Ministry of Higher Education and Scientific Research
Hassiba Benbouali University of Chlef
Faculty of exact sciences and computer science
Department of Computer Science

# THESIS

Submitted in partial fulfillment of the requirements for the degree of

## DOCTORATE

Field: Computer Science

Specialization: Artificial Intelligence and Software Engineering

By

**BENABOURA AMINA**

Theme :

## A task classification strategy for IoT/Cloud collaboration

Defended on 29/01/2026, before the jury composed of:

| | | | |
|---|---|---|---|
| Aridj Mohamed | MCA | University of Chlef | President |
| Bechar Rachid | MCA | University of Chlef | Rapporteur |
| Kadri Walid | MCB | University of Chlef | Co-rapporteur |
| Louazani Ahmed | MCA | University of Blida 1 | Examiner |
| Tahar Abbes Mounir | Professor | University of Chlef | Examiner |
| Douga Yassine | MCA | University of Blida 1 | Examiner |

Academic year: 2025/2026

# Acknowledgements

First and foremost, I thank God Almighty for granting us the health and determination to accomplish this work.
I would like to express my deep gratitude to my supervisor Mr. Bechar Rachid from chlef university, for his guidance, valuable advice, and availability throughout this work. His scientific and human guidance greatly contributed to the progress and quality of this study.
My thanks go to Mr. Kadri Walid, from chlef university for having co-supervised me, for his orientation, availability, listening, and patience during the realization of this job.
I would like to thank everyone who helps me to improve my work. and who gave me any remark that helped me to perfect this manuscript.
I am extremely grateful to all the teachers and administrators in the computer science department.
I would also like to express my gratitude to all the members of the jury Dr. Aridj Mohamed, Dr. Louazani Ahmed, professor Tahar Abbes Mounir, and Dr. Douga Yassine, for agreeing to evaluate this work.

# *Dedication*

*I dedicate this modest project:*

*To the man of my life, my moral support and source of joy and happiness, the one who has always sacrificed himself to see me succeed, to my father whom I adore.*

*To the light of my life, the source of my never-ending efforts, who has always prayed for me, supported me, and stood by me so that I could achieve my goals and find happiness, my mother.*

*To the constant source of strength, my moral support, to the person whose unwavering patience, sacrifices, and encouragement have accompanied me throughout this journey, to my husband.*

*To my dear brother Abderahmen.*

*To my dear sisters Wafaa, Anfel, Noorelhouda, and Abrar, for their moral support and encouragement.*

*To everyone who has helped me in any way, near or far, please accept my gratitude.*

*Thanks!*

*Amina Benaboura*

# Abstract

There is a growing need for efficient computational offloading technologies that can handle large data flows while adhering to strict response time and power consumption constraints as a result of the explosive growth of Internet of Things (IoT) devices. Real-time IoT applications require scalability and responsiveness, which traditional cloud computing cannot provide. To address these issues, this thesis presents a task offloading framework based on Deep Q-Network (DQN) in a collaborative architecture between IoT, fog computing, and cloud computing levels for intelligent and adaptive decision-making that maximizes system performance. The DQN-based strategy achieves superior performance, with reductions of up to 50% in total cost, 33% in latency, and 25% in energy consumption compared to competing methods, according to extensive simulation experiments against state-of-the-art algorithms, such as bat, DJA, and DDPG-based approaches. The DQN algorithm also shows strong convergence behavior, low variance, and high reliability across multiple scenarios, confirming its robustness and adaptability in distributed computing environments. In order to improve Quality of Service (QoS) and Quality of Experience (QoE) in IoT–fog–cloud ecosystems, this work offers a scalable, data-driven offloading solution, which advances the expanding field of intelligent task management. The suggested framework opens perspectives for more independent and energy-conscious computing paradigms by laying the groundwork for future studies on multi-agent deep reinforcement learning, federated offloading techniques, and optical network-enhanced IoT infrastructures.

***Key words:*** *Internet of Things (IoT), Task Offloading, Energy Consumption Minimization, Latency, Cloud Computing, IoT System, Deep Q-Network (DQN)*

# Résumé

En raison de la croissance explosive des appareils connectés à l'Internet des Objets (IdO), il existe un besoin croissant de technologies efficaces de déchargement informatique capables de gérer d'importants flux de données tout en respectant des contraintes strictes en matière de temps de réponse et de consommation d'énergie. Les applications IdO en temps réel exigent une évolutivité et une réactivité que le cloud computing traditionnel ne peut offrir. Pour répondre à ces problèmes, cette thèse présente un cadre de déchargement des tâches basé sur le Réseau Q Profond (RQP) dans une architecture collaborative entre les niveaux IdO, fog computing et cloud computing pour une prise de décision intelligente et adaptative qui maximise les performances du système. La stratégie basée sur le RQP permet d'obtenir des performances supérieures, avec des réductions pouvant atteindre 50% du coût total, 33% de la latence et 25% de la consommation d'énergie par rapport aux méthodes concurrentes, selon des expériences de simulation approfondies par rapport aux algorithmes de pointe, tels que les approches basées sur bat, DJA et DDPG. L'algorithme RQP présente également un comportement de convergence élevé, une faible variance et une grande fiabilité dans de multiples scénarios, confirmant sa robustesse et son adaptabilité dans les environnements informatiques distribués. Afin d'améliorer la Qualité de Service (QdS) et la Qualité d'Expérience (QdE) dans les écosystèmes IdO-fog-cloud, ce travail propose une solution de déchargement évolutive et basée sur les données, qui fait progresser le domaine en pleine expansion de la gestion intelligente des tâches. Le cadre proposé ouvre des perspectives pour des paradigmes informatiques plus indépendants et plus économes en énergie en jetant les bases d'études futures sur l'apprentissage profond par renforcement multi-agents, les techniques de déchargement fédérées et les infrastructures IdO améliorées par les réseaux optiques.

***Mots clés :*** *Internet des Objets (IdO), déchargement des tâches, minimisation de la consommation d'énergie, latence, cloud computing, système IdO, Réseau Q Profond (RQP)*

# ملخص

هناك حاجة متزايدة لتقنيات تفريغ حسابي فعالة يمكنها التعامل مع تدفقات البيانات الكبيرة مع الالتزام بقيود صارمة على وقت الاستجابة واستهلاك الطاقة نتيجة للنمو الهائل لأجهزة إنترنت الأشياء. تتطلب تطبيقات إنترنت الأشياء في الوقت الفعلي قابلية التوسع والاستجابة، وهو ما لا تستطيع الحوسبة السحابية التقليدية توفيره. لمعالجة هذه المشكلات، تقدم هذه الأطروحة إطار عمل لتفريغ المهام يعتمد على خوارزمية الشبكة العصبية العميقة في بنية تعاونية بين مستويات إنترنت الأشياء والحوسبة الضبابية والحوسبة السحابية من أجل اتخاذ قرارات ذكية وقابلة للتكيف تعمل على تعظيم أداء النظام. تحقق الاستراتيجية القائمة على خوارزمية الشبكة العصبية العميقة أداءً فائقًا، مع انخفاض يصل إلى ٥٠٪ في التكلفة الإجمالية، و ٣٣٪ في زمن الاستجابة، و ٥٢٪ في استهلاك الطاقة مقارنة بالطرق المنافسة، وفقًا لتجارب محاكاة مكثفة مقابل أحدث الخوارزميات. تُظهر خوارزمية الشبكة العصبية العميقة أيضًا سلوكًا قويًا في التقارب، وتباينًا منخفضًا، وموثوقية عالية عبر سيناريوهات متعددة، مما يؤكد متانتها وقدرتها على التكيف في بيئات الحوسبة الموزعة. من أجل تحسين جودة الخدمة وجودة التجربة في أنظمة إنترنت الأشياء-الضباب-السحابة، يقدم هذا العمل حلاً قابلاً للتطوير وقائمًا على البيانات لتفريغ الأحمال، مما يعزز مجال إدارة المهام الذكية الآخذ في التوسع. يفتح الإطار المقترح آفاقًا لنماذج حوسبة أكثر استقلالية ووعيًا بالطاقة من خلال إرساء الأساس لدراسات مستقبلية حول التعلم المعزز العميق متعدد الوكلاء وتقنيات التفريغ الموحدة والبنى التحتية لإنترنت الأشياء المعززة بالشبكات الضوئية.

**كلمات مفتاحية**: إنترنت الأشياء ، تفريغ المهام ، تقليل استهلاك الطاقة ، زمن الاستجابة ، الحوسبة السحابية ، نظام إنترنت الأشياء، خوارزمية الشبكة العصبية العميقة.

# Contents

# List of Figures

# List of Tables

# List of Abbreviations

| | |
|---|---|
| **IoT**: | Internet of Things. |
| **DQN**: | Deep Q-Network. |
| **QOE**: | Quality of Experience. |
| **QoS**: | Quality of Service. |
| **NAS**: | network-attached storage. |
| **AI**: | artificial intelligence. |
| **SQL**: | Structured Query Language. |
| **PSO**: | Particle Swarm Optimization. |
| **GA**: | Genetic Algorithm. |
| **ACO**: | Ant Colony Optimization. |
| **BLA**: | Bees Life Algorithm. |
| **FPGA**: | Field Programmable Gate Arrays. |
| **PGA**: | Priority–aware GA. |
| **PDST**: | percentage of deadline satisfied tasks. |
| **LTB-TAOO** : | Latency Time Based Task Assignment Combine with Online and Offline. |
| **MD**: | mobile devices. |
| **VR/AR**: | Virtual Reality and augmented reality. |
| **DRL**: | Deep Reinforcement Learning. |
| **RL**: | Reinforcement learning. |
| **DL**: | Deep Learning. |
| **CPU**: | Central Processing Unit. |
| **RAM**: | Random Access Memory. |
| **SVM**: | Support Vector Machines. |
| **ANN**: | Artificial Neural Networks. |
| **DNN**: | Deep Neural Networks. |
| **DDPG**: | Deep Deterministic Policy Gradient. |

| | |
|---|---|
| **MARL**: | Multi-Agent Reinforcement Learning. |
| **NSANNOM**: | Network through Smart ANN-based Offloading Mechanism. |
| **OCOP**: | Optimal Clustering and Offloading Parameters. |
| **LTC**: | Latency Time Compute. |
| **MEC**: | Mobile Edge Computing. |
| **PPO**: | Proximal Policy Optimization. |
| **MIMO**: | Multiple-Input Multiple-Output. |
| **ACORL**: | Adaptive Computation Offloading Model. |
| **IoMT**: | Internet of Medical Things. |
| **DM-MIMO**: | Distributed Massive Multiple-Input Multiple-Output. |
| **IoV**: | Internet of Vehicles. |
| **PSOS**: | Particle Swarm Optimization Strategy. |
| **NSGA-II**: | Non-dominated Sorting Genetic Algorithm II. |
| **IC**: | Internet Cloud. |
| **ILP**: | Integer Linear Programming. |
| **HWGA**: | Hybrid Whale Genetic Algorithm. |
| **VEC**: | Vehicular Edge Computing . |
| **RDA**: | Red Deer Algorithm. |
| **SA**: | Simulated Annealing. |
| **RDSA**: | Hybrid Meta-Heuristic Algorithm. |
| **FNs**: | Fog Nodes. |
| **MFLOPs**: | Mega Floating-Point Operations Per second. |
| **CI**: | Confidence Interval. |
| **MB**: | Mega bits. |
| **UAV** : | Unmanned Aerial Vehicle. |
| **CB** : | CPU cycles per Bit. |
| **DJA**: | Discrete Jaya Algorithm . |

# General introduction

## Introduction

In the information technology era, data is the most important commodity. In data-driven enterprises, having more data usually generates more value [2]. According to estimations of the International Data Corporation, at the end of the year 2025, the number of connected devices will be more than 41 billion, with the ability to generate more than 79 zettabytes annually [3]. By 2030, the number may rise to 500 billion connected devices. These connected devices constitute the internet of things (IoT), which has the capacity to generate data [4]. The exponential growth of IoT brought about the birth of large networks of mobile devices, creating a huge amount of data. These devices are frequently used, but they are often constrained by their limited capabilities in terms of computation power, capacity of storage, and energy consumption. In current IoT applications, most data that needs storage, analysis, and computation power are sent to the cloud centers [5].

Cloud computing is a radically new computing paradigm that provides on- demand and easy access to a common, and unlimited pool of resources that can include storage, processing power, and applications over the internet. The vast pool of resources and services has enabled the emergence of several new applications, such as virtual reality, smart grids, and intelligent environments [6]. Thus, individuals and organizations are freed from building and maintaining a large on-premise infrastructure, allowing them to have flexible and scalable resources, which are cloud-provisioned and realized dynamically. In this model, cloud computing becomes a pay-as-you-go utility, making it a cost-effective, efficient technology, available to any business, small to very large. The benefits of cloud computing – minimal management effort, convenience, rapid elasticity, pay per use, ubiquity – have given birth to a multi-billion industry that is growing worldwide. The reliance on centralized cloud computing leads to considerable physical and logical separation between IoT devices and the cloud, which causes higher network latency and inconsistent performance [7]. This central-

ization leads to increased delays in data processing and transmission, hindering the real-time response required by many IoT applications. The challenge becomes more pronounced and intense as more and more IoT devices and objects integrate in the human daily life [8]. The traditional cloud computing paradigm is unable to meet the critical requirements such as: low latency, location awareness, and mobility support effectively. Moreover, in some applications, transmitting the data to the cloud may not be practical solution due to privacy concerns.

Over the years, several researchers have proposed new technique to address the urgent need for computing paradigm that is near the connected devices, particularly in terms of geographic and bandwidth dissemination-related problems in latency. The fog computing paradigm is the central concept that provides cloud functionality of processing, and storage power, to minimize latency and maximize bandwidth [9, 10]. Any connected device from the IoT level can benefit from the fog computing services. In addition, while fog can operate independently and collaborate with each other, they are not separate from the traditional cloud. The installation of fog provides a practical and imaginary solution for real-time applications has become a popular approach in this paradigm [11]. In many applications, IoT devices produce large computational tasks. However, the limited energy available for compute and storage services necessitates moving tasks to fog close to the devices, rather than regularly travelling through the cloud [12]. Moreover, the fog is a mediating platform between the physical IoT devices (users) and the cloud infrastructure so as to fulfill most of its tasks for performance explicitly optimization for delay, workload balancing, and resource utilization - a term generalizing task offloading [13].In this context, the main objective of this thesis is to propose a task classification method designed to optimize the offloading process in IoT–fog–cloud systems [14]. The task offloading is achieved by mutual benefits through fog-cloud combinations, offering continuous IoT services based on various quality of service (QoS) requirements. However, the introduction of a fog creates a major concern: whether to offload tasks to the fog or to the cloud. Several factors significantly affect the study regarding policies for offloading decisions, including deadlines and availability of computation sources and fog node capabilities. By employing a deep Q-network (DQN)-based learning framework, the proposed approach dynamically learns optimal task classification and offloading policies that minimize both energy consumption and latency, thus improving overall system efficiency and adaptability. This method intelligently classifies tasks according to their computational requirements, latency sensitivity, and resource constraints, in order to determine the most appropriate execution layer (IoT device, fog node, or cloud server).

