states
Optimizer	Adam
Loss function	MSE

4.2 Experiment Applications

The original versions of applications were modified to satisfy the problem formulation, and the revised editions used in this section are available in the project repository. We utilized Python v3.10 as a programming language and Flask v2.3.1 as a web framework for the development of the all applications. This section explains the services available in the fog and cloud environment.

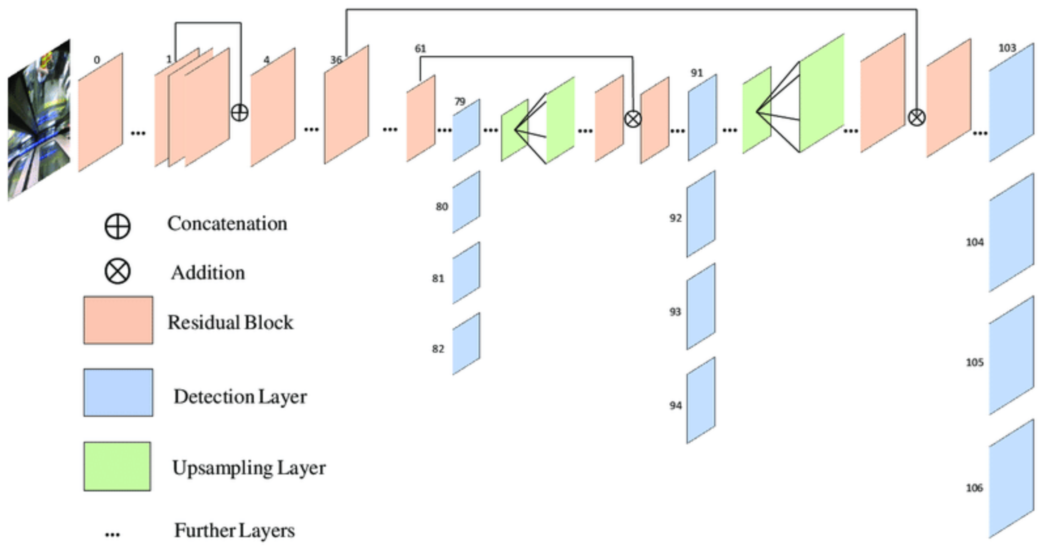


Figure 5. Network architecture of YOLOv3 [1].

4.2.1 Application 1: Deep learning based object classification

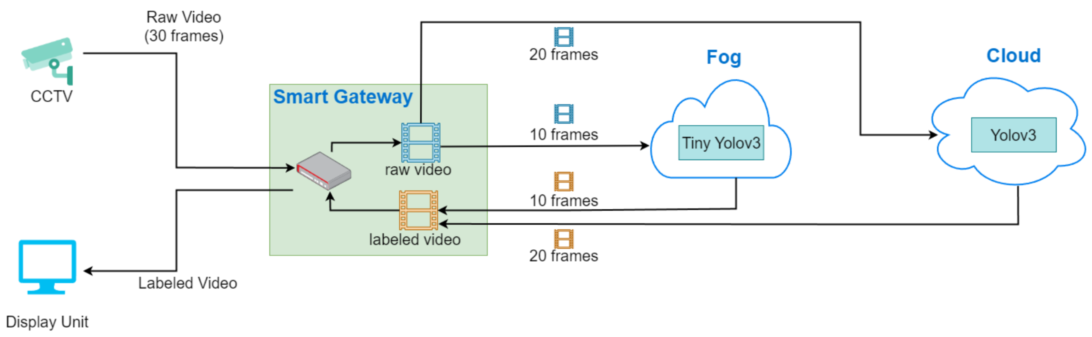


Figure 6. Architecture of Application 1: Deep learning based object classification

YOLOv3 (You Only Look Once) is real-time object detection algorithm that uses a single convolutional neural network (CNN) to detect objects in an image. It works by dividing an input image into a grid of cells, and predicting bounding boxes and class probabilities for each cell. Each bounding box consists of four coordinates (x, y, width, height) and a confidence score, which represents the likelihood that the box contains an object. Class probabilities are predicted for each box, indicating the likelihood that the object in the box belongs to a certain class. To improve the accuracy of object detection, YOLOv3 uses a few key techniques. One is feature pyramid networks, which use a pyramid of features extracted from multiple layers of the network to improve the detection of objects at different scales as shown in Figure 5. Another technique is residual connections, which allow gradients to flow more easily through the network during training, leading to better accuracy.

This application is a suitable choice for Fog environment, it can be deployed to detect objects in order to reduce communication delays in fog computing. The Algorithm 1 represents how Yolov3 used in this application. Additionally, The Figure 6 illustrates an architecture of deep learning based object classification application.

Algorithm 1: Deep learning based object classification

```
Input: Video to be analyzed
Result: Detected objects in the video
1 Load the pre-trained YOLOv3 object detection model;
2 foreach frame in video do
3     Extract image from frame;
4     Perform object detection using YOLOv3;
5     Draw bounding boxes around detected objects;
6     Add labels to bounding boxes;
7     Insert the annotated image back into the frame;
8     Add the frame to the output video;
```

Additionally, we have developed a flowchart illustrating the experimental setup for streaming video in Application 1. We considered slices as a set of frames for this experiment. In the flowchart the system initially verifies if the input is streaming before the procedure starts. The system then reads the frame that was acquired by the cameras after receiving the input from the CCTV camera. The system stores the captured frame for further processing after reading the frame. The system then determines if X seconds have passed since the last frame processing. In the event that the necessary amount of time has passed, the system loops back to reading frames and loading the DRL model for distributing the gathered frames between fog and cloud. If the required time has not passed, the system checks the current frame is last frame or not. The system loops back to read frame if it is found that the current frame is not the final one in the streaming input. The system starts an agent to distribute the gathered frames between fog and cloud if it is the last frame. After successfully assigning collected frames, the system collects various metrics such as throughput, communication times, CPU and RAM usages from the assigned fog and cloud environments. The processing cycle is finished when the system shows all the processed output. In our study, "online mode" refers to to employing a trained model for real-time inference from live streaming data. On the other hand, "offline mode" refers to the training phase of the agent when allocating frames between fog and cloud. Lastly, X equals to 6 seconds in our experiments.

The Figure 7 illustrates an flowchart of Experimental setup for streaming slices in Application 1.