## Challenges and Motivation

As thoroughly discussed, computational offloading is driven by the need to overcome resource constraints on IoT devices, which often lack performance, energy efficiency, scalability, and cost requirements while enabling advanced applications. IoT devices can save energy, prolong battery life, and enhance performance by shifting computationally demanding tasks, such as real-time or time-sensitive data analytics, machine learning inference, or video processing, to more powerful cloud server or fog nodes. Furthermore, choosing which part of the task should be offloaded to the edge is a challenge in the offloading decision-making process, as these tasks are dynamically generated [15]. It is also difficult to decide where to offload: dividing up tasks between local, fog nodes, and cloud server requires balancing trade-off between compute efficiency, energy usage, and communication delay, particularly for applications that need real-time responses [16]. These challenges highlight the need for intelligent offloading strategies that learn with dynamic environments while provide efficient and effective resource use. Computational offloading can help IoT systems reach their full potential by resolving these issues and allowing them to provide effective, scalable, and responsive solutions for a variety of applications. Finally, heterogeneity in interoperability across various platforms, along with financial costs of cloud services and fog infrastructure deployment, are challenges to large scale adoption. To address such challenges, novel novel adaptive IoT-fog-cloud architecture, smart task-partitioning algorithms, and standardized security protocols are required to unlock the full potential of offloading.

## Problem statement

The dynamic nature of IoT environments characterized by varying workloads, diverse devices, and unpredictable network conditions renders static or rule-based offloading policies ineffective, even with significant advances in cloud and fog computing. Traditional experimental and optimization techniques often fall short in real-time adaptation, leading to resource imbalances, communication delays, and increased energy consumption. This thesis uses DQN, a reinforcement learning (RL)-based technique that can learn optimal policies through interaction with the environment, to develop an intelligent and adaptive method for classifying and offloading tasks across the Internet of Things, fog, and cloud layers to overcome these difficulties.

## Research Methodology Overview

The study employs an organized methodology that combines experimental validation with theoretical modeling:

- **System Modeling**
  A three-layer IoT–fog–cloud architecture is modeled, taking into account practical factors such as processing capacity, task size, and bandwidth.

- **Problem formulation**
  The offloading problem is formulated mathematically as a joint optimization problem that minimizes the whole cost of the system, including energy consumption and latency

- **Algorithm Architecture**
  Using learnt policies and environmental conditions, a DQN-based learning algorithm is created to dynamically classify and assign tasks

- **Simulation and Evaluation**
  To evaluate the performance of the proposed algorithm based on a variety of criteria, it is put into practice and contrasted with cutting-edge techniques.

## Contributions

In this thesis, we examine the issue of computation offloading in a three-layer IoT–fog–cloud architecture, emphasizing task classification and intelligent placement to attain effective offloading choices. Important issues such constrained device resources, fluctuating network circumstances, and the trade-off between latency and energy usage are all addressed in the work. Based on the current system conditions, we suggest a DQN-based approach that can develop a dynamic and adaptive offloading policy that categorizes and allocates tasks to the best execution layer local, fog, or cloud. Therefore, the main contributions of this work are described as follows:

- - We formalize the task offloading and classification process in IoT-fog-cloud system as a joint optimization problem between energy consumption and offloading latency at each IoT device.

- To solve the formulated problem, we design DQN learning-based energy efficient and latency-aware algorithm for task classification optimal offloading decision strategies in

the IoT-fog-cloud collaboration model and provides a methodology for assigning tasks to different layers.

- We put into practice the suggested DQN-based framework, which does not require prior knowledge of the environment and automatically learns a task classification and offloading policy that minimizes the overall system cost.
- Through comprehensive simulations and performance comparisons with current state-of-the-art algorithms, we assess and validate the suggested model. The findings show that, in terms of latency, energy economy, and adaptability, the DQN-based task classification and offloading strategy performs noticeably better than conventional techniques.

## Thesis structure

The thesis is structured into 4 chapters as follows:

- **Chapter One**
  This chapter provides background information on the collaboration between the Internet of Things, fog computing, and cloud computing, as well as the reason for conducting the research. The chapter describes the problem, explains the objectives of the study, discusses the growing need for effective task offloading mechanisms, and highlights the importance of intelligent task classification and decision-making in distributed computing environments. The chapter concludes by summarizing and organizing the main contributions of the thesis.
- **Chapter Two**
  This chapter provides a comprehensive analysis of research on data and task offloading techniques in IoT – fog – cloud architectures. It divides task offloading strategies into two main categories: those based on machine learning and those that are not. Focusing on empirical, meta-empirical, and RL methodologies, the analysis highlights their advantages, disadvantages, and suitability for implementation in real-world IoT systems. The chapter concludes by identifying important research gaps that support the proposed strategy.
- **Chapter Three**
  This chapter provides a detailed introduction to the proposed offloading approach based on DQN. First, the system model, which includes cloud, fog, and Internet of Things

layers, as well as how they interact, is explained. The optimization problem is then formulated in the chapter as a joint minimization of latency and energy consumption. The mathematical model and constraints are then presented. In order to dynamically learn the best offloading options under different network conditions, a DQN algorithm is created. A task classification system that guides decision-making within the offloading framework is also presented in this chapter.

- **Chapter Four**
  In this chapter, we focus on the experimental evaluation and performance assessment of the proposed DQN-based offloading algorithm. The evaluation metrics such as latency, energy consumption, total cost, and convergence rate are defined and analyzed. The results are compared with those of existing methods, including bat, DJA, Optimal, and DDPG-based algorithms, under varying task loads and environmental conditions.

- **Chapter Five**
  The final chapter highlights the contributions of the proposed DQN-based offloading method and summarizes the main findings of the research. It concludes that deep reinforcement learning (DRL) provides a scalable and flexible method for effectively offloading tasks in IoT-fog-cloud contexts. Finally, the chapter suggests future research possibilities, such as real-time deployment scenarios, edge resource simulation, and multi-agent RL.

This structure highlights our research contribution, which is a smart strategy proposal for task classification and offloading based on DQN for IoT-fog-cloud systems, and ensures logical continuity between chapters as you progress from theoretical background to algorithm design and performance evaluation.

# Chapter 1

# IoT Systems Architecture

## 1.1 Introduction

IoT has emerged as a transformative force connecting physical devices and systems with the digital world, facilitating unprecedented levels of data collection, analysis, and automation. The architecture of the IoT system is considered a critical determinant in ensuring efficient, scalable, and reliable operation of such interconnected networks. The architecture of an IoT system includes the design, organization, and interaction of the various components that make up an IoT ecosystem. These include IoT devices, communication protocols, cloud infrastructure, data analysis engines, and user interfaces. The architecture must resolve major issues such as heterogeneity, interoperability, security, and scalability to ensure seamless communication and functionality between the different IoT devices. In a typical IoT architecture, there are three levels on which this chapter focuses: the IoT device level, the fog level, and the cloud level.

## 1.2 Notions of Internet of Things

IoT comprises a set of interconnected physical devices, which incorporate sensors, actuators, and other objects embedded with software, network connectivity, and data processing capabilities as shown in figure 1.1. These devices collect, transmit, and analyze data for automation, real-time decision-making, and enhanced efficiency in various applications. IoT systems combine hardware, software, and communication technologies to create smart environments that enhance user experiences and optimize processes [17]. The IoT is a growing technology with The potential to alter many aspects of life. Software engineers must understand the

rudimentary principles of IoT to facilitate its growth and acceptance.


Figure 1.1: The Internet of Things technology

## 1.3 Internet of Things applications

The Internet of Things incorporates several existing technologies, including hardware, software, sensing and control systems, information technology, and communication infrastructure, to enhance customer engagement, lower costs, improve processes, and spur innovation in products and business models [18]. IoT applications will touch many areas, including smart homes, where IoT automates lighting, security, and energy management, while in healthcare, it powers wearable devices and remote patient monitoring, and industrial IoT, which enhances manufacturing through predictive maintenance and asset tracking as illustrated in figure 1.2 . In smart cities, IoT enhances traffic management, waste monitoring, and public safety [19], while assisting in agriculture with precision farming and livestock tracking [20]. IoT also creates advanced applications for transportation with fleet management and autonomous vehicles, retail with smart shelves and a personalized shopping experience, and energy management with smart grids and meters. Industry efficiency, cost reduction, and decision making are enhanced by incorporating the IoT into these systems, opening the door to a more intelligent and connected society.

The development of creative smart applications across a range of industries is made possible by the IoT, which makes it simple to access and interact with a wide range of smart devices and performance items. Currently, semantic aspects are being incorporated into IoT applications across several sectors.

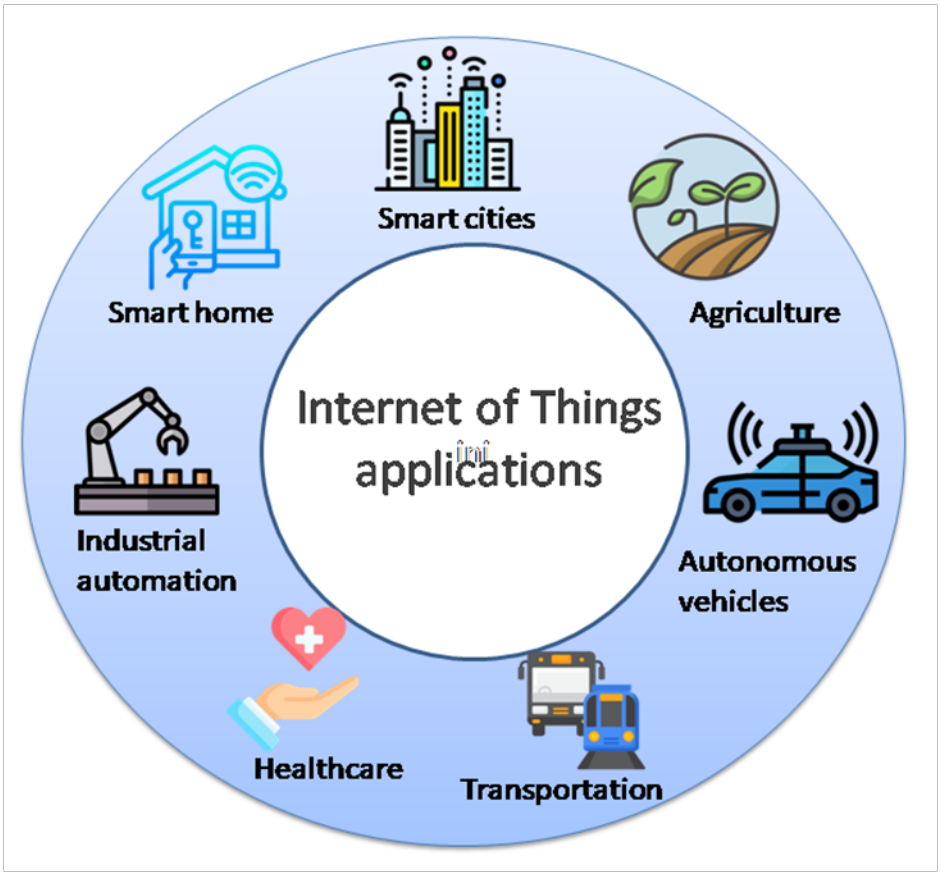


Figure 1.2: IoT applications

## 1.4 Benefits of Internet of Things systems

IoT is indeed highly beneficial when implemented in various sectors for its actual enhancement in efficiency, productivity, and standard of living. It provides real-time data collection and analysis, thus enabling data-supported decisions for organizations or individuals. Be it manufacturing that requires predictive maintenance to break down equipment downtime as well as operational costs by detecting problems before they develop, or the extent of having high automation, so much to streamlining processes and improving accuracy across manufacturing, logistics, and service sectors.

Transforming healthcare IoT devices, such as wearable sensors and patient monitoring

systems, enters a new world [21]. This makes it possible to provide ongoing health surveillance with input in real time on vital signs and patient conditions, which usually allows early detection of possible health disruptions, quickly leading to medical intervention [22]. Thus, such technology significantly improves patient care quality and health outcomes. Such as providing abnormal heart rates or oxygen levels through wearable devices brings the patient and health care going to proactive treatment and thus reduces hospital re-admissions [23].

Smart cities depend on the IoT in urbanization and improvement initiatives. This as well brings into play all of the connected devices in municipalities to make intelligent traffic management systems that would eventually relieve congestion, result in better commute times, and improve safety on the roads. There are also waste management sources that incorporate IoT which can optimize the collection routes and schedules into a concept of a clean city, and at the same time, reduce the operational costs dealing with it. Efficient energy systems in smart buildings regulate lights, heating, and cooling in accordance with the occupancy and environmental conditions thus save energy, and yet provide for comfort to the occupants. But IoT does not only concern urbanism; it is also part of a global sustainability perspective. Smart energy grids will allow better distribution and consumption of electricity, and they will also better integrate renewable energy resources. They will make real-time monitoring of energy use to have some reduction in carbon footprint through better resource handling and energy-saving mode [24]. For example, smart meters can make consumers observe their consumption behavior and be energy-conscious.

These benefits are indicative of the extent to which IoT will potentially transform industries as well as habits. Concerning innovations and efficiency, IoT improves the operational process, hence benefiting individual and environmental welfare. These collective features will, in all probability, because of technological integration even more efficiency into different industries in the future, revolutionizing how people will live, work, and relate with one another.

## 1.5 Challenges faced by Internet of Things

Due to the nature of IoT applications, it is crucial and challenging to design and develop a safe solution for devices with minimal resources. Many issues must therefore be resolved. IoT device requirements and resource constraints should be the focus of IoT solutions. This section highlights several research challenges and potential solutions associated with IoT as shown in figure 1.3. One of the primary challenges is the security and privacy risk, as IoT

devices are frequently attacked due to weak encryption, insecure communication protocols. These flaws can raise serious privacy concerns since they may result in data breaches, illegal access, and other privacy violations. Furthermore, interoperability remains a huge area of concern because, in the absence of common standards and protocols, the security capabilities of the IoT ecosystems are limited by devices across different manufacturers unable to align and converse harmoniously. Scalability is also a critical issue since the exponential expansion of connected devices put a strain on data storage, processing power, and network infrastructure, necessitating creative solutions to effectively handle large-scale deployments. Also, many of the proposed frameworks, while effective in controlled environments, face significant obstacles when scaling to large, heterogeneous IoT networks [25].

Another challenge is the energy efficiency particularly for battery-powered IoT devices mounted around odd locations or places that are not easily accessible. Energy consumption optimization is crucial in prolonging operational life and minimizing environmental impact [26]. Moreover, data management and analytics is another dimension of difficulties, as the IoT generates extremely huge amounts of data that must in turn be stored, processed, and analyzed in an optimized manner to ensure actionable insights while maximizing data precision and integrity. The main issues must be addressed in building a truly secure, scalable, and sustainable IoT ecosystem that would leverage the full transformational potential.

The solution to these challenges is collaboration and stakeholder participation, such as researchers, industry leaders, policymakers, and developers. Strong security measures, uniform standards, energy-optimized architecture, and ethical data practices are the key foundational blocks toward overcoming these hurdles and ensuring that the IoT ecosystem fulfills its general promise. As technology advances, proactive approaches and creative strategies will be needed to establish a secure, scalable, and sustainable IoT future. Only by facing these challenges head-on can we truly expect IoT to transform our ability and realize the interconnection of the entire world for the benefit of all.

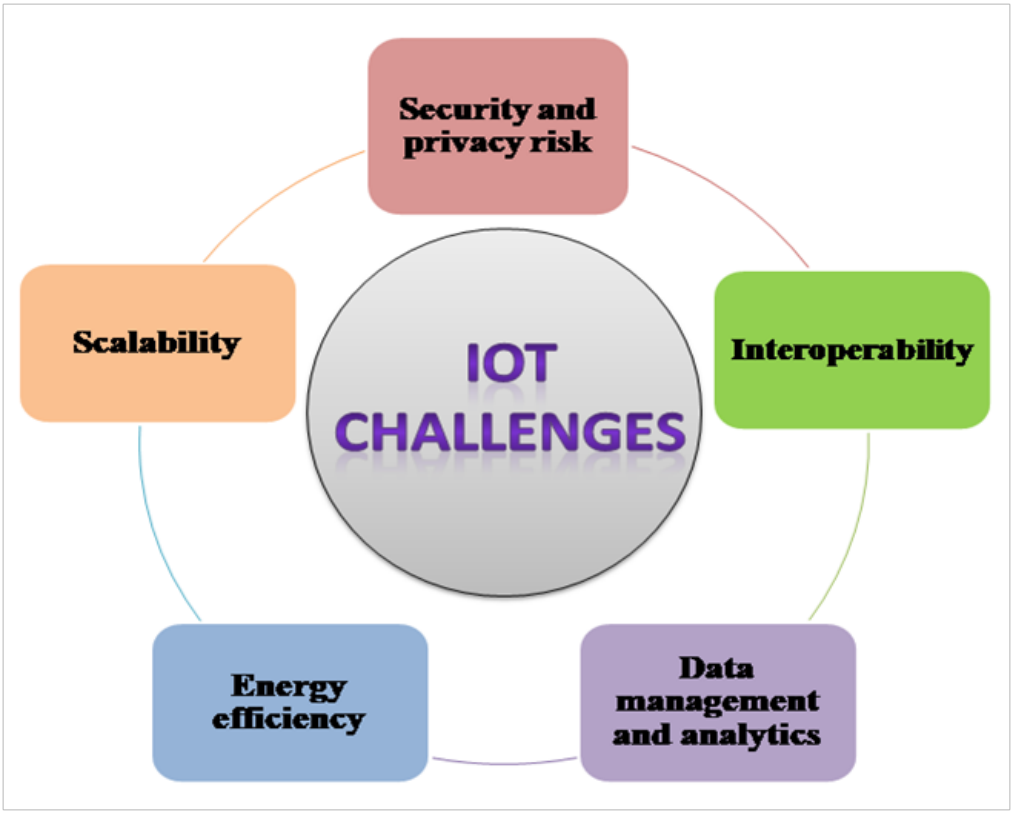


Figure 1.3: IoT challenges

## 1.6 IoT system architectures

The IoT architecture is the structured framework that describes how the different elements of an IoT system interact and work together. It usually consists of three primary layers that allow these devices to connect and communicate with one another: the network layer, the perception layer, and the IoT application layer [27]. There is no universally accepted consensus among experts and the global community regarding the architecture of the Internet of Things. Researchers have put forth a wide variety of architectures. According to some researchers, the most basic IoT architecture is composed of three layers. But the continuous development in IoT is not sufficient for research on IoT because research often focuses on finer aspects of the IoT system. That is why, we have more layered architectures proposed in this part. Therefore, some researchers support the four-layer architecture [28]. It has three layers like the previous architecture, but also has an additional layer called the support layer. This architecture played an important role in the development of the IoT system.

Despite these improvements, concerns regarding security, privacy, and data storage per-

sist in four-layer architectures [29]. To address these challenges, a five-layer architecture is proposed, which includes the processing layer as an additional layer dedicated to security and trust management. This model aims to enhance the security and privacy aspects of IoT ecosystems, making them more robust against potential cyber threats [30]. As IoT technology continues to evolve, researchers are actively exploring more sophisticated architectures to enhance efficiency, scalability, and security in interconnected environments.

## 1.7 Overview of IoT- fog- cloud system architecture

The IoT-fog-cloud system is an integrated computing paradigm, designed to effectively manage all huge amounts of data and to address the limitations of traditional cloud computing in specific terms concerning IoT systems, including high latency, bandwidth limitation, and scalability issues. It integrates the benefits of cloud computing, fog computing, and IoT devices into a multi-level hierarchical architecture to process, store, and make decisions. This three-tier system exploits the different layers to comprehensively make use of their resources. The architecture is designed to exploit the unique capabilities of each of the layers to create a robust, efficient, and scalable ecosystem for modern data-driven applications. Figure 1.4 describes the architecture of the IoT-fog-cloud system that integrates IoT-based devices along with the fog layer over short distances, with a long extension toward the cloud server. When the fog nodes computational ability is exceeded, the workloads from the IoT devices shift to the distant cloud server [31]. The three-layer collaboration model and roles in modern technology are considered in the following:

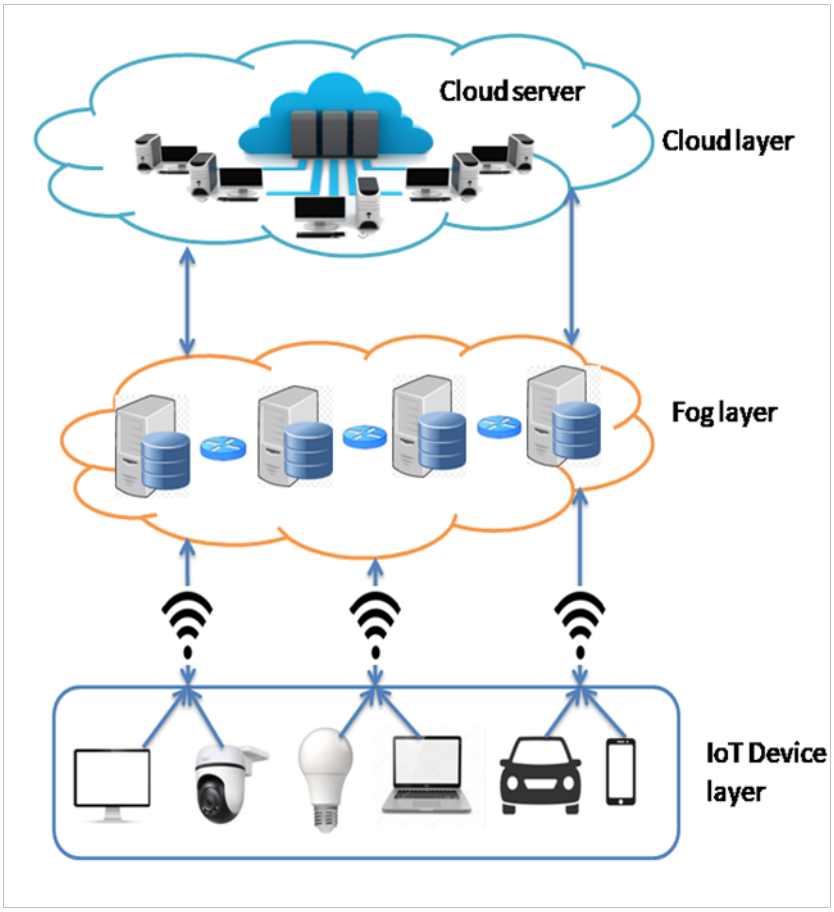


Figure 1.4: The architecture of the IoT-fog-cloud system

### 1.7.1 IoT device layer

The core component of the IoT-fog-cloud architecture is the IoT device layer. It consists of physical devices such as sensors, actuators, and embedded systems that interact directly with the environment to collect, monitor, and transmit data. The primary purpose of this layer is to collect real-time information such as temperature, humidity, motion, and light, as well as perform basic processing on it, without transmitting it to the fog or cloud layer for further processing [32]. These devices perform complex tasks, which require a wide range of computing power. These complex tasks, which are presented as online applications, raise the demands on computing power, required data rates, and other resources [33, 34]. However, with all the powerful features in the IoT devices, these devices are often resource-constrained, with limited power, memory, and processing capability. So they are unable to run new smart applications that require high processing capabilities, such as virtual reality, smart healthcare, some IoT applications, etc [35, 36]. The figures below show some examples

of IoT device-level.

Figure 1.5: A small temperature sensor


Figure 1.6: Smart wearable watch

Figure 1.7: A smart surveillance camera

### 1.7.2 Fog layer

The fog layer acts as an intermediate decentralized infrastructure between the IoT device layer and cloud layer; it extends the cloud services closer to the IoT device [37]. This layer consists of computing devices such as fog nodes (FNs), and micro data centers distributed over different regions, offering a range of functionalities [38]. Also, it includes networking equipment such as IoT gateways, routers, switches, and access points, ensuring efficient data transmission between IoT devices, the fog layer, and the cloud. It processes data locally, reducing the need to transfer large volumes of data to the cloud, thus reducing response time and bandwidth usage. The fog layer is particularly useful for real-time applications, such as smart cities, healthcare, and industrial automation, where low latency and high reliability are crucial. In addition, fog layer provides the ability to process tasks, manage networks, and store data in storage devices like local servers and network-attached storage (NAS) that are accessible to IoT devices. By its own very nature, fog computing supports user mobility, resource and interface heterogeneity, and also distributed data analytics for meeting the requirements and expectations of widely distributed applications that may not be well-suited for traditional cloud environments, such as applications which calls for real time response with very low latency [39, 40].

### 1.7.3 Cloud layer

The cloud layer operates as a centralized, scalable infrastructure that offers virtually unlimited computational power, storage, and services. It has elastic, reliable, and cost-effective computing resources to accommodate the large number of physical machines [41]. It includes data centers and cloud servers that handle vast amounts of IoT data, along with virtual machines and AI processors for optimized computing and machine learning applications.

The cloud also incorporates storage devices, such as distributed file systems, and databases ( structured query language (SQL), No-SQL, big data platform) for efficient data management. While the fog layer performs latency-sensitive tasks closer to the IoT device, the cloud layer handles resource-intensive tasks, complex decision making, and global data integration. All these services provide on-demand computing services such as storage and data processing [42]. Besides providing the above-mentioned services, cloud computing also focuses on the dynamic optimization of resources shared by many users.

In conclusion, even if there are several structural differences between the above paradigms, this does not imply that they should be ignored in favor of developments in other related domains. Due to the similarities between the paradigms, it is reasonable to assume that

there will be mechanisms and platforms that can offer a generic solution to a common issue. Such solutions can then be adapted to other edge paradigms.

## 1.8 Data and task management in IoT systems

An IoT system consists of heterogeneous devices that are capable of performing data collection, computations, storing data, and forwarding messages [43]. Due to the dynamic nature of IoT systems and the mobility of their constituent devices, data management is a great challenge. While data management focuses on handling information effectively, task management ensures that the right actions are taken at the right time using that information. Considering the importance of managing data and tasks efficiently, this would optimize IoT system performance and achieve the highest utilization of fog and cloud processing capability. Since then, various attempts at improving the management of IoT systems have been considered.

### 1.8.1 Data management in IoT systems

IoT data management includes the processes of acquiring, transmitting, storing, processing, and securing data from IoT devices. It's in the sensing phase where these devices collect data from the environment, and the data is sent over protocols. These protocols facilitate reliable and efficient data exchange between devices and central systems [44]. Faster data transfer means less response time and real-time decision-making for real-time IoT applications. Data can be stored temporarily in fog nodes for processing and analysis or sent to the cloud for long-term storage and complex analysis. Proper data management ensures the IoT application works smoothly with security for integrity, availability, and processing speed. Data protection methods like encryption and authentication cover data while being transferred or at rest, while anomaly detection can be used to counter threats to information security.

### 1.8.2 Task management in IoT systems

Task management in IoT systems is concerned with how the computational tasks are assigned, scheduled, and executed at all levels of the IoT architecture. Immediate-response tasks, such as real-time control, are processed at the IoT device, while more complex but latency-tolerant tasks are done at fog nodes. The heaviest computing tasks, such as big data analysis and machine learning model training, are pushed further into the cloud. Effective

scheduling strategies, such as static, dynamic, and priority scheduling, ensure optimum resource utilization. Load balancing techniques, such as distributed load balancing and task offloading, avoid overloading of any device. In addition, fault tolerance strategies such as redundancy and fail over mechanisms ensure maximum resilience for the systems. In [45], the authors introduced a federated reinforcement learning-based framework for adaptive task scheduling in fog computing environments. The work in [46] demonstrates the importance of task management and its impact on QoS parameters such as cost, make span, and energy consumption. Various heuristics and meta-heuristic algorithms like minimum-maximum (Min max), particle swarm optimization (PSO), genetic algorithm (GA), ant colony optimization (ACO), and bees life algorithm (BLA) are evaluated to show the task placement and their impacts in a Fog environment to utilize the limited resources.

## 1.9 Data and task placement in IoT systems

Data and task placement is the placement of data and computational tasks across distributed IoT systems to improve performance, reduce latency, and optimize resource utilization. The placement of data guarantees that it is stored or perhaps replicated close to the processing node, which minimizes the overhead of data transfer and improves access efficiency. Task placement is where computational workloads are assigned to nodes based on processing power, network bandwidth, and data location to balance the load and maximize execution efficiency. The effectiveness of data and task placement together is more important in the areas of cloud, edge, and IoT domains, leading to faster processing, less energy consumption, and better scalability of the IoT system [47]. To deal with these side effects, the authors of [48] presented a novel context-aware framework for data and task placement within fog computing, by separating data from tasks and employing context-based scheduling to enhance performance and efficiency.

### 1.9.1 Data placement in IoT systems

The placement of data refers to the strategy for keeping the data generated by an IoT device in a suitable location for quick access and processing. Different locations have different trade-offs concerning the latency in accessing that data, the storage capacity, and the processing power to deal with that data. This decision involves a trade-off between these three layers that depend on the requirements for the data. Due to the large scale of such systems and their geographically distributed nature, the main concern in [49] is to place data optimally for

processing across the three IoT layers. In addition, the authors of [50] proposed a Fuzzy-PSO framework to optimize the placement of IoT-oriented data in cloud data centers.

The first decision for data placement is whether to store the data directly on the IoT device itself. Such decisions are generally made when the data generated is small, time-critical, or needs immediate processing by local means. By storing the data within the IoT device, it avoids latency, reduces bandwidth consumption, and ensures privacy since no data needs to be transmitted. Such an approach is valuable for real-time monitoring applications, control systems, or fog analytics that require fast decision-making. However, it is limited by the internal storage and computational power of the device, in addition to its energy consumption. For instance, sensors in industrial automation or wearable health devices usually store and process data locally and send only important data to the upper layers (fog or cloud) for analysis. Such balanced efficiency against resource limitations, first of all, forms the step for placing IoT data.