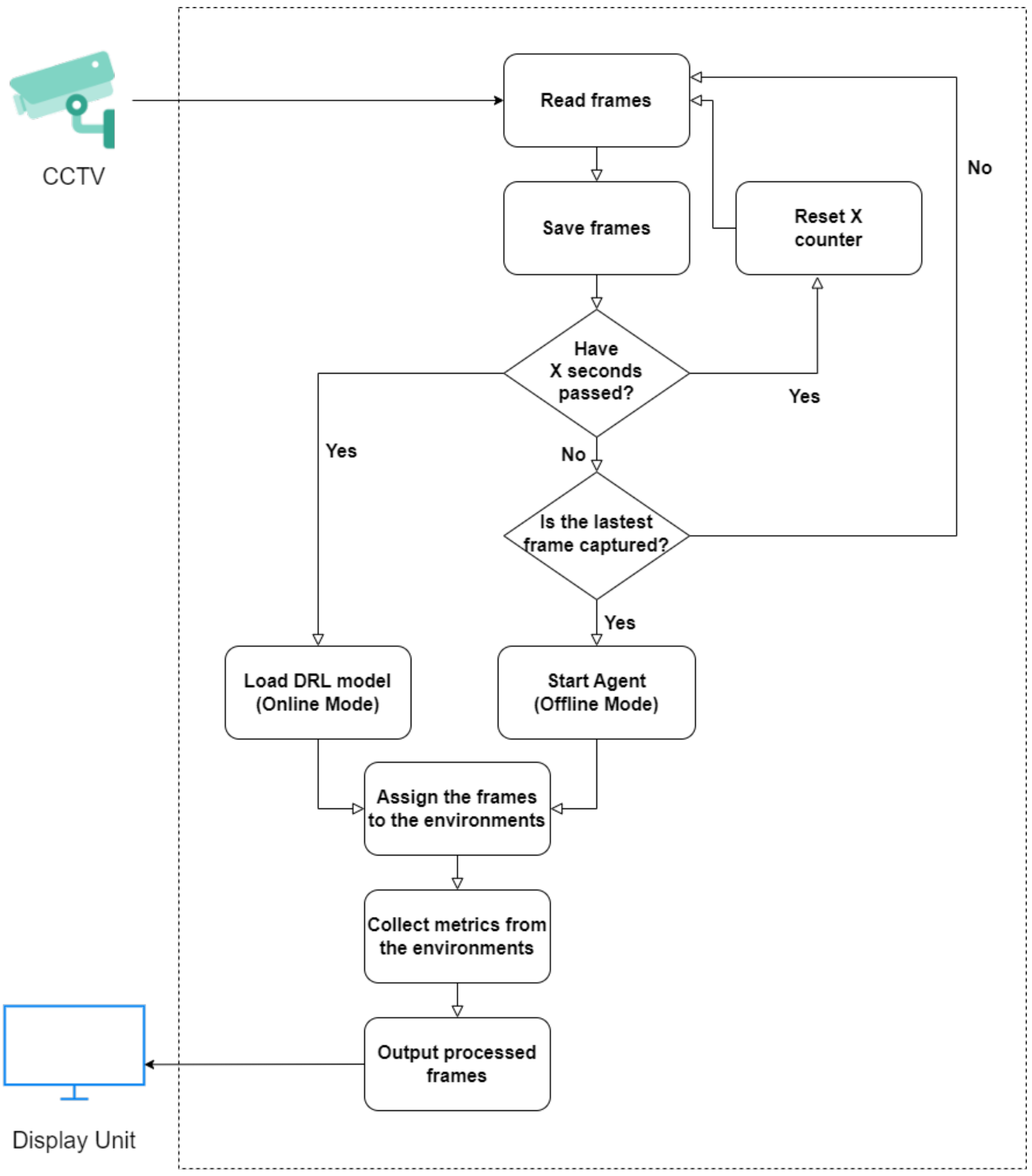

CCTV
Read frames
Save frames
Reset X counter
No
Have X seconds passed?
Yes
Yes
No
Is the lastest frame captured?
Yes
Load DRL model (Online Mode)
Start Agent (Offline Mode)
Assign the frames to the environments
Collect metrics from the environments
Output processed frames
Display Unit

Figure 7. Flowchart of Experimental setup for streaming slices in Application 1

4.2.2 Application 2: Text-audio synchronisation or forced alignment

Aeneas is a text-alignment library that aligns text with audio or video content. As shown in Algorithm 2, the library create sync map, allowing users to jump to the corresponding point in the audio or video when clicking on a specific word or phrase in the text. Aeneas

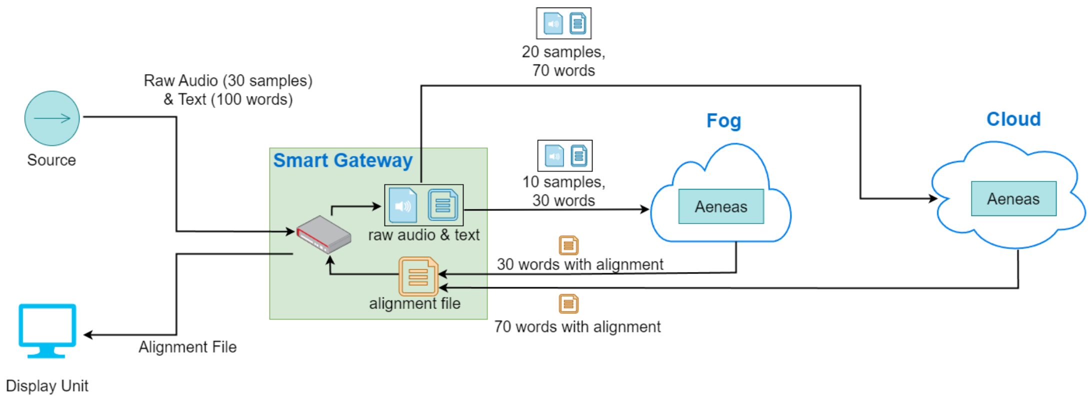


Figure 8. Architecture of Application 2: Text-audio synchronisation or forced alignment

works by first converting the audio or video into a time-aligned phonetic transcript, known as an "alignment graph." It then generates a time-aligned version of the text, known as a "fragmented transcript," based on the words and timing information in the alignment graph. Next, Aeneas performs a process called "forced alignment," which uses statistical models to align the transcript to the audio or video. This process allows for a precise time-based synchronization of the text and media content. The resulting output of Aeneas can be an XML, JSON, SRT files that includes a detailed mapping of the text and media content, including time codes and other metadata. The Figure 8 illustrates an architecture of text-audio synchronisation or forced alignment application.

Algorithm 2: Text-audio synchronisation or forced alignment

```
  Input: Text and Audio Files
  Output: Alignment File (*.srt)
1 Initialize the Aeneas object;
2 Create the sync map;
3 Perform the alignment;
4 Save the alignment from the Aeneas;
5 Output the alignment file;
```

The experimental setup for streaming segments in Application 2 closely resembles that of Application 1, except in this experiment, the slices are treated as collections of segments of audio and text. The source diverges from that of Application 1 as well, in this case, the requirement is to stream content from a source capable of transmitting both audio and text. Furthermore, it is important to note that during audio streaming, the text content is sliced at a rate of two words per second [46].

The Figure 9 illustrates an flowchart of Experimental setup for streaming slices in

Application 2.

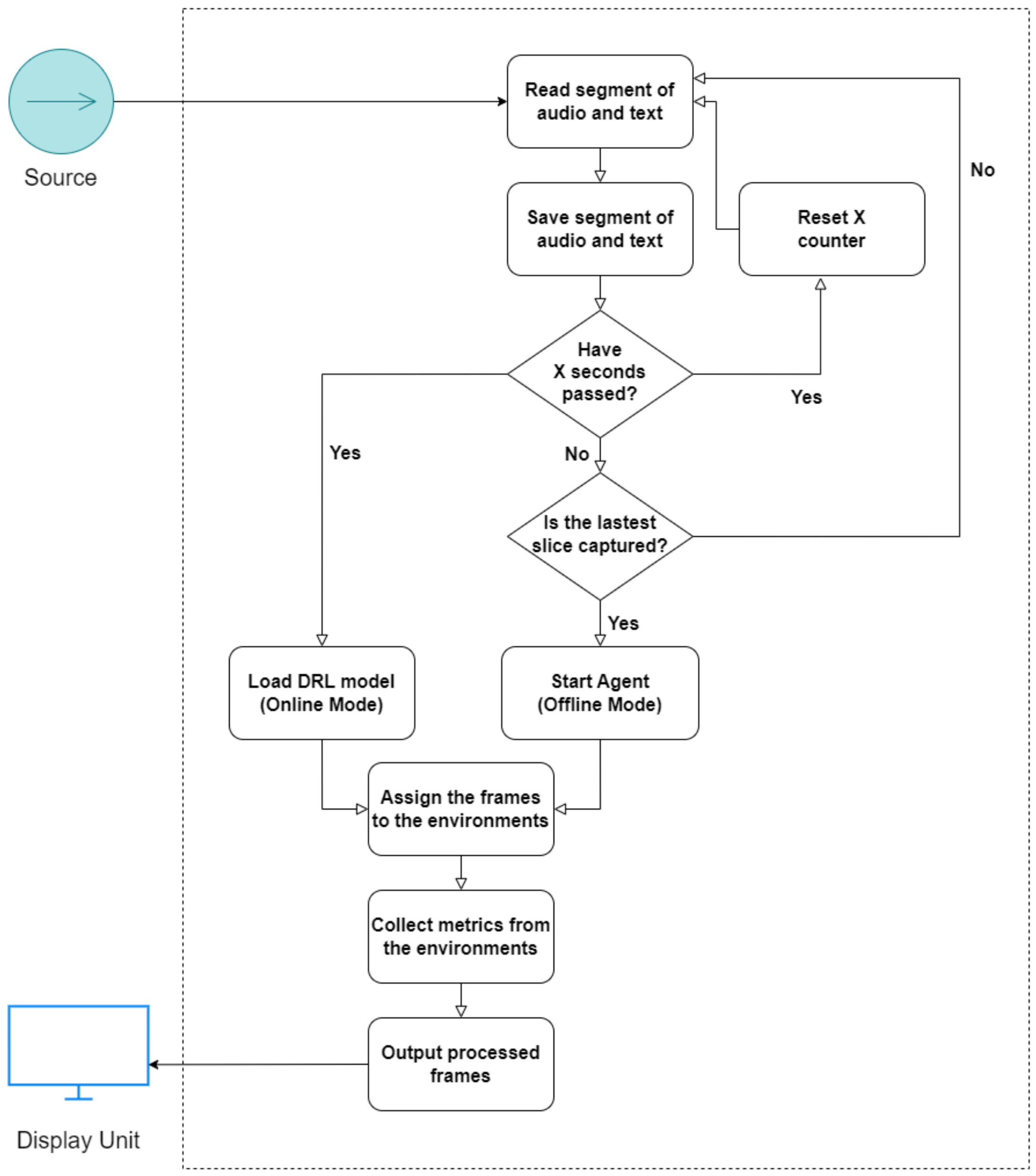

Source
Read segment of audio and text
Save segment of audio and text
Reset X counter
No
Have X seconds passed?
Yes
Yes
No
Is the lastest slice captured?
Yes
Load DRL model (Online Mode)
Start Agent (Offline Mode)
Assign the frames to the environments
Collect metrics from the environments
Output processed frames
Display Unit

Figure 9. Flowchart of Experimental setup for streaming slices in Application 2

4.2.3 Application 3: Speech-to-text conversion

```
Algorithm 3: Speech-to-text conversion
   Input: Audio file to be transcribed
   Result: Transcription of the audio file
 1 Load the audio file;
 2 Initialize the PocketSphinx recognizer;
 3 Set the language model and dictionary;
 4 Process the audio file in chunks;
 5 Feed the chunks to the recognizer;
 6 Get the transcription from the recognizer;
 7 Output the transcription;
```

PocketSphinx is a lightweight speech recognition engine. It is based on the Sphinx-4 open source speech recognition system, which uses Hidden Markov Models (HMMs) to model the acoustic features of speech. PocketSphinx is capable of recognizing speech in a variety of languages, and can be used for tasks such as voice search, dictation, and command recognition. The PocketSphinx library provides a set of Python APIs for developers to use in their applications. As shown in Algorithm 3, to use PocketSphinx, firstly start by configuring a decoder, which specifies the acoustic and language models to be used for speech recognition. The audio input is then passed to the decoder, which performs a series of signal processing and feature extraction operations to produce a stream of speech recognition hypotheses. These hypotheses are then scored and ranked according to their likelihood of being correct, and the most likely hypothesis is returned as the recognized text. The Figure 10 illustrates an architecture of speech-to-text conversion application.