The next decision following the placement of data in an IoT system deals with storing data in the fog layer, which acts as an intermediary between IoT devices and the cloud [51]. A fog computing node, such as a dedicated fog router, gateway, or fog server, is closer to the edge of the network than the cloud and provides more storage and computing resources than individual IoT devices. This better suits applications that may allow for fairly high latency, for example, smart grids, health monitoring applications, or smart city systems, where data needs to be processed quickly but may not need heavy cloud resources. By having a fog layer where data is stored, IoT systems can reduce the burden on the cloud, thereby reducing latency compared to cloud-only solutions while still being able to perform some of the heavier processing tasks that can be done by IoT devices. Fog nodes will also aggregate and pre-process data from multiple IoT devices so that only relevant or summarized information is sent to the cloud for long-term storage or further analysis. This approach allows for a marginal reduction in latency associated with device-based applications and the high scalability associated with cloud computing, thus enabling a variety of IoT applications.

The third decision regarding distributed storage in IoT systems is essentially the use of a cloud environment for data storage because it is rich in resources and scalable. It is not suited for applications where latency is a hard constraint, but mostly requires heavy storage, advanced computing resources, and large-scale data analytics. Examples are long-term data archiving, training machine learning models, and running big data processing for applications like predictive maintenance, climate monitoring, or enterprise-level IoT. This allows an IoT system to indefinitely store huge amounts of data in the cloud and perform

complex computations unachievable by the IoT device or fog layers. On the downside is the increased latency because of the distance where the IoT devices and the cloud servers, and hence their distance from each other is shown. Also, the added bandwidth utilization is due to the data transmitted over. This means that cloud data placement is necessary for centralized administration on a global scale and integration with other cloud-based services.

### 1.9.2 Task placement in IoT systems

Task placement is a vital undertaking in any system, organization, or workflow, involving the moving of various tasks to obtain desired performance, resource utilization, and efficiency. Thus, it becomes a very critical element of workload management, allowing tasks to run by the goals of the system or project. Tasks will be imposed on various aspects of distributed systems and cloud computing project management. When one gives a different task to a different member of the workforce, this leads to improvements in productivity, scalability, and success. In this process, resource availability, dependencies between tasks, system constraints, and performance requirements are analyzed before decisions are made on task allocation.

There are several categories of task placement strategies, each one particularly suited for certain needs and scenarios. These strategies can thereby be categorized on a broad basis along the lines of the style of their operation, elasticity, and decision-making techniques. Below is a complete description of each type of task placement.

#### 1.9.2.1 Static task placement

Static task placement is the allocation of tasks to resources before the start of system execution according to specific rules established through system knowledge or prior information. Static task placement seems to be a simple way to allocate resources, where predictable and stable task requirements and resource availability form the typical environment [52]. In [53], authors proposed a static hardware task placement based on genetic and greedy algorithms to efficiently place a set of tasks before the system starts to work on the multi-context field programmable gate arrays (FPGA) to achieve the best logic capacity utilization. On the other hand, online algorithms make decisions without knowing information about incoming tasks in the future. For example, in a production assembly line, tasks are usually assigned to specific people or machines in a fixed order. Static task placement appears less flexible because it fails to accommodate changes due to resource failures or unexpected changes in workloads.

#### 1.9.2.2 Dynamic task placement

Dynamic task placement, commonly referred to as adaptive task placement, provides a real-time adjustment of task allocation according to the conditions that affect this allocation. This is a very flexible approach in highly dynamic environments where workloads and resource availability are subject to fluctuations [54]. In cloud computing, for example, dynamic task reallocation is implemented to balance loads and address hardware failures. Dynamic task placement is determined by continuous real-time evaluation through appropriate monitoring mechanisms and decision-making algorithms towards optimal performance and resource utilization. Within the federated fog computing environment, Sarkar et al. [55] suggested a deadline-aware dynamic task placement (DDTP) strategy to offload and place the tasks on an appropriate computing fog node. Task management is optimized via the DDTP technique, which ranks tasks according to their due dates and uses a dispatch-constrained policy to redistribute unsuccessful jobs to available fog nodes.

#### 1.9.2.3 Hybrid task placement

Hybrid task placement fuses both static and dynamic paradigms so that their advantages can be enjoyed while their weaknesses can be minimized. In such hybridism, some static resource assignments will be made for some tasks while others will be dynamically assigned as per real-time needs. Very profitable hybridization since some fixed-task requirements are predictable, whereas some are highly variable. Therefore, with the benefits of both the predictability of static placement and the adaptability of dynamic placement, hybrid task placement creates a balanced solution to the heterogeneous requirements of systems. In [56], the authors proposed a modified GA algorithm in fog-cloud computing called Priority–aware GA (PGA) to optimize the multi-objective function that is a weighted sum of overall computation time, energy consumption, and percentage of deadline satisfied tasks (PDST). In [57], a latency time based task assignment combined with online and offline (LTB-TAOO) is proposed in an edge cloud environment. This framework can meet the static and dynamic scenarios of real-world applications.

Task placement directly impacts performance, efficiency, and scalability, and becomes a serious concern in system design and workload management. Static task placement is, for example, less cumbersome and predictable, but inflexible, and works in an environment where conditions are stable. Dynamic task placement is very flexible and enables optimal resource utilization, but it becomes heavy in terms of overhead. That is, it applies to very rapidly changing and unpredictable environments. Therefore, hybrid task placement repre-

sents a middle ground: the best compromise as a method of task placement to accommodate systems that work with mixed workloads with different requirements. The proper selection and implementation of task placement strategy within an organization can lead to the optimization of systems toward the desired performance and efficiency goal [58, 59].

In the next chapters, we will explore these strategies in greater detail, examining their algorithms, implementation challenges, and real-world applications. We will also discuss advanced techniques and emerging trends in task placement, providing a comprehensive understanding of this critical area of system design.

Both static and dynamic task placement mechanisms have their significant roles to play in IoT systems due to their application, variability in the environment, and the resources they want to use. With advancements in edge and fog computing, combined with AI-driven algorithms, task placement continues to evolve, providing smarter solutions for diverse IoT applications. While static placement is best for stable and predictable environments in which its simplicity works well, dynamic placement is ideally applied to situations where flexibility and efficiency are important, such as real-time responsiveness and adaptability. However, in reality, some systems might employ a combination of the two for performance enhancement.

## 1.10 Conclusion

This chapter provided a comprehensive overview of the IoT system, including its definition, uses, advantages, and disadvantages. It also discussed the evolution of the IoT-fog-cloud architecture and the features of data management and related tasks. These technologies have shown great potential for improving productivity, convenience, and overall quality of life, and are increasingly being used in a variety of fields. Despite these advantages, there are still many barriers to IoT adoption, some of which are significant and must be overcome to ensure long-term expansion and widespread implementation. To overcome these limitations, the IoT-fog-cloud model has emerged as a viable hybrid architecture that improves efficiency, scalability, and reliability while facilitating real-time decision-making to meet the growing demand for IoT applications. In this context, task distribution, also known as task offloading, is critical for managing computational workloads, as it determines how tasks are distributed across local devices, fog nodes, and cloud servers to achieve maximum performance. Given its importance, the next chapter of this thesis reviews previous research and methods that address the issue of task distribution in IoT systems, focusing on offloading tactics and how they affect system performance.

# Chapter 2

# IoT Task offloading problem

## 2.1 Introduction

The Internet of Things has revolutionized the way we interact with the world around us, connecting billions of devices and enabling intelligent communication and data management. In recent years, as mobile devices (MDs) and wireless communication systems have advanced significantly, the number of creative applications that require substantial processing power has increased exponentially. Examples include streaming high definition media, Virtual Reality and augmented reality (VR/AR) applications, real-time online 3D gaming, image processing, and face recognition [60], some of which may also be delay-sensitive [61]. These smart MDs are limited in terms of processing power, battery life, and storage capacity. In order to address these limitations, task offloading has emerged as a key technique to improve the performance of these MDs. In addition, offloading enables more powerful external resources (cloud servers, fog nodes) to perform computation-intensive tasks.

Task placement involves determining the optimal location for different workloads and applications within the distributed IoT infrastructure. This decision must take into account a multitude of factors, including resource availability, latency, processing capacity, energy consumption, and even security constraints. In addition, task placement enables more powerful external resources (cloud servers, fog nodes) to perform computation-intensive tasks, improve QoS, and ensure efficient use of available resources while achieving increased processing speed, better energy efficiency, and more advanced application capabilities for the users.

## 2.2 IoT systems: data and tasks

The IoT has revolutionized how devices interact, communicate, and process data, enabling a wide network of interconnected sensors, actuators, and smart devices [49]. As such, data will be produced continuously and huge amounts of data need to be efficiently processed to drive real-time applications such as smart homes, healthcare monitoring, industrial automation, and intelligent transportation systems.

IoT data management and task persistence require extensive planning because they have many distributed architectures, resource-constrained devices, and different network conditions. Management of data in IoT is the collection, storage, transmission, and processing of data across different layers, such as IoT devices, fog, and cloud infrastructures, to minimize latency, bandwidth consumption, and data reliability [62]. Task management, on the other hand, is related to the allocation, scheduling, placement, and execution of various tasks in terms of computational workloads distributed across IoT devices and supporting infrastructures. The management of both data and tasks is therefore essential to realize the best possible utilization of resources, responsive systems, and scalability requirements of IoT applications. However, their complexity, high data ranges, and concurrent dynamic workloads are crucial areas of future research endeavors. This research endeavor is a clear sign, among others, of the requirement for innovative, integrated, artificial intelligence, machine learning, and decentralized approaches to invaluable research areas of high-rate-on-demand, real-time processing.

### 2.2.1 Data in IoT systems

IoT devices constantly generate various types of data by sensing their environment or tracking user activities. This data can include environmental readings (temperature, humidity, motion, location, etc.), location coordinates, motion detection, video streams, etc. It forms the basis for analysis, decision-making automated actions, and service provision across various IoT applications. This data is characterized by high volume, diverse formats (structured, semi-structured, and unstructured), rapid generation speed, varying accuracy levels, and significant potential value. Efficient management of IoT data is essential to minimize latency, enhance system performance and reliability, and ensure real-time services across applications such as smart homes, healthcare, transportation, and industrial automation [63, 64].

### 2.2.2 Tasks in IoT systems

In IoT systems, tasks refer to computational operations or processes performed on data collected by IoT devices to enable intelligent decision-making, automation and service delivery [65]. These tasks can range from simple functions such as filtering, aggregating and transmitting data to complex operations such as data analysis, machine learning inference, anomaly detection and real-time monitoring. Tasks in IoT systems can be executed at different levels - directly on IoT devices, on intermediate fog nodes or in the cloud - depending on factors such as computational complexity, latency requirements, power consumption and network conditions. Efficient task management and placement are keys to ensuring optimal system performance, reducing response times and improving resource utilization in a variety of IoT applications.

## 2.3 The data offloading problem

In IoT systems, data offloading is essential for handling massive amounts of data produced by devices. Also, it is a significant method in optimizing IoT system performance and resource consumption. In the context of IoT, data offloading is the act of moving data from limited IoT devices to more powerful resource, either in the fog nodes, or at the cloud server [66]. This procedure aids in lowering the computational load on Internet of Things devices, preserving their energy, and enhancing their general effectiveness [67].

## 2.4 The task offloading problem

Task offloading refers to the technique of transferring task or part of a task, from IoT devices with resource limited to entities with higher capabilities, like fog or cloud computing infrastructure. It's used in scenarios where the local device lacks the necessary resources, and offloading helps to complete tasks more efficiently. The decision-making process takes into account various factors like network latency, bandwidth, task complexity, and energy consumption. This entity executes the offloaded task and the result is presented back to the IoT device. Offloading strategy can be a form of distributed computing to improve performance, energy efficiency, latency, and utilization of resource through the remote processing capabilities. With the introduction of cloud computing, fog computing and 5G technology, which allows for low-latency communication between devices and external resources, this idea is becoming increasingly popular.

## 2.5 Types of task offloading

There are various methods for task offloading, such as collaborative offloading, partial offloading, and full offloading, each having its own benefits and drawbacks. By putting into practice efficient data offloading techniques, system responsiveness is improved, device battery life is increased, and resource consumption is optimized. Several offloading techniques are employed to optimize performance and resource utilization. The following are examples of various offloading mechanisms [68]:

### 2.5.1 Full offloading

Full offloading, also called total offloading or binary offloading, is mainly focused on indivisible applications. The computation tasks are either run on the MDs or fully offloaded to another powerful server. It refers to the process of moving all task processing from an IoT device to a fog node or cloud server with greater capacity. In this method, all calculations, storage, and analysis are done remotely, with the IoT device only gathering and transmitting data. This method is frequently applied when a high processing power is needed or when the IoT device has restricted computing capabilities. For the binary offloading scenario, the authors of [69] proposed an adaptive task offloading and resource allocation mechanism to meet diverse task demands, using the DRL method to determine whether a task needs to be offloaded or not. The mechanism successfully reduces the average task response time and energy consumption.

### 2.5.2 Partial offloading

In some application scenarios, an application consists of multiple components. Each component represents a subtask that can be offloaded individually. Such offloading is known as partial offloading. The partial offloading is targeted at divisible applications. It manages the application by dividing it into subtasks, and the only choice that the offloading strategy must make is to determine where to offload the subtasks. It manages applications by splitting them up into locally executed subtasks and offloadable subtasks. The term “offloadable subtasks” refers to subtasks that can be offloaded to execute on a fog node or cloud server, and typically include computationally intensive data processing but do not require interaction with the local device. For the partial offloading scenario, [70] proposed an adaptive computation offloading method based on DRL, aiming to minimize the cost that includes energy consumption and data communication latency in vehicular networks.

With the same optimization goal and the same collaboration approach. Since full offloading only requires considering offloading all tasks to a powerful device in the first scenario, it has a lower complexity in step two of the task offloading process [71]. In partial offloading, researchers must investigate which tasks must be offloaded, which must be achieved locally, and where the tasks should be offloaded. The optimization approach for full offloading is often less complicated, but it performs poorly. In [72], the authors proposed a hybrid offloading mode in our algorithm to overcome the drawbacks of a single offloading mode in a complex network. This hybrid strategy reduces latency and energy usage by combining partial and full offloading. They also suggest a hardware classification approach necessary for population initialization. To increase population variety, this algorithm uses low-battery devices as relays when making offloading decisions.

## 2.6 Factors influencing offloading decisions

In order to make the offloading decisions, several aspects need to be taken into consideration. Firstly, we must determine which subtask should be offloaded and which subtask should be executed locally. Secondly, we should find the appropriate location to offload the subtask. Finally, we should find the optimal moment to offload, such as when the amount of computation is large. The following section provides a summary of elements that may affect the offloading decisions.

### 2.6.1 Task characteristics

The decision to offload the task largely depends on the type of task that has to be processed. While highly computation-intensive tasks with lower data volumes are good candidates for processing in fog or cloud server, operations with huge input data may be expensive and sluggish to transfer, discouraging offloading. Furthermore, in order to minimize latency, tasks with strict deadlines or real-time needs are better handled locally or close fog nodes as opposed to cloud servers. Therefore, to ascertain if offloading improves performance, factors including work size, complexity, and time restrictions must be taken into account.

### 2.6.2 Network conditions

In the offloading process, the quality of the communication network is a critical factor. High network bandwidth and low latency facilitate effective data transfer to and from the fog

node or cloud server. However, offloading may result in significant delays or failures in cases of limited capacity, severe network congestion, or irregular connectivity. Therefore, to determine whether offloading is effective or dangerous, network metrics including signal strength, transmission speed, and jitter should be evaluated in real-time.

### 2.6.3 Device capabilities

Because MDs frequently have limited memory, computing power, and battery life, offloading tactics are greatly impacted. The device with low central processing unit (CPU) performance or insufficient random access memory (RAM) may not be able to execute complex tasks locally, prompting the need to transfer to fog nodes or cloud servers to reduce the battery life and computing power. Determining whether to offload tasks is heavily influenced by the capabilities and the energy consumption of the MDs. Offloading may be advantageous for energy-saving devices with short battery lives or those utilized in distant areas.

### 2.6.4 Fog/cloud resource availability

Whether fog and cloud resources can effectively accept and carry out offloaded activities depends on their availability and condition. Offloading might result in lengthy wait times or task rejection if the target node (cloud or fog) is overloaded or receives a lot of requests. The choice may also be influenced by financial considerations like pay-per-use fees for cloud computing services. Additionally, latency is influenced by the physical distance between the device and the processing node, cloud servers that are farther away have slower reaction times than fog nodes that are closer.

### 2.6.5 Application requirement

Application-specific needs also affect task offloading. Applications with strict quality of service requirements, such as real-time responsiveness, low jitter or high reliability, may prioritize low-latency processing via fog nodes or even local execution. Security- and privacy-sensitive applications, such as those dealing with personal health data, may avoid cloud offloading due to the potential for data compromise. Thus, the sensitivity of the application and its performance expectations are vital determinants in offloading strategies.

In conclusion, task offloading decisions are influenced by diverse factors that span technical, contextual, and user-centric aspects. Therefore, in order to make the best choice, an efficient offloading mechanism needs to be flexible, context-aware, and able to assess these

many elements in real-time. These elements must be understood and integrated in order to build intelligent, efficient, and scalable task offloading frameworks that meet the various needs of next-generation smart environments.

## 2.7 Targets of task offloading

Researchers in the field of task offloading consider several offloading targets, which are considered either jointly as a multi-objective problem or individually as illustrated in the figure 2.1. This section will provide a brief explanation of various offloading targets for field task offloading. When solving a task offloading problem, numerous objectives can be applied. These goals are formulated with the use of the objective function, which also provides formal and mathematical guidance for the offloading solution. The offloading targets serve as the objectives. Response time, latency, energy, cost, and bandwidth are among the offloading goals discussed .

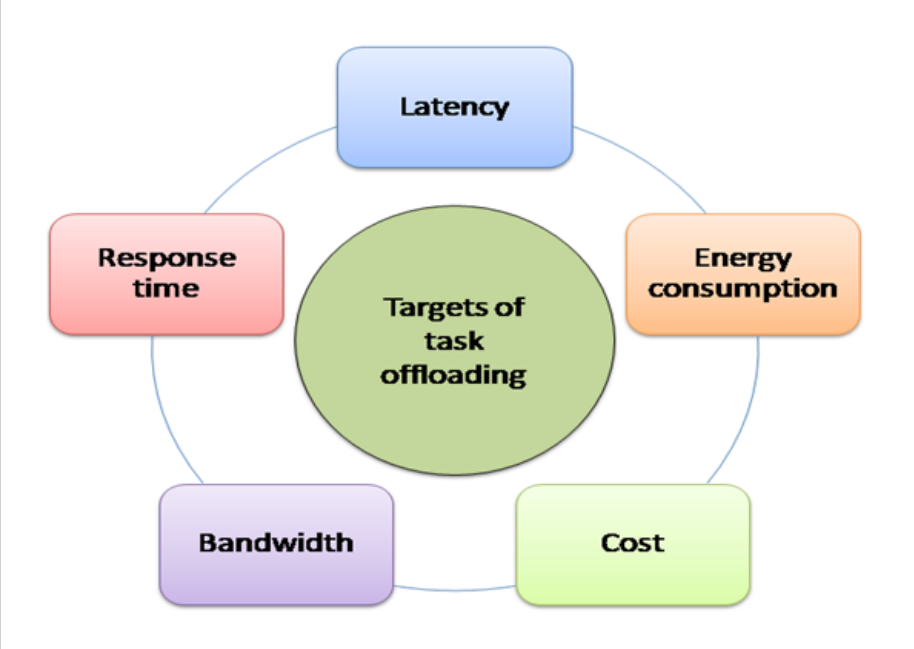


Figure 2.1: Various targets of task offloading

### 2.7.1 Latency

One of the primary goals of task offloading is latency, which is defined as the entire amount of time required to complete a task [73]. The task can be carried out locally at the device, at the fog, or in the cloud. This latency can be broken down into several different factors that

contribute to the latency, namely the transfer of the task to the powerful server, the time it takes to execute the task on the server, and the time it takes to return the results to the device. The total delay of all the tasks included in a mobile application or the minimization of the average delay of each task can be used to represent the delay objective. The available resources and the network circumstances are closely correlated with this goal [74].

### 2.7.2 Energy consumption

The second most common target while offloading tasks is to minimize the energy consumption. The total energy required for offloading includes the energy needed to move the task from the device to another server, the energy required to finish the work at the fog node or cloud server, and the energy needed to send the results back to the device, making up the total energy needed for offloading [75]. Because MDs are typically battery-powered, maximizing battery life by reducing the device's power usage is a major concern. It is certainly reasonable to expect the greatest energy savings to be achieved through the use of full offloading technology. From the perspective of the entire network, it is easy to understand how the issue is transferred to end-to-end or cloud infrastructures.

### 2.7.3 Cost

Cost is a measure of how many resources are needed to send tasks across the transmission medium, run them on the server, and get a satisfactory answer from the source of the request. Especially when using cloud or fog infrastructure, which may include usage-based fees. These costs are influenced by the location of the task, reaction time, task demand, and task energy consumption. Since the consumption of these two related metrics is critical. This means that overall execution costs are similar to the two previously described metrics, which include local, fog, and cloud execution costs, as well as the processing delays. Cost management is essential to preserve economic efficiency, especially in large-scale deployments where operational expenses are a major concern.

### 2.7.4 Bandwidth

Another important constraint is the amount of bandwidth that is available on the access network and how many users may share it to offload tasks from an IoT device to a fog or cloud server. Nonetheless, it may also be regarded as an aim because of its significant impact on task offloading performance [73]. Efficient bandwidth utilization is critical in

environments with limited or shared connectivity resources, where excessive usage can lead to network congestion and degraded system performance. In order to preserve bandwidth availability, offloading solutions must optimize data transport by eliminating unnecessary connections and adjusting to current network circumstances.

### 2.7.5 Response time

Response time measures how long it takes the system to send a task and receive the result. While it concentrates on how users perceive performance, it incorporates all delays (response time). Changes in system processing time and response time, caused by changes in hardware resources or usage, can affect this. In offloading scenarios, the response time represents the interval of time between offloading tasks from local devices to distant servers and receiving the proper response in the designated devices as a performance metric [75]. This target directly affects the QoS and user satisfaction, must enhance the responsiveness of the system, and is often prioritized in real-time and mission-critical applications.

In conclusion, task offloading is a complicated process that is controlled by a number of interrelated performance targets, each one of which captures a critical component of system performance, such as execution speed, resource efficiency, or user experience. However, improving one target frequently leads to trade-offs in others; for example, minimizing latency by using high-performance server resources may increase cost and energy consumption, whereas local processing may result in worse responsiveness and higher delays. Therefore, it is possible to investigate the trade-off between many objectives by using multi-objective solutions. Energy consumption and delay minimization are simultaneously taken into account by the most popular multi-objective techniques [76].

## 2.8 Task offloading strategies

In this section, we divide the discharge techniques used in IoT systems into two groups: machine learning-based algorithms and non-machine learning-based algorithms. Most current reviews focus on machine learning techniques and provide detailed classifications [77], while only a few address non-machine learning approaches. By considering both approaches, this section provides a more comprehensive overview and a clearer picture of their evolution.

### 2.8.1 Machine learning based offloading algorithms

Here, we will present several machine learning-based algorithms that researchers have developed and implemented to solve problems of task offloading. As we know, machine learning is defined as an application of artificial intelligence (AI), which allows systems to learn and improve automatically from experience without being explicitly programmed. Machine learning involves developing computer programs that can access and use data to learn independently [43]. There are three main categories of machine learning algorithms [78].

- Supervised machine learning algorithms:

  These algorithms are based on labeled datasets, where both input features, such as task size, network latency, and device capabilities, and desired output, such as optimal offloading location or resource allocation decision, are known during the training phase. By learning the mapping between inputs and outputs, supervised models can make accurate predictions for new, unseen data. In the context of task offloading, this enables the system to predict whether a task should be processed locally, at the fog node, or in the cloud to achieve goals such as minimizing latency, energy consumption, or cost. Algorithms such as support vector machines (SVM), Decision Trees, Random Forests, Artificial Neural Networks (ANN), and Deep Neural Networks (DNN) are commonly used to classify tasks based on predefined criteria or to regulate optimal execution parameters. However, the quality and diversity of the training data have a considerable impact on its performance, and when operating circumstances change dramatically, retraining can be necessary.

- Unsupervised machine learning algorithms

  This type of algorithm is valuable for task offloading, especially when labeled datasets are limited or unavailable. In contrast to supervised learning, unsupervised methods find hidden patterns, structures, or groups within input data rather than depending on predetermined outputs. Unsupervised learning may be applied in task offloading by grouping tasks or devices according to shared attributes like device capabilities, network latency, workload size, or energy needs. This grouping facilitates the identification of task categories that are more appropriate for local execution, fog processing, or cloud offloading, allowing for more intelligent and flexible offloading choices. For example, clustering algorithms such as K-means or hierarchical clustering can classify incoming tasks based on shared resource profiles. Based on performance limitations and real-time availability, these groups can then be mapped to the appropriate computer resources.

- Reinforcement learning algorithms

  Reinforcement learning algorithms are machine learning algorithms that become effective tools for dynamic and adaptive issues, especially in task offloading. Reinforcement Learning (RL) functions by continuously interacting with the environment and learning the optimal decision via trial and error, in contrast to supervised learning, which trains models using labeled datasets. When it comes to task offloading, RL agents are designed to make choices based on feedback in the form of rewards. Usually, performance measurements like latency, energy use, execution cost, or overall QoS are used to establish these rewards. In order to adapt to changing workloads, varied device capabilities, and varying network circumstances, the RL agent gradually learns a strategy that optimizes long-term cumulative rewards. For example, Q-learning, Deep Q-Network (DQN), Double DQN, Deep Deterministic Policy Gradient (DDPG), and Multi-Agent Reinforcement Learning (MARL) are common RL techniques used in task offloading for more precise and scalable decision-making in complex, real-time systems.

This section will examine the most important applied machine learning algorithms for offloading problem, these algorithms are: SVM, Decision Tree, DNN, K-means, Q-learning, DQN, Double DQN, and DDPG. Figure 2.2 shows the most commonly used machine learning algorithms, classified according to their learning paradigm (supervised, unsupervised, and reinforcement learning)

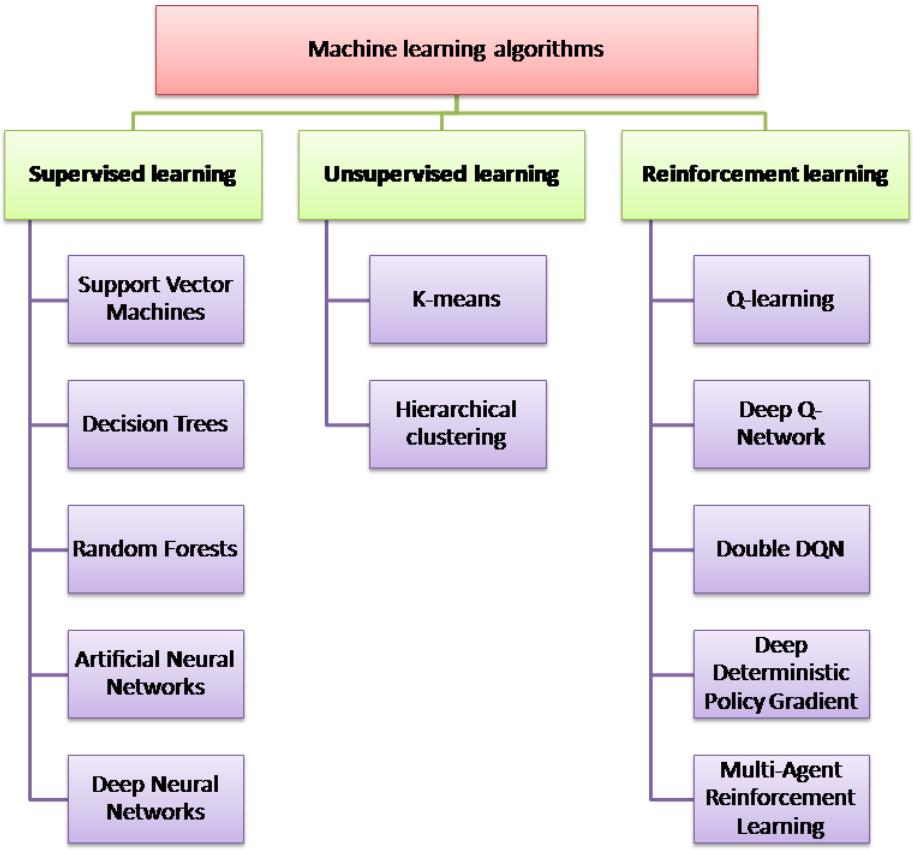


Figure 2.2: Machine learning algorithms

#### 2.8.1.1 SVM-based algorithm:

The SVM [79] is a standard supervised machine learning algorithm that may be used to solve classification and regression issues quickly. Also, it may be used in the offloading decision-making by transforming the problems into a classification problem. It allows for more accurate and speedy offloading progress.

In [79], the authors originally defined the problem as an NP-hard nonlinear programming problem, which was then converted into two sub problems: one based on dynamic programming and the other on convex programming for computing resource allocation. The tradeoff between the quantity of tasks finished and the cost was optimized using the Lyapunov drift-plus-penalty optimization approach based on SVM. Likewise, in [80], they formulated an optimization problem to minimize the energy consumption for task computation and transmission in cellular networks. Then, they proposed an SVM-based federated learning algorithm to determine the task execution location and make the offloading decision.

#### 2.8.1.2 Decision tree –based algorithm

The decision tree [81] is a machine learning model that may achieve high accuracy and optimum performance in various tasks, while it is very easy to understand. Decision trees stand out from other machine learning models due to their clarity of information representation. Task offloading decision trees are useful because they define the issue so that all options may be considered and allow researchers to thoroughly examine the potential outcomes of a choice.