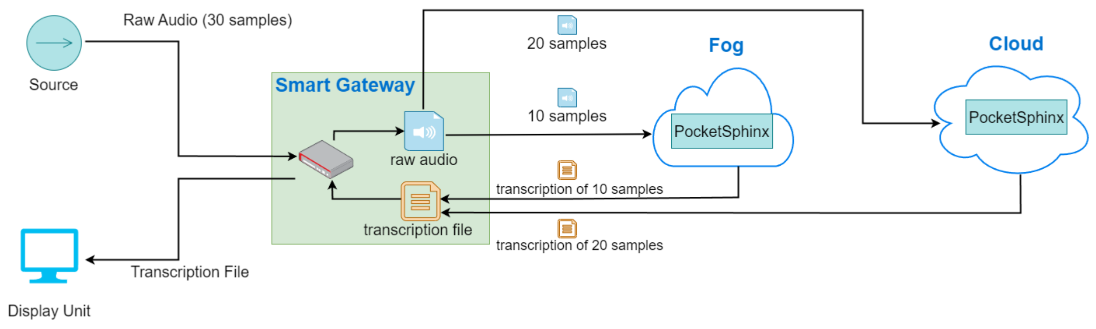


Figure 10. Architecture of Application 3: Speech-to-text conversion

The experiment setup for streaming segments within Application 3 closely mirrors that of Application 1 and Application 2. However, in this particular experiment, the slices are handled as a set of of audio segments. Additionally, the data source differs from that in Application 1 and Application 2, as it must now facilitate the streaming of both audio and text content. The figure 11 represents the flowchart outlining the experimental setup for streaming slices in Application 3.

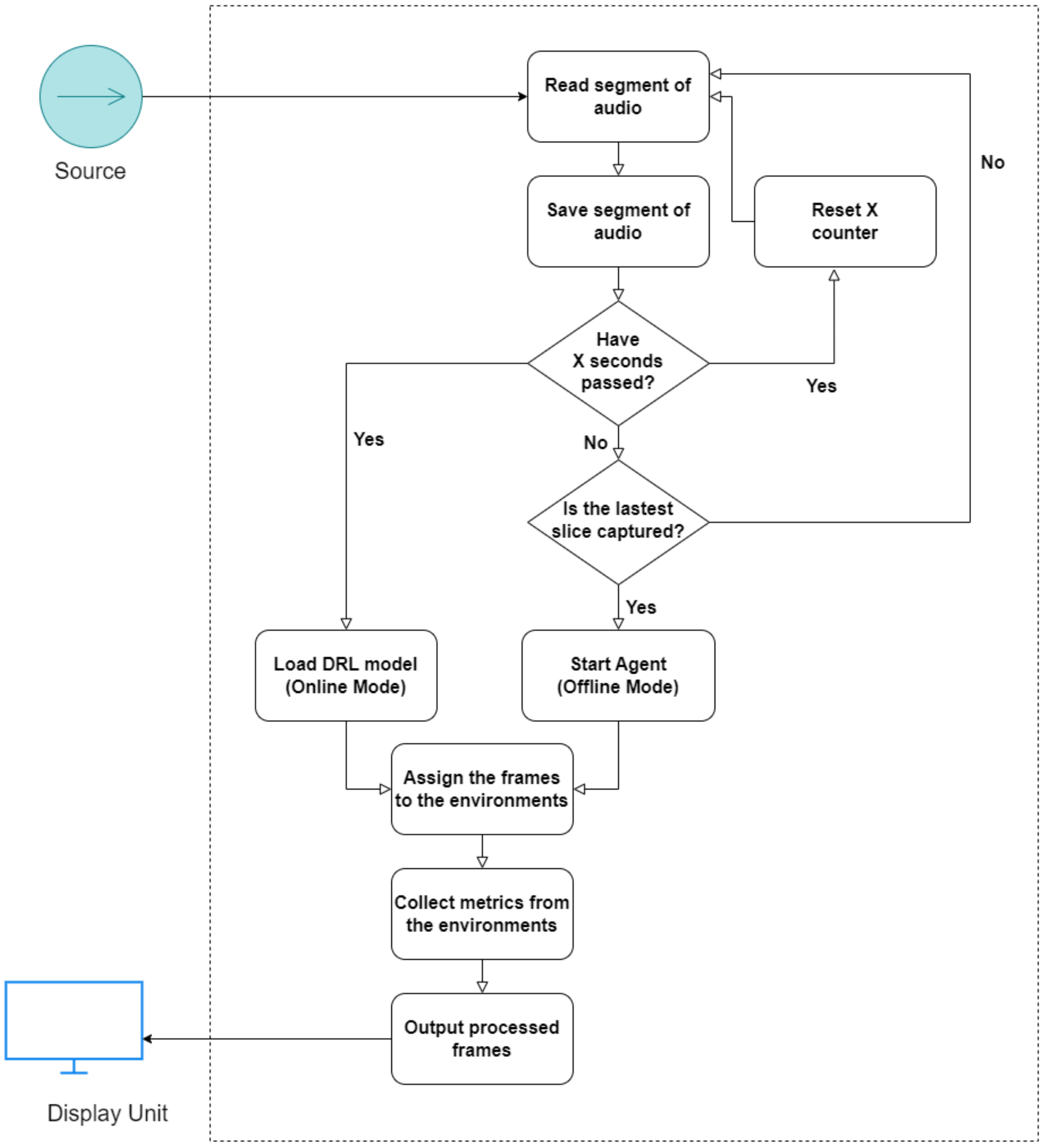

Source
Read segment of audio
Save segment of audio
Reset X counter
No
Have X seconds passed?
Yes
Yes
No
Is the lastest slice captured?
Yes
Load DRL model (Online Mode)
Start Agent (Offline Mode)
Assign the frames to the environments
Collect metrics from the environments
Output processed frames
Display Unit

Figure 11. Flowchart of Experimental setup for streaming slices in Application 3

4.2.4 Application Settings

The service requests vary in size for all applications, spanning from 50MB to 250MB. There are two types of resource requirements for these service requests: CPU and RAM. Application 1 exhibits a pronounced requirement for both CPU and RAM resources, while Application 2 necessitates an intermediate level of these resources. Conversely, Application 3 entails a minimal demand for CPU and RAM capacities. Sensitivity value for services range from 1 to 5. The sensitivity value for each services is assigned randomly in Smart Gateway. Table 4 provides a summary of the configurations for these applications.

Table 4. Application details

Name	CPU	RAM	Input size (MB)		Sensitivity value	
			Min Value	Max Value	Min Value	Max Value
Application 1	High	High	50	250	1	5
Application 2	Medium	Medium	50	250	1	5
Application 3	Low	Low	50	250	1	5

4.3 Performance metrics

We employ the following metrics for assessing the effectiveness of our solution.

Success Rate: The effectiveness of the agent's learning in distributing service request slices is evaluated through the success rate, which represents the proportion of times in which the agent achieves its objective by exploring different states and actions. Factors taken into consideration for assessing the success rate include the quantity of slices per service request and the number of iterations performed.

Distribution of Service Request Slices: Efficiently distributing service requests between fog and cloud environments is a key aim of our algorithm. Evaluating the distribution of service request slices allows us to assess the efficacy of our proposed algorithm in accomplishing this objective.

In-depth analysis encompassing various metrics: Further, we used throughput, communication time, CPU usage, and RAM usage in in-depth analysis of all three data-intensive applications.