In their contribution, Rego et al. [82] proposed a novel approach based on decision trees and software-defined networking to handle general offloading challenges, in particular, the challenge of determining what metrics to track, when and where to offload. Additionally, the suggested plan facilitates user mobility and offloads the decision-making process. Also, the authors of [81] have suggested a network traffic-aware decentralized task offloading technique that empowers fog nodes to make adaptive offloading choices by utilizing Decision Tree intelligence.

#### 2.8.1.3 Deep learning-based algorithm

DNN is a multi-layered ANN algorithm frequently used for prediction, anomaly detection [83], and optimization problems to determine the optimal answer from poorly trained information by appropriately adjusting mathematics. The number of layers, starting weight, and learning rate are just a few of the factors that need to be carefully considered when utilizing the DNN approach in order to prevent overfitting and computation time [84]. Unlike the time and resource-intensive entities of the offloading ecosystem, these challenges need a significant amount of processing power.

The authors of [85] introduced the a network and device resources utilization through smart ANN-based offloading mechanism (NSANNOM) expert system that is intended for use in heterogeneous edge and cloud computing for efficient network resource allocation mechanisms and optimal computational offloading decision making. The system uses ANN to make accurate decisions, validates real-world datasets, and shows improved latency reduction, cost efficiency, and energy savings. The paper [86] tackles the problem of effectively running DNN on mobile devices with limited resources, particularly when there are several tasks running at once. To address computational and battery life constraints, the authors suggest a combined model partitioning and task offloading strategy.

#### 2.8.1.4 K-means-based algorithm

One of the most popular methods for clustering in unsupervised machine learning is the K-means algorithm [87]. The main principle behind K-means is to divide a set of input data into K clusters. Each cluster is represented by a centroid, which is the mean value of its members. Tasks and devices are then allocated to the cluster that has the closest centroid. Overall, K-means provides a simple, comprehensible method to assist with scalable and clever work offloading techniques.
The authors of [88] suggest using K-means to cluster various workloads into compute-intensive, memory-intensive, and network-intensive applications after classifying them according to their resource requirements, such as CPU, communications, and task priority. Also, [89] proposes a model for joint energy and latency minimization based on K-means to cluster terminal nodes with respect to fog nodes. Based on this, an algorithm called optimal clustering and offloading parameters (OCOP) is developed, which has a lower temporal complexity than the typical quadratic case.

#### 2.8.1.5 Q-learning-based algorithm

Q-learning is an off-policy, model-free reinforcement learning technique that uses the state to decide on the optimal course of action [16]. Learning a policy that instructs an agent on what to do in a given circumstance is the aim of Q-learning. Stochastic transition and reward issues may be handled by Q-learning without requiring adaptation. Additionally, it can assist in resolving optimization decision-making issues by identifying the optimal action-state selection strategy. Thus, offloading issues can be addressed via Q-learning.
In their contribution, [57] propose a Q-learning-based task assignment algorithm for spatial crowd sourcing to address the problem of latency time computation (LTC). The proposed algorithm dynamically receives workers and tasks from the perspective of the full assignment procedure. Moreover, the authors of [90] propose a standard RL algorithm, namely Q-learning. Whether data is handled locally or offloaded to edge devices is the basis for decision-making in an edge computing situation. They accomplished this by using a cost function that included the effects of delay and power usage.

#### 2.8.1.6 DQN-based algorithm

In order to enable agents to learn the best action in a virtual environment to accomplish their goals, Deep reinforcement learning (DRL) combines reinforcement learning with artificial neural networks [1]. One kind of DRL that matches state-action pairs to future rewards

is called deep Q-learning, which combines function approximation with goal optimization. The technique known as deep Q-learning, or DQN, it uses to approximate the Q-value of a reinforcement learning architecture.
In the mobile edge computing (MEC) context, the authors of [16] presented a DQN-based strategy to address the state space expansion curse in the task offloading problem. Then, they introduce a general proximal policy optimization (PPO) strategy to learn the optimal offloading policy in a massive multiple-input multiple-output (MIMO)-based system. Furthermore, Huang et al. [60] introduced a DQN-based method for resource allocation and multiple task offloading. In order to find the optimal solution, they transformed mixed-integer nonlinear programming into an RL problem. They devised cooperative offloading decision-making and distributed bandwidth in order to save total costs. In [91], a DQN-based algorithm was proposed to investigate the situation where the devices are unavailable or cannot meet demand. They divided the complex task into smaller subtasks. Then, based on DQN, they suggested a distributed computation offloading approach, which aims to identify the optimal offloading policy to reduce the time required to complete challenging work.

#### 2.8.1.7 Double DQN-based algorithm

The double DQN is an enhancement of the standard DQN algorithm designed to address the problem of overestimation in action-value functions, which can result in less-than-ideal decisions [92]. Because Double DQN separates the action selection and assessment procedures during Q-value updates, it provides increased accuracy and stability. It employs two neural networks: a target network to assess the value of that action and a target network to choose the optimal action. This separation helps avoid the overoptimistic value estimates that standard DQN frequently generate, particularly in dynamic or complex environments.
In [93], the resource distribution issue in mobile edge computing was examined in a different research. The authors put forth a complex model that efficiently distributes resources using DRL. According to the study, their strategy resulted in a more balanced allocation of system resources, decreased delay, and decreased energy use.

#### 2.8.1.8 DDPG-based algorithm

The DDPG algorithm is an off-policy, model-free method designed for environments that involve continuous action spaces [87]. By integrating the advantages of DQN and policy gradient techniques, DDPG proves particularly effective in solving complex control tasks with high-dimensional action domains.

In [94], authors proposed an Adaptive Computation Offloading model based on RL known as ACORL in a heterogeneous vehicular network, based on DRL, specifically DDPG, aiming to balance between energy consumption and data transfer latency. This model adds noise after the output of the actor network to avoid the complexity caused by a high-dimensional action space. Unlike the above method, Yang et al. [87] proposed DDPG with optimized K-means task offloading, a framework for edge-computing-enabled internet of medical things (IoMT) systems that integrates optimized K-means clustering with a DDPG-based decision strategy with the objective of reducing latency and energy consumption while ensuring load balancing.

In order to train models that can forecast the best offloading choices based on different factors, supervised machine learning for task offloading uses labeled historical data. Supervised machine learning algorithms can provide rapid and accurate predictions once trained, which makes them appropriate for environments with comparatively stable and regular patterns. Although this method is simple to deploy and has a low decision-making latency, it is highly dependent on the caliber and variety of training data. Its primary drawback is its decreased capacity to adjust to extremely dynamic or novel situations, where performance must be maintained by retraining. As a comparison, the table 2.1 shows different cited supervised machine learning approaches with their factors and targets.

Table 2.1: Comparison of research publications based on machine learning.

| **Reference** | **Problem addressed** | **Factors** | **Proposed solution** | **Targets** |
|---|---|---|---|---|
| [60] | Task offloading and resource allocation | Task characteristics, network conditions, device capabilities | DQN-based | Energy consumption, latency, cost |
| [16] | Task offloading | Task characteristics, network conditions | DQN-based | Energy consumption, latency |
| [57] | Task assignment | Task characteristics, device capabilities | Q-learning-based | Latency, Bandwidth |

| | | | | |
|---|---|---|---|---|
| [79] | Task offloading and resource allocation | Task characteristics, the maximum delay. | SVM-based and Lyapunov-based | Energy consumption and cost |
| [85] | Task offloading and resource allocation | Task characteristics, network conditions | ANN-based | Energy consumption, cost, latency. |
| [80] | Task offloading | Task characteristics, device capabilities | SVM-based and federated learning | Energy consumption |
| [82] | Task offloading | Task characteristics, device capabilities, resource availability. | Decision trees and software-defined networking techniques | Energy consumption |
| [81] | Task offloading | Task characteristics, network condition | Decision tree intelligence | Response time |
| [86] | Task partitioning and offloading | Task characteristics, resource availability, network condition | DNN-based | Latency |
| [88] | Task classification and scheduling | Task characteristics, application requirement | K-means-based | Latency |
| [89] | Task offloading | Task characteristics, Fog resource availability | K-means-based | Energy consumption, latency |
| [90] | Resource allocation and task offloading | Task characteristics, device capabilities, application requirement | Q-learning-based | Cost, latency |
| [91] | Task offloading | Task characteristics, network condition, device capabilities. | DQN-based | Response time, latency |

| | | | | |
|---|---|---|---|---|
| [93] | Resource allocation and task offloading | Task characteristics, device capabilities, application requirement | Double DQN-based | Energy consumption, latency |
| [94] | Task offloading | Task characteristics, network condition | DDPG-based | Energy consumption, latency |
| [87] | Task offloading | Task characteristics, Fog/cloud resource availability, application requirement | DDPG and K-means based | Energy consumption, latency |
| [95] | Task offloading | Task characteristics, network condition | DDPG-based | Cost, latency |

Reinforcement learning, especially deep reinforcement learning methods such as DQN and DDPG, is the leading approach to dynamic and complex IoT task offloading problems. These methods are preferred because they can adapt to changing environments (network conditions, resource availability) and optimize multiple objectives simultaneously (latency, energy, cost). Hybrid approaches (such as combining reinforcement learning with supervised learning) are emerging to improve performance. This trend indicates a shift toward deep learning-based solutions for handling high-dimensional state spaces and making real-time decisions in IoT environments.

### 2.8.2 Non Machine learning based offloading algorithms

Non-machine learning algorithms in task offloading are decision-making techniques that, independent of data-driven learning models, choose where and how computational activities are carried out (for example, locally, at fog nodes, or on the cloud). Instead of adapting decisions through historical data analysis, these methods operate on the basis of clearly defined logic, mathematical formulations, or heuristic rules. Non-machine learning methods can offer low-complexity, transparent, and computationally efficient solutions for task offloading and resource allocation by utilizing these established or analytically derived methodologies.

#### 2.8.2.1 Lyapunov optimization

In the field of task offloading, Lyapunov optimization is a mathematical and online control framework used to make real-time, online decisions in dynamic systems such as task offloading in order to reduce performance costs while maintaining system stability [79]. In order to accomplish this, it solves a drift-plus-penalty problem at each decision interval. The penalty term represents the optimization target, while the Lyapunov drift regulates the growth of the queue backlog, allowing for near-optimal judgments without requiring knowledge of future system states.

The study in [96] presents an online dynamic computing offloading algorithm, known as the partial offloading framework using Lyapunov optimization. The suggested approach aims to minimize energy consumption, guarantee system stability, and optimize the offloading of computationally demanding tasks without requiring prior system knowledge. In another work, [97] address the problem of task offloading in mobile edge computing by proposing an online task offloading and resource allocation using Lyapunov optimization theory. The framework aims to reduce the average long-term latency of tasks while maintaining queue stability and energy consumption limitations for mobile devices.

#### 2.8.2.2 Convex and non convex optimization

Convex optimization is a powerful technique for resolving optimization problems since it is solvable [98]. According to this paradigm, the offloading limits are expressed as constraint functions, while the offloading target or objectives are stated as objective functions. The global optimization goal can be achieved by solving the given optimization model using conventional techniques if it is convex. Converting the offloading model into a convex optimization is a common solution if it is a non-convex optimization problem.

In their contribution, Khan et al. [99] propose a framework that addresses the challenges posed by limited computing and communication resources by formulating a cost-minimization problem that jointly considers transmission energy and latency. They formulate a cost-minimization problem that takes into account both transmission energy and latency, and they use a decomposition-based approach to solve it: convex optimization for sensing interval adjustment, matching algorithms for resource allocation and task offloading. Furthermore, the authors of [98] design a distributed massive multiple-input multiple-output (DM-MIMO) aided multi-tier vehicular edge computing system. In order to reduce overall latency and energy consumption, they establish a joint optimization problem for task offloading, sub-channel allocation, and pre-coding using a partial task offloading model. Three convex sub-problems

are extracted from the non-convex problem, and an iterative alternate optimization approach is used to solve them.

#### 2.8.2.3 Game theory

Game theory is a useful tool for creating distributed methods. In order to achieve a mutually agreeable offloading solution, it can be applied in a multi-user offloading situation in which each other end device selects an appropriate technique locally [100].In this paradigm, each end device decides how much to offload, gets incentives, and then updates its choice. This process is carried out repeatedly until the benefits stop getting better. Game theory offers resources for researching equilibrium states, forecasting behavior, and creating plans of action that result in reliable and effective work distribution. Depending on the system scenario, various game models can be applied, such as non-cooperative games, cooperative games, or Stackelberg games.
The work in [101] proposes a task offloading model for binary tasks, which based on game theory algorithm with polynomial time complexity. The model transforms the task offloading challenge into a game of dynamically updating the offloading strategy of the system to determine the optimal offloading strategy. This approach demonstrates the ability to reduce energy consumption, reduce the execution time of computing tasks, and fully utilize system resources. On the other hand, [100] proposes a method of joint resource allocation and task offloading based on Stackelberg game theory. The problem is formulated as a bi-level optimization model with multiple leaders and multiple followers. While users strive to reduce delay, energy consumption, and cost by deciding their task offloading strategies in response to the allocation, edge nodes seek to maximize income and decrease energy cost by choosing how much computing resources to allocate.

#### 2.8.2.4 Particle swarm optimization

PSO is a metaheuristic optimization technique, which can used to find near-optimal solutions for complex optimization problems with imperfect information and computational limitations [50]. Because PSO is a stochastic optimization technique, the result depends on the arbitrary spawn variables. In task offloading, PSO is especially helpful since it can manage non-linear, non-convex, and multi-objective optimization problems, including jointly minimizing cost, energy consumption, and latency while adapting to dynamic IoT, fog, and cloud environments.
In [50], authors propose a PSO-based framework for optimizing IoT data placement in cloud

data centers, addressing challenges related to resource allocation, privacy, energy efficiency, and data access time. Then, a greedy strategy is added to an upgraded PSO algorithm, which searches the complex solution space and chooses the best virtual machine instances for effective data placement. Similarly, authors in [102]introduce a system framework composed of parking clusters, mobile cars, edge servers, and cloud servers in order to handle the problem of task offloading in the internet of vehicles (IoV). They modeled task offloading and resource scheduling as a dual decision problem, and an improved particle swarm optimization strategy (PSOS) is proposed, which uses Non-dominated sorting genetic algorithm II (NSGA-II) to handle multi-objective optimization and the simulated annealing cooling procedure to prevent local optima.

#### 2.8.2.5 Genetic algorithm

GA is a bio-inspired metaheuristic optimization technique that finds near-optimal solutions for complex and non-convex problems by simulating the process of natural selection [56]. However, because it takes multiple generations to produce sufficient outputs, GA may be a viable option to address a variety of optimization issues [53]. But it is only appropriate for a few. However, when fitness measurement is difficult, GA does not produce the best results. In [103], the authors address the task offloading problem in mobile edge computing (MEC), where this challenge requires resource-constrained edge servers to choose between processing tasks locally and offloading them to the internet cloud (IC), which has higher latency but offers more resources. As the problem is NP-complete, a genetic algorithm based metaheuristic is proposed to efficiently obtain near-optimal solutions, which takes user delay constraints into account and enhances decision-making for task execution between MEC and IC. The problem is formulated as an integer linear programming (ILP) model with the goal of minimizing task processing latency. In order to optimize task offloading in a three layer edge computing architecture consisting of edge, fog, and cloud layers, the authors of [104] suggest a hybrid whale genetic algorithm (HWGA) combined with RL. The best layer for task execution is first determined using a multi-layer RL technique, and then the HWGA fine-tunes the task offloading choices within each layer.

#### 2.8.2.6 Heuristic algorithm

A heuristic algorithm is a strategy for solving problems that, when determining the precise best answer is too difficult or time-consuming, finds good, approximate solutions using practical methods or rules of thumb [105]. In task offloading, a heuristic algorithm is an

approximation of a problem-solving technique intended to effectively schedule and distribute work among edge, fog, and cloud resources without ensuring a precise ideal answer. Heuristic algorithms employ rule-based, greedy, or problem-specific methodologies to rapidly produce near-optimal solutions because task offloading problems are frequently NP-hard and computationally costly.

In their contribution, authors of [106] formulated the problem of task offloading as an NP-hard mix integer nonlinear problem and then proposed an effective low complexity heuristic algorithm that yields a near-optimal solution. Their goal was to maximize the number of offloaded tasks for all devices in uplink communication while maintaining low system latency. On the other hand, [105] addresses the challenge of efficient task offloading in vehicular edge computing (VEC) networks, where time-sensitive and resource-intensive activities are produced by vehicular applications. In order to address these needs, the authors use a multi-objective optimization model to describe the offloading problem, taking into account both transmission latency and task completion time as QoS factors. The red deer algorithm (RDA) for global exploration and simulated annealing (SA) for local refinement are combined in their proposed hybrid meta-heuristic algorithm known as red deer simulated annealing (RDSA). As comparison, the following table shows different cited non-supervised machine learning approaches with their factors and targets.

Here, we will present a number of non-machine learning algorithms that have been embraced and applied in both industry and research to address edge computing offloading issues. As illustrated in table 2.2.

Table 2.2: Comparison of research publications based on non-machine learning algorithms.

| **Reference** | **Problem addressed** | **Factors** | **Proposed solution** | **Targets** |
|---|---|---|---|---|
| [50] | Data placement | Task characteristics, Fog/ cloud resource availability | PSO-based | Energy consumption, latency, response time |
| [96] | Task offloading | Task characteristics, device capabilities, application requirement | Lyapunov optimization-based | Energy consumption, bandwidth |

| | | | | |
|---|---|---|---|---|
| [97] | Task offloading and resource allocation Task | characteristics, device capabilities, network conditions | Lyapunov optimization-based | Energy consumption, latency |
| [98] | Task offloading and resource allocation | Task characteristics, device capabilities | Convex optimization-based | Energy consumption, latency |
| [101] | Task offloading | Task characteristics, Fog/ cloud resource availability, application requirement | Game theory-based | Energy consumption, latency |
| [100] | Task offloading and resource allocation | Task characteristics, device capabilities, application requirement | Game theory-based | Energy consumption, latency, cost |
| [102] | Task offloading and resource allocation | Task characteristics, device capabilities, network conditions | PSO-based | Energy consumption, latency, cost |
| [103] | Task offloading | Task characteristics, device capabilities, application requirement | Genetic algorithm | Energy consumption, latency |
| [104] | Task offloading | Task characteristics, device capabilities, network conditions | Genetic algorithm | Energy consumption, latency, response time |
| [106] | Task offloading | Task characteristics, device capabilities, application requirement | Heuristic algorithm | Energy consumption, latency |
| [105] | Task offloading | Task characteristics, device capabilities, network conditions | hybrid meta-heuristic algorithm | Latency, response time |

Because non-machine learning algorithms offer effective and useful decision-making tech-

niques without depending on extensive training data, they are essential to task offloading. Using techniques like game theory, convex and non-convex optimization, heuristic approaches, and evolutionary algorithms like GA or PSO, these methods tackle important issues like resource allocation, latency, and energy consumption in cloud, fog, and IoT environments. Because of their interpretability, reduced computing cost, and efficacy in situations with clearly specified optimization aims and limitations, non-ML algorithms continue to be relevant even though they do not have the flexibility and predictive ability of machine learning models.

## 2.9 Conclusion

. One essential technique for IoT systems is task offloading, which can increase application performance, lower execution latency, reduce energy consumption, and extend battery life of resource constrained devices. Task offloading in practice is influenced by a variety of factors such as network conditions, device heterogeneity, energy constraints, and application-specific requirements. This study highlighted the trade-offs that affect the efficacy of recent developments in task offloading within edge computing systems.Offloading techniques, algorithms, and offloading-influencing elements were all covered in this study's comprehensive analysis of current task offloading in IoT systems. Subsequently, we divided offloading tactics into two groups: full and partial. Additionally, the study looked at the main factors that affect offloading decisions and divided the existing algorithms into two main groups: machine learning-based methods, which make use of predictive intelligence and flexibility, and non-machine learning methods, which depend on optimization, heuristics, and evolutionary algorithms. Finally, various concerns and research challenges related to edge computing were discussed.

# Chapter 3

# DQN-based offloading algorithm

## 3.1 Introduction

IoT has grown rapidly, resulting in the deployment of billions of linked devices that continuously produce massive amounts of latency-sensitive and computation-intensive tasks. Although they provide a wealth of processing power, traditional cloud computing infrastructures frequently fail to meet the demanding latency requirements of these kinds of tasks because of their physical separation from end users. By bringing computational resources closer to IoT devices and facilitating quicker processing, fog and edge computing paradigms have arisen as complementary layers to the cloud in order to overcome this restriction. Task offloading is a crucial approach in this multi-tier architecture that enables devices to shift energy-intensive operations to distant cloud servers or nearby fog nodes, decreasing execution time and increasing system efficiency.

Despite these advantages, creating an efficient task offloading strategy is still a challenging problem. Traditional heuristic and optimization-based approaches are insufficient for real time decision-making due to the dynamic and heterogeneous nature of IoT environments, which are characterized by changeable wireless channel conditions, fluctuating resource availability, and various application needs. Furthermore, adaptive and intelligent decision making processes are needed in such complex situations to achieve an optimal trade-off between latency, energy consumption, and resource usage.

In recent years, RL has attracted substantial interest for tackling decision-making challenges in uncertain and dynamic contexts. By interacting with the environment and getting feedback in the form of incentives, RL allows an agent to learn optimum policies. However, when used in huge state-action spaces, which are typical in IoT–fog–cloud systems, tradi-

tional RL techniques like Q-learning have scaling problems. DRL methods, especially the DQN, have been offered as a solution to this problem. DQN offers a viable paradigm for task offloading optimization and facilitates effective learning in high-dimensional environments by using deep neural networks to approximate Q-values. Accordingly, this chapter presents and explains in detail the contribution of this thesis, which is based on a DQN-driven task offloading strategy tailored to the characteristics and constraints of IoT–fog–cloud architectures.

## 3.2 System architecture

The main objective of this chapter is to present our contribution, which consists of a DQN based task offloading solution for optimizing performance in IoT–fog–cloud systems. In order to accomplish this, a three-layer collaborative architecture is presented, which captures various work contexts and resource interactions across cloud, fog nodes, and IoT devices. Following a thorough description of each layer, the task offloading model is developed, and the objective function that directs the optimization process is created. When combined, these components create the framework for an intelligent and flexible offloading system that improves energy management, scalability, and efficiency in distributed IoT settings.

Establishing a thorough system architecture that reflects the operational features of the IoT-fog–cloud ecosystem is a prerequisite for designing a DQN-based offloading technique. The mathematical basis for explaining the relationships between cloud servers, fog nodes, and IoT devices, as well as the limitations affecting offloading choices, are provided by the system architecture. As shown in Figure 3.1, the device, fog, and cloud layers make up the three-layer collaborative architecture designed to support efficient task offloading and resource management.

- The first layer is the IoT device layer, also known as the infrastructure layer, is made up of mobile devices (MD1-MD6), including smart phones, laptops, smart cameras, smart bulbs, and automobiles. These IoT devices can either process tasks locally or transfer them to higher layers. Although it has the lowest latency, this layer has a limited amount of processing power, making them dependent on offloading mechanisms.

- The second layer is the fog layer, also known as the fog layer, is made up of several FNs (F1, F2, F3, and FM) that are dispersed geographically, which offers computing power and intermediate latency [38]. FNs act as tiny servers that provide localized computing power for both traditional applications. Each IoT device has the ability to transfer

its tasks (T1, T2, T3, etc.) to fog servers, which subsequently provide the calculated outcomes (Response T1, Response T2, Response T3).

- The cloud layer, which is represented by the third layer, has a comparatively greater latency but provides a significant amount of computational capacity and is backed by massive data centers that house a big number of high-performance computers. Both wired and wireless links are used for communication between these layers, and each connection tier is given a specified maximum bandwidth to guarantee effective data transfer. The cloud servers process tasks (e.g., T1, T3) that are forwarded by the fog layer and return the corresponding responses to the IoT devices.

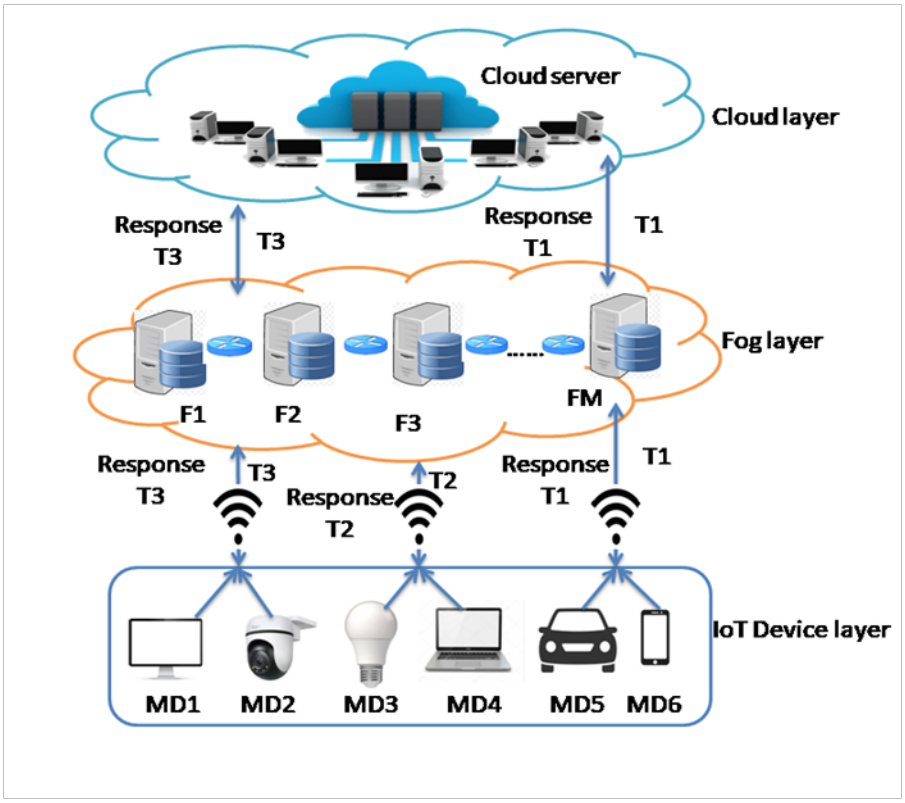


Figure 3.1: Overview of IoT–fog–cloud architecture [1]

The task offloading workflow is captured by this layered architecture in which IoT devices dynamically decide based on latency, energy consumption, and resource availability whether to execute tasks locally, offload them to neighboring fog nodes, or transfer them to the cloud. A summary of the main notations used here is provided in Table 3.1.

Table 3.1: Abbreviations used in the proposed model

| Notation | Description |
|---|---|
| MD | A set of mobile devices |
| N | Number of tasks |
| T | Set of tasks |
| $T_i$ | The task number i |
| $D_{Ti}$ | Task data size |
| $\tau_{Ti}$ | The maximum acceptable delay to execute task $T_i$ |
| CB | CPU cycles required per bit of data |
| $W_{Ti}$ | Total workload of the task $T_i$ |
| F | Set of fog node |
| M | Number of fog nodes |
| C | Cloud server |
| $L_{Ti}$ | Latency(execution time of task $T_i$) |
| $X_{Ti}^{k}$ | Decision offloading matrix of task $T_i$ from user i in the location k |
| $f_d$ | CPU frequency of device d for a processing task |
| Q | The initial latency queue |
| E | Energy consumption of the task |
| $\eta$ | Energy efficiency factor |
| $P_{d,Ti}^{loc}$ | Time processing of task $T_i$ locally |
| $L_{d,Ti}^{loc}$ | The total latency of processing task $T_i$ |
| $E_{d,Ti}^{loc}$ | Energy consumption of task $T_i$ that processing locally |
| $C_{d,Ti}^{loc}$ | Overall cost of task $T_i$ that processing locally |
| $T_{d,Ti}^{f}$ | The transmission time of task $T_i$ to fog layer |
| $P_{d,Ti}^{f}$ | The processing time of task $T_i$ in fog layer |
| $L_{d,Ti}^{f}$ | The total latency of processing task $T_i$ in fog layer |
| $E_{d,Ti}^{f}$ | The energy consumption of task $T_i$ that processing in fog layer |
| $B_{u2f}$ | The MD and fog node communication bandwidth |
| $PW_{u2f}$ | The communication transmission power between MD and fog node |
| $C_{d,Ti}^{f}$ | Cost of processing task $T_i$ over fog layer |
| $T_{d,Ti}^{c}$ | The transmission time of task $T_i$ to cloud server |
| $P_{d,Ti}^{c}$ | The processing time of task $T_i$ in cloud server |
| $L_{d,Ti}^{c}$ | The total latency of processing task $T_i$ in cloud server |

| | |
|---|---|
| $B_{f2c}$ | The communication bandwidth between fog node and cloud server |
| $PW_{f2c}$ | The communication transmission power between the cloud server and fog node |
| $C^{c}_{d,Ti}$ | Cost of processing task $T_i$ over cloud server |

## 3.3 Task offloading model

A precise system model that accurately reflects the nature of the activities and the decision-making process must be established before an offloading strategy can be designed and evaluated. Offloading decisions between local devices, fog nodes, and cloud servers are formulated by the task offloading model, which also gives the mathematical description of computing tasks produced by IoT devices.