4.4 Summary

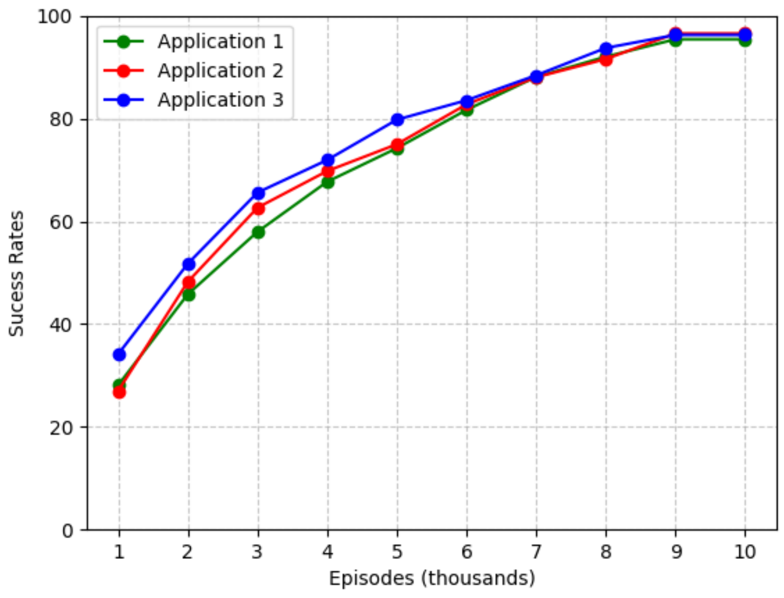
In conclusion of this chapter, we observed the setup of the experiment and its execution. We introduced the coding language, the utilized library, and provided details on the computational capabilities. Additionally, we explored experimental settings, hyperparameters for DRL, and application settings. In the next chapter, we will will delve into the discussion of results.

5 Results and Discussion

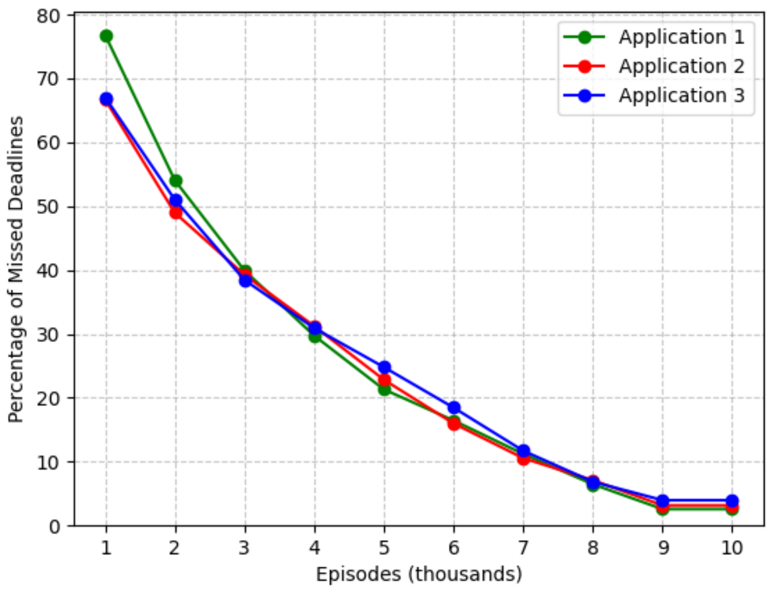
This chapter describes the results from experiments that we conducted for all three data-intensive applications. We describe both insights from training phase, and from in-depth analysis encompassing various metrics.

5.1 Success Rates

Figure 12a illustrates the correlation between the success rate and the number of episodes during the training. The success rate pertains to the overall percentage of successfully assigned slices by agent while exploring the state space. The results are collected through the utilization of varying number of Service Requests (SRs) across all applications and slices, with ranging from 1 to 6. The number of episodes, which ranges from 1000 to 10,000, is shown on the X-axis. The success rate (in %) for Applications 1, 2, and 3 is shown on the Y-axis. As the number of episodes grow, the success rate is increasing among all the applications with Application 3 having the highest value during whole process. On the other hand, Application 1 has a lower rate of success in the initial phase, but as the number of episodes grow the application catches up to the others.



(a) Success Rates for each Applications.



(b) Missed Deadlines for each Applications.

Figure 12. Success Rates and Missed Deadlines for each Application.

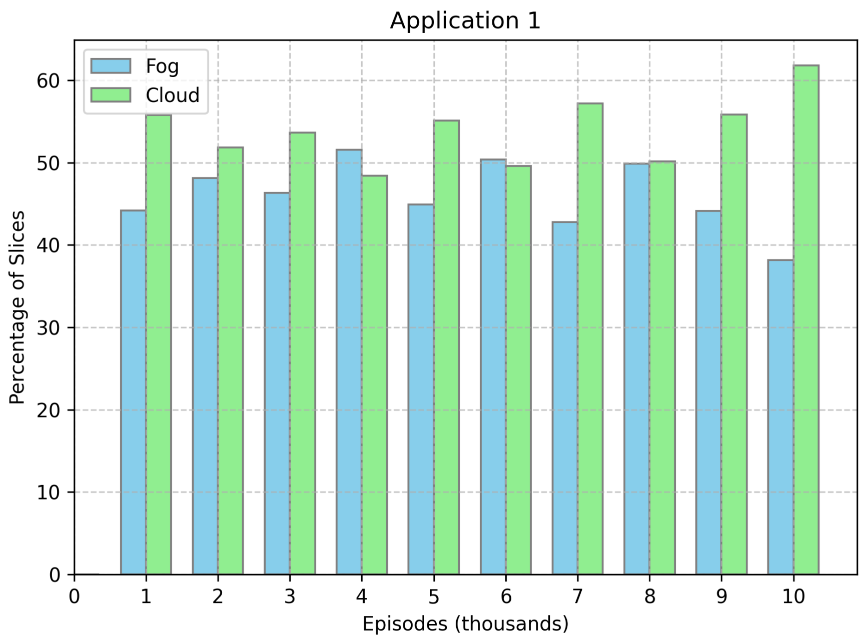
On the contrary, Figure 12b illustrates the correlation between the slices that deadlines are missed while assigning and the number of episodes during the training. The whole graph has a down trend of the percentage of missed deadlines, when the number of episodes increase. During the initial phase in the first thousands, the percentage of missed deadlines between the applications varies, with Application 1 having a higher percentage in comparison with the others. Although in the next episodes as the numbers grow the variation between the applications becomes non-existent. Based on the results that obtained from Figure 12a and Figure 12b we can say that agent has some struggles to grasp the situation for Application 1, but in the long run all the applications have similar results both in number of missed deadlines and their respective success rates.

5.2 Percentage of Slices

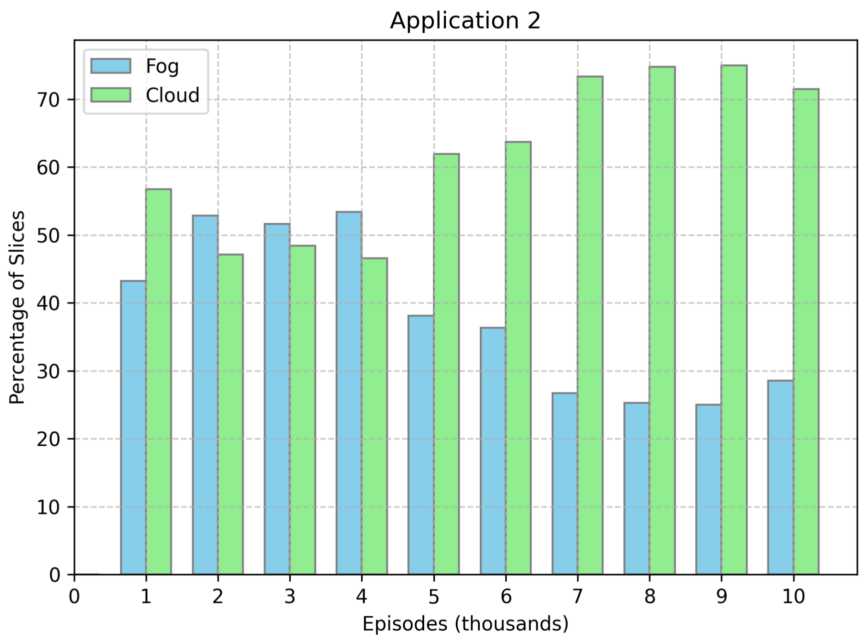
Figure 13, illustrates the distribution of SR slices between fog and cloud environments for each applications. The number of episodes, which ranges from 1000 to 10,000, is shown on the X-axis, and percentage of total number of slices in fog and in cloud is shown on the Y-axis. The SRs and slices are ranging from 1 to 6 for each applications in this experiment. It is clear from Figure 13a that the agent assigned approximately 50% of slices for each environment at 6000th and 8000th episodes. The agent assigned slightly more slices to Fog at 4000th episode. For other episodes, more slices are assigned to Cloud for Application 1.

From Figure 13b, we can say that at 2000th, 3000th and 4000th episodes, the agent tried to assign slightly more slices to Fog. For other episodes, we clearly see that mostly Cloud is chosen by agent for processing slices. From Figure 13c, it is observed that the agent assigned approximately 50% of slices for each environment at 2000th. Slightly more slices are assigned to Fog at 8000th and 9000th episodes. For other episodes, more slices are assigned to Cloud for Application 3.

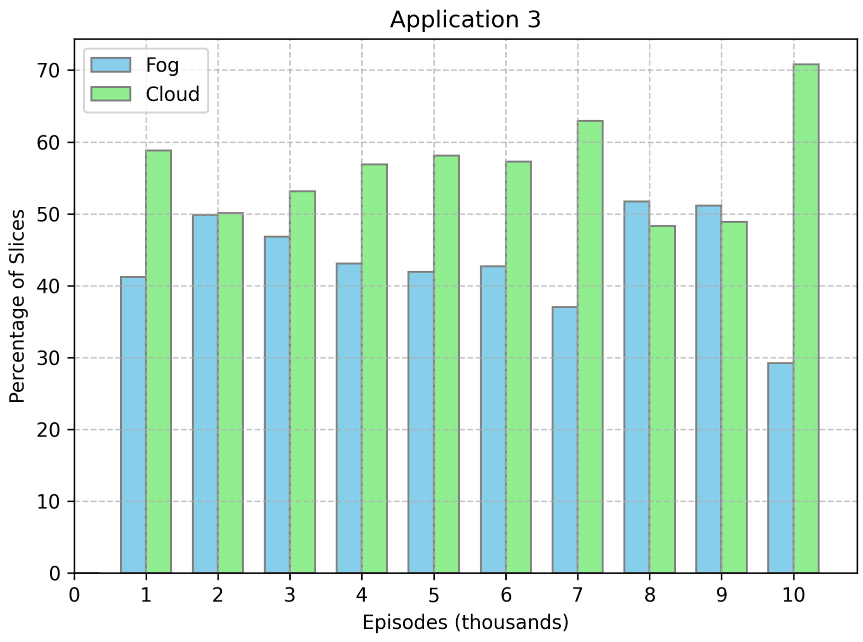
The varying loads of the environments are mostly responsible for the non-uniform variation in the distribution of slices in each run. Nevertheless, it is important to highlight that the agent distributes a greater number of slices to the Cloud across all applications.



(a) Distribution of SR Slices for Application 1.



(b) Distribution of SR Slices for Application 2.



(c) Distribution of SR Slices for Application 3.

Figure 13. Distribution of SR Slices for each Application.

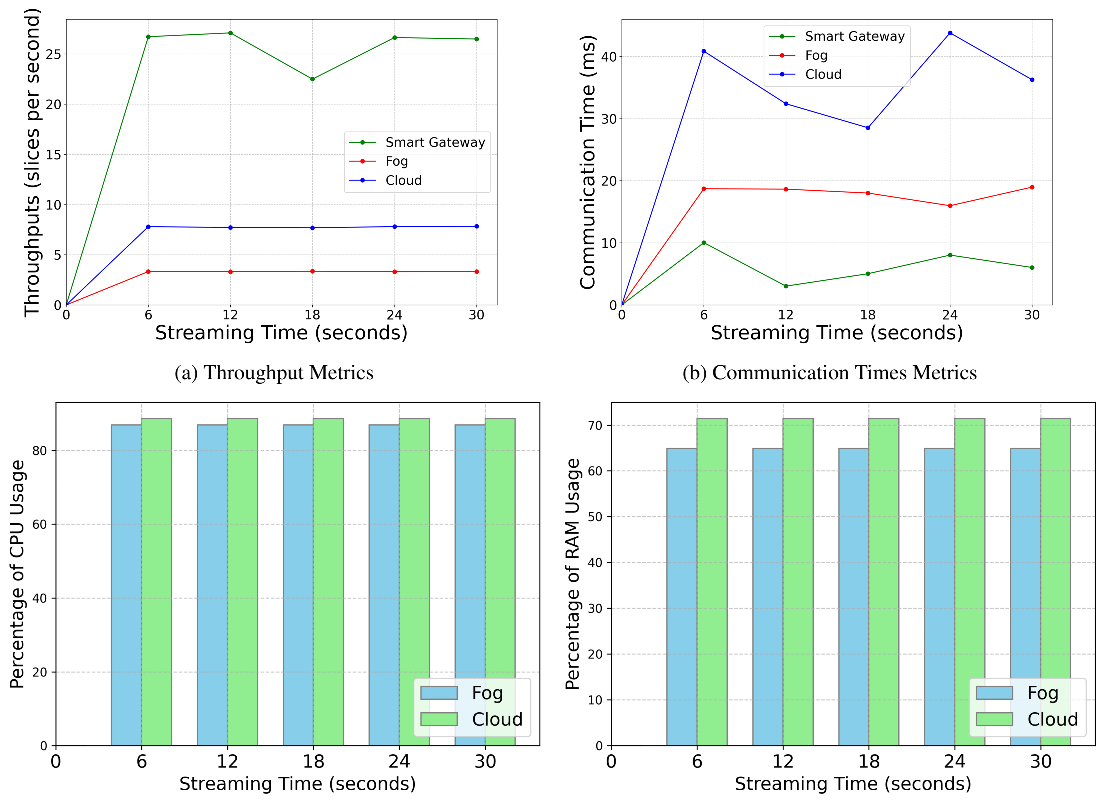
5.3 In-depth analysis encompassing various metrics

5.3.1 Application 1: Deep learning based object classification

The Figure 7 illustrates a comprehensive flowchart detailing the methodology employed in this experiment. The experiment focuses on an in-depth analysis of Application 1 encompassing throughput, communication time, CPU and RAM usage in Smart Gateway, Fog, and Cloud computing environments as shown in The Figure 14. Figure 14a visualizes the throughput metrics associated with each of the environments. Throughput is measured as slices per second meaning that set of frames per second in this experiment. It is clear that the throughput levels of Fog and Cloud remain consistent throughout the streaming duration. This indicates that the number of slices allocated to both Fog and Cloud during the streaming period remain unchanged, and their processing time remains consistent in both settings. However, the throughput of the Smart Gateway experiences a minor fluctuation in the 18th second of streaming. This situation might arise because the Smart Gateway is equipped with numerous threads designed for various functions, concurrently handling tasks such as video processing, slice allocation, and output display.

Figure 14b represents communication times for environments during the streaming time. We observed fluctuating trend on Fog, Cloud and Smart Gateway. This variability might be attributed to the dynamic nature of environments. The values indicate that communication times are minimal in all environments. This could be affect less the deadlines for assignment process of slices. So that enables agent to effectively allocate slices based on its past experiences. Both throughput and communication time results for Fog and Cloud suggests that resource utilization should remain consistent and steady.

In Figure 14c, we are presented with CPU usage metrics throughout the streaming duration, while Figure 14d displays RAM usage metrics for both Fog and Cloud. It is evident that resource utilization remains stable in both environments. This stability implies a larger allocation of video frames to the Cloud. Given Fog's constrained resources and the substantial demand from Application 1, the resource utilization on the Fog side remains consistently high.



(a) Throughput Metrics (b) Communication Times Metrics

(c) CPU Usage Metrics (d) RAM Usage Metrics

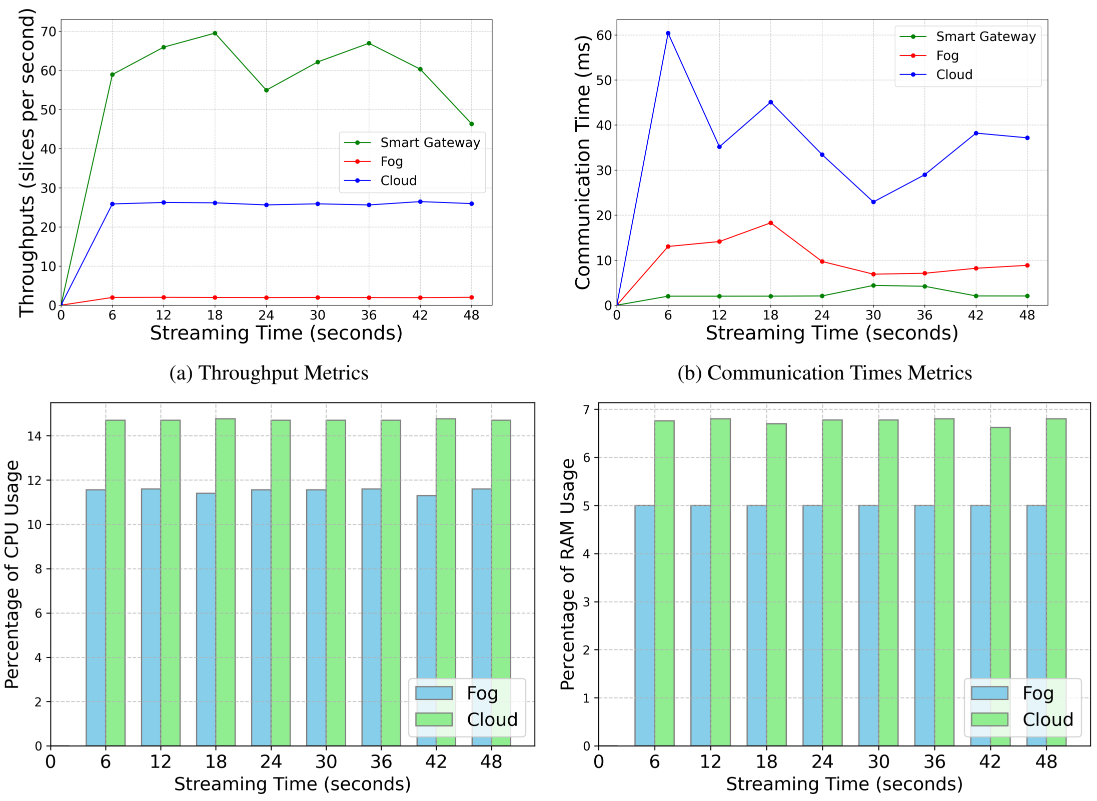
Figure 14. Application 1 - In-depth analysis metrics Metrics

5.3.2 Application 2: Text-audio synchronisation or forced alignment

The Figure 9 shows a comprehensive flowchart outlining the methodology employed within this experiment.

Figure 15a represents the throughput metrics associated with each environment. Throughput is quantified in terms of slices per second, denoting sets of audio and text segments per second during this experiment. The data demonstrates consistent throughput levels for Fog and Cloud throughout the streaming duration. However, the Smart Gateway's throughput experiences slight fluctuations in the 24th and 48th seconds of streaming. This scenario could occur due to the presence of multiple threads within the Smart Gateway, each tailored for different functions. These threads work simultaneously to manage tasks like audio and text processing, distribution of slices, and displaying the output.

Figure 15b illustrates the communication times among the environments during streaming. Observable is a fluctuating trend across Fog, Cloud, and Smart Gateway. This



(a) Throughput Metrics

(b) Communication Times Metrics

(c) CPU Usage Metrics

(d) RAM Usage Metrics

Figure 15. Application 2 - In-depth analysis metrics Metrics

variability can be attributed to the dynamic nature of these environments. For Application 2, the data demonstrates that communication times are minimal across various settings. This is unlikely to significantly impact assignment process deadlines for slices, thereby empowering agents to efficiently allocate slices drawing from their prior experiences. The findings for both throughput and communication time in Fog and Cloud environments imply the importance of maintaining a consistent and stable resource utilization.

In Figure 15c, we are presented with metrics regarding CPU usage throughout the streaming period. Meanwhile, Figure 15d provides insight into RAM usage for both Fog and Cloud. It is evident that resource utilization remains stable across both environments. Furthermore, a notable observation is the larger allocation of audio and text segments to the Cloud.

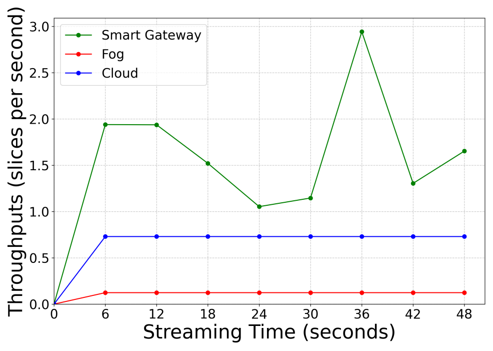
5.3.3 Application 3: Speech-to-text conversion

The flowchart shown in Figure 11 presents a comprehensive flowchart outlining the methodology employed in this experiment.

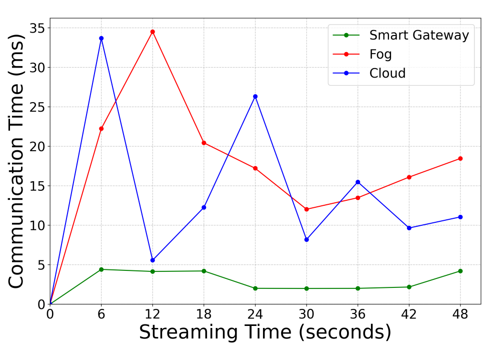
Figure 16a illustrates the throughput metrics related to each environment, quantifying the number of audio segment sets processed per second during the experiment. The results indicates consistent throughput levels for both Fog and Cloud throughout the streaming period. However, the Smart Gateway's throughput experiences a fluctuating trend during the streaming time. Similar to previous applications results, the current situation may also arise because of the existence of numerous threads within the Smart Gateway, each designed for distinct purposes. These threads function concurrently to oversee activities such as audio, allocation of slices, and displaying of output.

Communication times among the environments during streaming are depicted in Figure 16b. A fluctuating trend can be observed across Fog, Cloud, and Smart Gateway, with this variability attributed to the dynamic nature of these environments. Regarding Application 3, the data indicates that communication times remain minimal across different scenarios. This is improbable to substantially affect the assignment process timelines for slices. As a result, agents can effectively distribute slices based on their past encounters. The results for both throughput and communication duration in Fog and Cloud environments underscore the significance of upholding steady and dependable resource utilization.

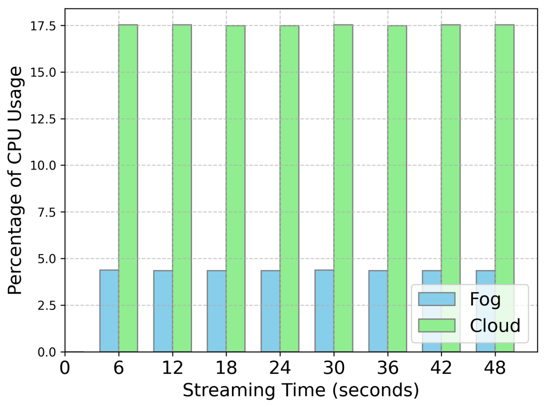
Metrics regarding CPU usage throughout the streaming period are presented in Figure 16c. Additionally, Figure 16d offers insights into RAM usage for both Fog and Cloud. It is evident that resource utilization remains stable across both environments. Furthermore, a notable observation is the larger allocation of audio segments to the Cloud.



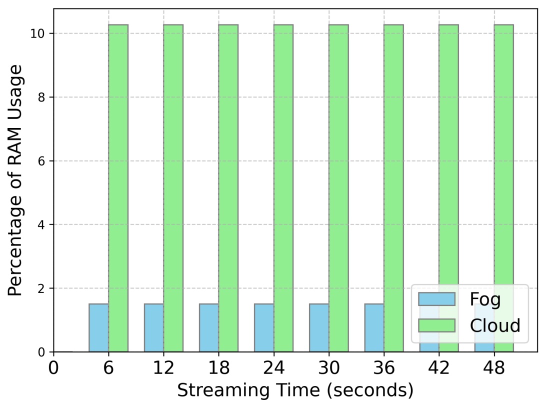
(a) Throughput Metrics



(b) Communication Times Metrics



(c) CPU Usage Metrics



(d) RAM Usage Metrics

Figure 16. Application 3 - In-depth analysis metrics Metrics

5.4 Summary

In this chapter, we not only presented our comprehensive research findings but also delved into a meticulous exploration of the intricate process of evaluation metric selection and its profound significance within the context of our study. By meticulously analyzing and selecting the most appropriate evaluation metrics, we aimed to ensure the robustness and accuracy of our research outcomes, thereby reinforcing the credibility of our study's conclusions. In the next chapter we will discuss future research directions.

6 Future research directions

The proposed DRL algorithm is implemented and tested using data-intensive applications to provide a comprehensive solution to the problem at hand. However, it is crucial to recognize the limits of our study. This section provides a more detailed description of the potential features that can be integrated.

6.1 Network Bandwidth Demand

In the context of optimizing service placement within the Edge-Cloud continuum, a critical aspect that warrants further exploration is the dynamic assessment and management of network bandwidth demand. This involves developing mechanisms to monitor and predict the network traffic patterns between edge devices and fog and cloud resources. By leveraging Deep Reinforcement Learning (DRL) techniques, future research could focus on creating intelligent algorithms capable of adapting service placement decisions based on real-time network bandwidth demand fluctuations. Such an approach would contribute to enhancing the overall efficiency and performance of the system by ensuring that service placements are aligned with the available network resources.

6.2 Disk I/O Bandwidth

Effective utilization of disk I/O bandwidth holds immense significance in the successful execution of services within the Edge-Cloud continuum. Extending the current research, an intriguing avenue for future investigation involves the integration of DRL-driven strategies to optimize service placements with a keen emphasis on minimizing disk I/O bottlenecks. This entails designing algorithms that intelligently distribute services across fog and cloud environments, taking into account the disk I/O characteristics and constraints of each environment and each application. A deeper exploration of this aspect promises to yield insights into enhancing data access efficiency and overall system responsiveness.

6.3 Service Cost Constraint

While the existing work focuses on optimizing service placement using DRL techniques, an exciting direction for future research is the integration of service cost constraints into the decision-making process. This entails devising algorithms that not only optimize for performance metrics but also account for the monetary costs associated with deploying services in the Edge-Cloud continuum. By incorporating cost-awareness into the optimization framework, researchers can contribute to the development of more practical and economically viable solutions. This can lead to the creation of strategies that strike a balance between performance optimization and cost-effectiveness, further enhancing the applicability of the proposed approach.

6.4 Energy Consumption

Addressing the sustainability aspect of the Edge-Cloud continuum, a critical dimension that deserves attention in future research is the management of energy consumption in fog and cloud environments. Expanding on the existing DRL-driven service placement strategy, researchers can explore novel algorithms that take into consideration the energy consumption profiles of fog and cloud resources. By incorporating energy-awareness into the decision-making process, the aim is to develop intelligent placement strategies that optimize not only for performance but also for minimizing energy consumption. This line of inquiry contributes to the broader goal of creating more environmentally friendly and energy-efficient computing systems.

6.5 Multi-Agent DRL Architecture

Building upon the foundation of DRL-driven service placement, an intriguing avenue for future work involves the exploration of multi-agent DRL architectures. This entails the development of sophisticated algorithms that enable multiple autonomous agents to collaboratively optimize service placements within the Edge-Cloud continuum. The multi-agent approach opens up possibilities for decentralized decision-making, enabling agents to interact, negotiate, and adapt their strategies in response to dynamic environmental changes. By embracing the multi-agent paradigm, researchers can unlock new levels of adaptability, scalability, and robustness in optimizing service placement, thus paving the way for more advanced and resilient Edge-Cloud continuum.

7 Conclusions

In this study, we looked at how well a deep reinforcement learning (DRL) strategy worked for distributing services between cloud and fog computing environments. To evaluate the effectiveness and performance of our suggested solution, we concentrated on success rate, distribution of service request slices and in-depth analysis encompassing various metrics for all three data intensive applications.

Our results show that applying a deep reinforcement learning-based artificial intelligence technique for service allocation produces promising results. The Success Rate indicator showed that our method successfully assigned services to the proper computing environment, resulting in a high rate of successfully completed service executions. This increase in success rates is a strong evidence of improved system performance and user satisfaction.

The author faced various difficulties that required to his focus and use his problem-solving abilities during the research process. One significant obstacle was the creation of the fundamental elements for the Deep Reinforcement Learning (DRL) algorithm: the State Space, Action Space, and Reward function. Designing these components carefully was essential to provide the algorithm with what it needed to learn effectively. As the experiment progressed, the author encountered a notable challenge related to handling data intensive applications. This challenge involved strengthening these applications to handle a large number of requests without any issues. He had to develop robust and resilient applications that could handle the high volume of requests.

As a result, our research adds to the body of information on the real time service distribution in Edge-Cloud continuum and highlights the advantages of deep reinforcement learning as a useful strategy. We can create the foundation for even more effective, long-lasting, and responsive distributed systems in the future by taking into account the constraints and expanding on our discoveries.

Acknowledgements

I would like to express how grateful I am to my dear family and my cherished friends Elnur and Agil. They have been a continuous source of encouragement and inspiration throughout my research journey. Their steadfast support, uplifting words, and unwavering belief in me have been truly invaluable, and I could not have been reached this milestone without them.

Moreover, I would like to extend my genuine gratitude to my supervisor, Chinmaya Kumar Dehury, Ph.D., for his expert guidance, thoughtful feedback, and constant dedication to my success. His mentorship has been pivotal in shaping this research, and I am profoundly thankful for his patience, devotion, and willingness to go the extra mile in aiding my academic growth.

Finally, I want to convey my appreciation to the faculty and staff of the Computer Science at the University of Tartu.

References

[1] Y. Dai, W. Liu, H. Li, and L. Liu, "Efficient foreign object detection between psds and metro doors via deep neural networks," *IEEE Access*, vol. 8, pp. 46723–46734, 2020.

[2] H. Elazhary, "Internet of things (iot), mobile cloud, cloudlet, mobile iot, iot cloud, fog, mobile edge, and edge emerging computing paradigms: Disambiguation and research directions," *Journal of Network and Computer Applications*, vol. 128, pp. 105–140, 2019.

[3] P. K. Sahoo, C. K. Dehury, and B. Veeravalli, "Lvrm: On the design of efficient link based virtual resource management algorithm for cloud platforms," *IEEE Transactions on Parallel and Distributed Systems*, vol. 29, no. 4, pp. 887–900, 2018.

[4] I. Stojmenovic, S. Wen, X. Huang, and H. Luan, "An overview of fog computing and its security issues," *Concurrency and Computation: Practice and Experience*, vol. 28, no. 10, pp. 2991–3005, 2016.

[5] V. Souza, X. Masip-Bruin, E. Marín-Tordera, S. Sànchez-López, J. Garcia, G. Ren, A. Jukan, and A. Juan Ferrer, "Towards a proper service placement in combined fog-to-cloud (f2c) architectures," *Future Generation Computer Systems*, vol. 87, pp. 1–15, 2018.

[6] S. Yi, Z. Hao, Z. Qin, and Q. Li, "Fog computing: Platform and applications," in *2015 Third IEEE workshop on hot topics in web systems and technologies (HotWeb)*, pp. 73–78, IEEE, 2015.

[7] M. Aazam and E.-N. Huh, "Fog computing and smart gateway based communication for cloud of things," in *2014 International conference on future internet of things and cloud*, pp. 464–470, IEEE, 2014.

[8] C. K. Dehury, P. Kumar Donta, S. Dustdar, and S. N. Srirama, "Ccei-iot: Clustered and cohesive edge intelligence in internet of things," in *2022 IEEE International Conference on Edge Computing and Communications (EDGE)*, pp. 33–40, 2022.

[9] H. Kim, "Fog computing and the internet of things: extend the cloud to where the things are," *Int. J. Cisco*, 2016.

[10] T. R. Chhetri, C. K. Dehury, A. Lind, S. N. Srirama, and A. Fensel, "A combined system metrics approach to cloud service reliability using artificial intelligence," *Big Data and Cognitive Computing*, vol. 6, no. 1, 2022.

[11] S. Yi, Z. Qin, and Q. Li, "Security and privacy issues of fog computing: A survey," in *Wireless Algorithms, Systems, and Applications: 10th International Conference, WASA 2015, Qufu, China, August 10-12, 2015, Proceedings 10*, pp. 685–695, Springer, 2015.

[12] F. Bonomi, R. Milito, P. Natarajan, and J. Zhu, "Fog computing: A platform for internet of things and analytics," *Big data and internet of things: A roadmap for smart environments*, pp. 169–186, 2014.

[13] A. V. Dastjerdi, H. Gupta, R. N. Calheiros, S. K. Ghosh, and R. Buyya, "Fog computing: Principles, architectures, and applications," in *Internet of things*, pp. 61–75, Elsevier, 2016.

[14] M. Taneja and A. Davy, "Resource aware placement of iot application modules in fog-cloud computing paradigm," in *2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, pp. 1222–1228, IEEE, 2017.

[15] O. Skarlat, M. Nardelli, S. Schulte, and S. Dustdar, "Towards qos-aware fog service placement," in *2017 IEEE 1st international conference on Fog and Edge Computing (ICFEC)*, pp. 89–96, IEEE, 2017.

[16] A. Brogi and S. Forti, "Qos-aware deployment of iot applications through the fog," *IEEE internet of Things Journal*, vol. 4, no. 5, pp. 1185–1192, 2017.

[17] R. Mahmud, S. N. Srirama, K. Ramamohanarao, and R. Buyya, "Quality of experience (qoe)-aware placement of applications in fog computing environments," *Journal of Parallel and Distributed Computing*, vol. 132, pp. 190–203, 2019.

[18] F. A. Salaht, F. Desprez, and A. Lebre, "An overview of service placement problem in fog and edge computing," *ACM Computing Surveys (CSUR)*, vol. 53, no. 3, pp. 1–35, 2020.

[19] J. Wang and D. Li, "Task scheduling based on a hybrid heuristic algorithm for smart production line with fog computing," *Sensors*, vol. 19, no. 5, p. 1023, 2019.

[20] S. Zahoor, S. Javaid, N. Javaid, M. Ashraf, F. Ishmanov, and M. K. Afzal, "Cloud–fog–based smart grid model for efficient resource management," *Sustainability*, vol. 10, no. 6, p. 2079, 2018.

[21] H. Rafique, M. A. Shah, S. U. Islam, T. Maqsood, S. Khan, and C. Maple, "A novel bio-inspired hybrid algorithm (nbiha) for efficient resource management in fog computing," *IEEE Access*, vol. 7, pp. 115760–115773, 2019.

[22] A. Yasmeen, N. Javaid, O. U. Rehman, H. Iftikhar, M. F. Malik, and F. J. Muhammad, "Efficient resource provisioning for smart buildings utilizing fog and cloud based environment," in *2018 14th International Wireless Communications & Mobile Computing Conference (IWCMC)*, pp. 811–816, IEEE, 2018.

[23] V. Yadav, B. Natesha, and R. M. R. Guddeti, "Ga-pso: Service allocation in fog computing environment using hybrid bio-inspired algorithm," in *TENCON 2019-2019 IEEE Region 10 Conference (TENCON)*, pp. 1280–1285, IEEE, 2019.

[24] X. Ren, Z. Zhang, and S. M. Arefzadeh, "An energy-aware approach for resource managing in the fog-based internet of things using a hybrid algorithm," *International Journal of Communication Systems*, vol. 34, no. 1, p. e4652, 2021.

[25] P. Hosseinioun, M. Kheirabadi, S. R. K. Tabbakh, and R. Ghaemi, "A new energy-aware tasks scheduling approach in fog computing using hybrid meta-heuristic algorithm," *Journal of Parallel and Distributed Computing*, vol. 143, pp. 88–96, 2020.

[26] S. Javanmardi, M. Shojafar, V. Persico, and A. Pescapè, "Fpfts: A joint fuzzy particle swarm optimization mobility-aware approach to fog task scheduling algorithm for internet of things devices," *Software: practice and experience*, vol. 51, no. 12, pp. 2519–2539, 2021.

[27] J. Xu, Z. Hao, R. Zhang, and X. Sun, "A method based on the combination of laxity and ant colony system for cloud-fog task scheduling," *IEEE Access*, vol. 7, pp. 116218–116226, 2019.

[28] T. Djemai, P. Stolf, T. Monteil, and J.-M. Pierson, "A discrete particle swarm optimization approach for energy-efficient iot services placement over fog infrastructures," in *2019 18th International Symposium on Parallel and Distributed Computing (ISPDC)*, pp. 32–40, IEEE, 2019.

[29] D. Rahbari and M. Nickray, "Task offloading in mobile fog computing by classification and regression tree," *Peer-to-Peer Networking and Applications*, vol. 13, pp. 104–122, 2020.

[30] H. Bashir, S. Lee, and K. H. Kim, "Resource allocation through logistic regression and multicriteria decision making method in iot fog computing," *Transactions on Emerging Telecommunications Technologies*, vol. 33, no. 2, p. e3824, 2022.

[31] F.-J. Ferrández-Pastor, H. Mora, A. Jimeno-Morenilla, and B. Volckaert, "Deployment of iot edge and fog computing technologies to develop smart building services," *Sustainability*, vol. 10, no. 11, p. 3832, 2018.

[32] J. He, J. Wei, K. Chen, Z. Tang, Y. Zhou, and Y. Zhang, "Multitier fog computing with large-scale iot data analytics for smart cities," *IEEE Internet of Things Journal*, vol. 5, no. 2, pp. 677–686, 2017.

[33] H. Priyabhashana and K. Jayasena, "Data analytics with deep neural networks in fog computing using tensorflow and google cloud platform," in *2019 14th Conference on Industrial and Information Systems (ICIIS)*, pp. 34–39, IEEE, 2019.

[34] A. A. Alsaffar, H. P. Pham, C.-S. Hong, E.-N. Huh, and M. Aazam, "An architecture of iot service delegation and resource allocation based on collaboration between fog and cloud computing," *Mobile Information Systems*, vol. 2016, 2016.

[35] S. Yadav, R. Mohan, and P. K. Yadav, "Task allocation model for optimal system cost using fuzzy c-means clustering technique in distributed system.," *Ingénierie des Systèmes d Inf.*, vol. 25, no. 1, pp. 59–68, 2020.

[36] P. Farhat, H. Sami, and A. Mourad, "Reinforcement r-learning model for time scheduling of on-demand fog placement," *the Journal of Supercomputing*, vol. 76, pp. 388–410, 2020.

[37] A. I. Orhean, F. Pop, and I. Raicu, "New scheduling approach using reinforcement learning for heterogeneous distributed systems," *Journal of Parallel and Distributed Computing*, vol. 117, pp. 292–302, 2018.

[38] Z. Tang, X. Zhou, F. Zhang, W. Jia, and W. Zhao, "Migration modeling and learning algorithms for containers in fog computing," *IEEE Transactions on Services Computing*, vol. 12, no. 5, pp. 712–725, 2018.

[39] P. Gazori, D. Rahbari, and M. Nickray, "Saving time and cost on the scheduling of fog-based iot applications using deep reinforcement learning approach," *Future Generation Computer Systems*, vol. 110, pp. 1098–1115, 2020.

[40] M. G. R. Alam, M. M. Hassan, M. Z. Uddin, A. Almogren, and G. Fortino, "Autonomic computation offloading in mobile edge for iot applications," *Future Generation Computer Systems*, vol. 90, pp. 149–157, 2019.

[41] Q. Zhang, M. Lin, L. T. Yang, Z. Chen, S. U. Khan, and P. Li, "A double deep q-learning model for energy-efficient edge scheduling," *IEEE Transactions on Services Computing*, vol. 12, no. 5, pp. 739–749, 2018.

[42] H. Lu, X. He, M. Du, X. Ruan, Y. Sun, and K. Wang, "Edge qoe: Computation offloading with deep reinforcement learning for internet of things," *IEEE Internet of Things Journal*, vol. 7, no. 10, pp. 9255–9265, 2020.

[43] Y. Wang, Y. Li, T. Lan, and N. Choi, "A reinforcement learning approach for online service tree placement in edge computing," in *2019 IEEE 27th International Conference on Network Protocols (ICNP)*, pp. 1–6, IEEE, 2019.

[44] C. K. Dehury and S. N. Srirama, "Personalized service delivery using reinforcement learning in fog and cloud environment," in *Proceedings of the 21st International Conference on Information Integration and Web-Based Applications & Services*, pp. 522–529, 2019.

[45] C. K. Dehury and S. N. Srirama, "An efficient service dispersal mechanism for fog and cloud computing using deep reinforcement learning," 2020.

[46] W. J. M. Levelt and A. S. Meyer, "Word for word: Multiple lexical access in speech production," *European Journal of Cognitive Psychology*, vol. 12, no. 4, pp. 433–452, 2000.

[47] J. McChesney, N. Wang, A. Tanwer, E. De Lara, and B. Varghese, "Defog: fog computing benchmarks," in *Proceedings of the 4th ACM/IEEE Symposium on Edge Computing*, pp. 47–58, 2019.

[48] T. H. Luan, L. Gao, Z. Li, Y. Xiang, G. Wei, and L. Sun, "Fog computing: Focusing on mobile users at the edge," *arXiv preprint arXiv:1502.01815*, 2015.

[49] R. Yu, G. Xue, and X. Zhang, "Application provisioning in fog computing-enabled internet-of-things: A network perspective," in *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*, pp. 783–791, 2018.

[50] H. Shah-Mansouri and V. W. S. Wong, "Hierarchical fog-cloud computing for iot systems: A computation offloading game," *IEEE Internet of Things Journal*, vol. 5, no. 4, pp. 3246–3257, 2018.

[51] R. Deng, R. Lu, C. Lai, and T. H. Luan, "Towards power consumption-delay tradeoff by workload allocation in cloud-fog computing," in *2015 IEEE International Conference on Communications (ICC)*, pp. 3909–3914, 2015.

[52] R. Deng, R. Lu, C. Lai, T. H. Luan, and H. Liang, "Optimal workload allocation in fog-cloud computing toward balanced delay and power consumption," *IEEE Internet of Things Journal*, vol. 3, no. 6, pp. 1171–1181, 2016.

[53] X. Fei, N. Shah, N. Verba, K.-M. Chao, V. Sanchez-Anguix, J. Lewandowski, A. James, and Z. Usman, "Cps data streams analytics based on machine learning for cloud and fog computing: A survey," *Future generation computer systems*, vol. 90, pp. 435–450, 2019.

[54] M. Selimi, L. Cerdà-Alabern, F. Freitag, L. Veiga, A. Sathiaseelan, and J. Crowcroft, "A lightweight service placement approach for community network micro-clouds," *Journal of Grid Computing*, vol. 17, pp. 169–189, 2019.

Appendix

I. Source Code

The source code of applications and developed algorithms for this study is located in repository `https://github.com/chinmaya-dehury/AI4FogCloudServiceDisperse`.The access to the repository could be granted by sending an email to `chinmaya.dehury@ut.ee`

II. Writing Assistance

ChatGPT is an instance of the GPT-3.5 architecture, a variant of the GPT (Generative Pre-trained Transformer) model developed by OpenAI that is able to process and generate natural language text. Natural Language Model ChatGPT's assistance was leveraged in the writing process of this thesis to fix grammar mistakes and reword some sentences in academic style.

III. Licence