### 3.3.1 Task model

In our suggested system, a network of MD is in charge of gathering diverse data that needs to be processed computationally. Each MD generates a set of tasks ($T_i$), where $T_i \epsilon$ [$T_i$..., $T_N$], and each task $T_i$ is characterized by three parameters, which are described as follows:
$D_{Ti}$: the data size of the task Ti.
$\tau_{Ti}$: the maximum tolerable delay for completing the task $T_i$ given in millisecond.
CB: represents the CPU cycles needed per bits of data for task execution.
The relationship between $D_{Ti}$ and CB is as follow:

$$W_{T_i} = D_{T_i} \times CB \tag{3.1}$$

Where $W_{Ti}$ is the total workload of the task ($T_i$).
The location of execution affects energy consumption and execution time of each task. IoT devices that use local execution typically use more energy and may have restricted processing capacity. Because of their close proximity, offloading to a fog server gives lower latency, while offloading to the cloud delivers more processing capabilities but at the cost of additional transmission delay.

### 3.3.2 The offloading decision problem formulation

MD tasks can be offloaded to the selected fog node or executed locally. With the presumption that every MD has an indivisible task and that all task delays equal T [105]. Additionally,

the decision matrix $X_{Ti}^{K}\epsilon$ {0,1} is presented indicating the decision to offload task $T_i$, which will then be carried out on server k thereafter. By balancing latency and energy usage in this situation, it aids in choosing the most suitable server to execute the task efficiently. Specifically, local execution is denoted by ($X_{Ti}^{loc} = 1$), fog node execution by ($X_{Ti}^{f} = 1$), and the cloud execution by ($X_{Ti}^{C} = 1$).

Additionally, each task $T_i$ must have a single execution location, whether local, fog, or cloud. Therefore, we can guarantee these conditions using the following equation:

$$X_{Ti}^{loc} + \sum_{F=1}^{M} X_{Ti}^{f} + X_{Ti}^{c} = 1 \ \forall i\epsilon\{0.....N\}, \tag{3.2}$$

Where local execution is represented by loc and cloud execution by C, and X is a binary variable. Nevertheless, the fog node, represented by F$\epsilon$ [1,..., M], requires that at most one of the following decision matrix be met:

$$X = \begin{cases} X_{Ti}^{loc} = 1 \ \Rightarrow local\, execution \\ \sum_{F=1}^{M} X_{Ti}^{f} = 1 \ \Rightarrow fog\, execution \\ X_{Ti}^{c} = 1 \ \Rightarrow cloud\, execution. \end{cases} \tag{3.3}$$

For example, in the case of local execution, the decision matrix that defines the task placement takes the form:

$$X = \left\{ \begin{array}{l} X_{Ti}^{loc} = 1 \\ X_{Ti}^{f} = 0 \\ X_{Ti}^{c} = 0 \end{array} \right\} \Longrightarrow \text{ local execution} \tag{3.4}$$

## 3.4 Task execution possibilities

Tasks created by IoT devices can be carried out in the suggested system in three different ways, depending on the energy limits, latency requirements, and available resources. This can be a local, fog or cloud execution.

### 3.4.1 Local execution

In this way, the IoT device uses its own computational resources to process the task directly. For resource-intensive applications, this may result in higher energy usage and longer execution times even while it eliminates transmission delays. The local execution is the first option accessible during the offloading process. The local processing time ($P_{d,Ti}^{loc}$) is the amount of

time that a device spends processing the task ($T_i$) from the device (d) without sending it to other devices.

$$P_{d,Ti}^{loc} = \frac{W_{Ti}}{f_d}, \tag{3.5}$$

Where ($f_d$) is the CPU frequency of the device (d) for processing a task, and ($W_{Ti}$) is the total workload of task ($T_i$).

The definition of total latency ($L_{d,Ti}^{loc}$) is as follows:

$$L_{d,Ti}^{loc} = P_{d,Ti}^{loc} + Q_{Ti}, \tag{3.6}$$

Where ($Q_{Ti}$) stands for the initial latency, which is the amount of time that tasks wait in the queue.

Likewise, for local task processing, the following equation defines the energy consumption $E_{d,Ti}^{loc}$:

$$E_{d,Ti}^{loc} = \eta_d \times W_{Ti} \times (f_d)^2, \tag{3.7}$$

Where $W_Ti$ stands for the overall workload of task $T_i$, $f_d$is the CPU frequency for task processing, and $\eta_d$ is the energy efficiency of the device d.

Consequently, the weighted local processing latency ($L_{d,Ti}^{loc}$) and local energy consumption ($E_{d,Ti}^{loc}$) are combined to calculate the overall cost of local processing

$$C_{d,Ti}^{loc} = \alpha \times E_{d,Ti}^{loc} + \beta \times L_{d,Ti}^{loc}, \tag{3.8}$$

It is obtained from equations (3.6) and (3.7), where $\alpha$ and $\beta$ are the energy weights and latency with values ranging from zero to one [2]. The weights are set as follows to guarantee equal weighting of the two goals: $\alpha = 0.18$ and $\beta = 0.82$. These weights are used to calculate the relative importance of each component in the overall cost.

### 3.4.2 Fog execution

The work is transferred to a nearby fog node, which has less latency than the cloud and more processing capacity than IoT devices. Applications that need quick responses and are delay-sensitive can use this option. It is necessary to offload a number of tasks to the fog nodes due to the restricted capabilities of MD. Transmission time ($L_{d,Ti}^{f}$)) and processing time ($P_{d,Ti}^{f}$) make up the two components of the total task delay. This formula is used to determine the transmission time ($T_{d,Ti}^{f}$) in the equation (3.9), and the same formula used for local processing is used to get the processing time ($P_{d,Ti}^{f}$) on the fog node like in equation(3.10):

$$T_{d,Ti}^{f} = \frac{D_{Ti}}{B_{u2f}}, \tag{3.9}$$

Where $B_{u2f}$ is the bandwidth available for data transfer between the MD and fog node, and $D_{Ti}$ is the size of the task to be performed by the fog node.

In the same manner as local processing, the processing time ($P^f_{d,Ti}$) on the fog node is computed using Equation (3.10):

$$P^f_{d,Ti} = \frac{W_{Ti}}{f_j} \ \forall j \epsilon [1 \dots M], \tag{3.10}$$

The CPU frequency of the fog node (j) is represented by the variable $f_j$.

The following definition applies to the overall latency ($L^f_{d,Ti}$) of the processing task (T$i$) of the device (d) in the fog layer:

$$L^f_{d,Ti} = P^f_{d,Ti} + T^f_{d,Ti} + Q^f_{Ti}, \tag{3.11}$$

Where the time tasks spend in the queue over the fog node (j) is denoted by $Q^f_{Ti}$.

Additionally, Equation (3.12) is used to determine the energy consumption ($E^f_{d,Ti}$) via the fog node:

$$E^f_{d,Ti} = PW_{u2f} \times L^f_{d,Ti} + \eta_j \times W_{Ti} \times f_j \ \forall j \epsilon [1 \dots\dwarf M], \tag{3.12}$$

Where ($PW_{u2f}$) is the communication transmission power between the MD and the fog node. $W_{Ti}$ is the total workload of task ($T_i$), and ($\eta_j$) and ($f_j$) are the energy efficiency factor and CPU frequency of the fog node j, respectively.

The total offloading cost over the fog node is calculated using equation (3.13).

$$C^f_{d,Ti} = \alpha \times E^f_{d,Ti} + \beta \times L^f_{d,Ti}. \tag{3.13}$$

### 3.4.3 Cloud execution

The task is delegated to distant cloud servers, which offer a wealth of storage and computational capacity. The cloud can effectively manage large-scale and complicated operations, but its lengthy transmission distance adds energy consumption and delay, making it less appropriate for real-time applications. When, tasks are processed in cloud data centers, the propagation time is increased due to the geographical distance between the task and the resources.

Combining two factors, transmission time ($T^c_{d,Ti}$), and processing time ($P^c_{d,Ti}$), represents the task's total delay ($L^c_{d,Ti}$). The following is an explanation:

$$T^c_{d,Ti} = \frac{D_{Ti}}{B_{u2f}} + \frac{D_{Ti}}{B_{f2c}}, \tag{3.14}$$

$$P^c_{d,Ti} = \frac{W_{Ti}}{f_c} + Q^c_{Ti}, \tag{3.15}$$

$$L^c_{d,Ti} = T^c_{d,Ti} + P^c_{d,Ti}, \tag{3.16}$$

Where $f_c$ is the CPU frequency of the cloud server, $Q^c_{Ti}$ is the initial latency referring to the cloud server, $W_{Ti}$ is the total workload of task $T_i$, $D_{Ti}$ is the data size of the task to be processed by the cloud server, and $B_{f2c}$ is the bandwidth available for data transmission between the fog node and the cloud server.

The energy consumption of the cloud server is determined using equation (3.17):

$$E^c_{d,Ti} = PW_{u2f} \times L^c_{d,Ti} + PW_{f2c} \times L^c_{d,Ti} + \eta_c \times W_{Ti} \times f_c, \tag{3.17}$$

Where ($f_c$) and ($\eta_c$) represent the energy efficiency factor and CPU frequency of the cloud server, and ($PW_{f2c}$) and ($PW_{u2f}$) represent the communication transmission power between the cloud server and fog node, and MD and fog node, respectively.

The following formula is used to determine the overall offloading cost over the cloud server:

$$C^c_{d,Ti} = \alpha \times E^c_{d,Ti} + \beta \times L^c_{d,Ti}. \tag{3.18}$$

The total cost of offloading is the sum of the execution latency and energy consumed by IoT devices and devices across fog and cloud servers [105]. This is how it is computed:

$$cost = \sum_{i=1}^{N} (X^{loc}_{Ti} \times C^{loc}_{d,Ti} + X_(Ti)^f \times C^f_{d,Ti} + X^c_{Ti} \times C^c_{d,Ti}). \tag{3.19}$$

## 3.5 Problem formulation

In the IoT– fog – cloud scenario under consideration, a set of user devices generate activities that require intensive computing that must be achieved locally, in a nearby fog node, or on remote cloud servers. Each task is characterized by the required latency, input data size, and computational load. The main objective of the problem is to design an offloading approach that balances user quality of service goals and minimizes the overall cost of the system, which is typically measured in terms of task execution time and energy consumption. However, the diverse nature of resources, the random nature of wireless channels and task arrival, and the correlation between communication and processing resources make the offloading decision problem inherently difficult. As a result, the optimization problem becomes high-dimensional, dynamic, and non-convex, rendering traditional optimization methods insufficient for obtaining efficient and timely solutions. We reformulate the task offloading

procedure as a sequential decision-making problem, which can be successfully solved through reinforcement learning, specifically DQN technology, in order to overcome these difficulties.

Our DQN-based solution continuously learns from the environment to iteratively enhance offloading strategies, in contrast to many previous approaches that rely on static allocation or basic heuristics. Furthermore, our method cooperatively optimizes resource allocation across local, fog, and cloud layers, which set it apart from other DRL based frameworks that usually concentrate on fog–cloud cooperation or assume simplified network models. Practical limitations include task deadlines; energy budgets for cloud data transmission, bandwidth availability between MD and fog nodes increased by IoT networks, and queue stability at each layer are all clearly incorporated. By taking these practical factors into account, the suggested approach outperforms traditional offloading systems by achieving significant reductions in system latency and energy usage while maintaining stringent QoS compliance. So, an objective function must be defined under some constraints and basing on cited parameters.

### 3.5.1 Objective function

The main goal of our task offloading model is to balance energy consumption and compute latency in order to maximize system performance. However, the task offloading process's cost reduction challenge is expressed mathematically as follows:

$$Min(cost) = min \sum_{i=1}^{N} (X_{Ti}^{loc} \times C_{d,Ti}^{loc} + \sum_{F=1}^{M} (X_{Ti}^{f} \times C_{d,Ti}^{f}) + X_{Ti}^{c} \times C_{d,Ti}^{c}), \tag{3.20}$$

Where X is a binary variable and loc and C represent local execution and cloud execution, respectively. However, F$\epsilon[1, ..., M]$ denotes the fog node. $C_{d,Ti}^{loc}$, $C_{d,Ti}^{f}$, $C_{d,Ti}^{c}$ denotes the overall cost of task $T_i$ that processing locally, over fog layer, over cloud server .

### 3.5.2 Contraints

To guarantee the viability and accuracy of the developed task offloading model, a number of limitations need to be established. These limitations ensure that every activity is performed in a valid and consistent way and define the acceptable values of the decision variables. Additionally, they make sure that the decision about offloading adheres to the fundamental needs of the system, including whether a task is processed locally, at the fog layer, or in the cloud. The primary limitations taken into account in our work are explained as follows:

#### 3.5.2.1 Binary variable constraint

$$C1 : X_{Ti}^{loc} \epsilon \{0, 1\} \tag{3.21}$$

$$C2 : X_{Ti}^{f} \epsilon \{0, 1\} \tag{3.22}$$

$$C3 : X_{Ti}^{c} \epsilon \{0, 1\} \tag{3.23}$$

C1, C2, and C3 denote that the decision variables $X_{Ti}^{loc}$, $X_{Ti}^{f}$, and $X_{Ti}^{c}$are binary, ensuring that each task Ti can only take a value of either 0 or 1, thereby indicating whether it is executed locally, on the fog server, or on the cloud server, respectively.

These constraints (C1, C2, C3) ensure that each task is executed either locally, at the fog layer, or at the cloud layer, but not simultaneously across multiple locations which can be written in C4 constraint as:

$$C4 : X_{Ti}^{loc} + X_{Ti}^{f} + X_{Ti}^{c} = 1 \ \ \forall i \epsilon N \tag{3.24}$$

#### 3.5.2.2 Bandwidth constraint

For the bandwidth, we give the C5 constraint as follow:

$$B_{Ti} > 0 \ \ \forall i \epsilon N \tag{3.25}$$

It makes sure that each activity has a precisely positive bandwidth allotment. Put differently, the allotted bandwidth $B_{Ti}$ must satisfy $B_{Ti}>0$ for each task $T_i$. This condition prevents transmission failures and ensures that data offloading is possible by assuring that no task is left without transmission capability. By enforcing this constraint, the system avoids unrealistic scenarios where tasks are assigned zero or negative bandwidth, which would make offloading impossible.

#### 3.5.2.3 Latency constraint

Also, for the latency parameter, we write C6 as:

$$C6 : X_{Ti}^{loc} \times L_{d,Ti}^{loc} + X_{Ti}^{f} \times L_{d,Ti}^{f} + X_{Ti}^{c} * L_{d,Ti}^{c} <= \tau_{Ti} \ \forall i \epsilon N \tag{3.26}$$

This constraint guarantees that task $T_i$ overall execution latency won't surpass its deadline $\tau_{Ti}$ local, fog, or cloud are the only three possible decision variables. The associated latency is taken into consideration based on the decision. The total of the chosen execution latency must be less than or equal to the maximum (deadline) that task $T_i$ can with stand. And so on.

## 3.6 Proposed solution

This section presents the proposed task offloading model, which aims to effectively manage computing allocation across the IoT – fog – cloud layers. The proposed framework combines two different approaches: a learning strategy based on a DQN and an optimal offloading strategy. Through a comprehensive analysis of every conceivable offloading option local execution on an IoT device, offloading to a fog node or delegation to a cloud server the first technique, known as the optimal strategy, seeks to determine the ideal execution location for each activity. To minimize the overall cost of the system, this method examines system factors such as execution latency, power consumption, and network latency. It generates a theoretical benchmark representing the most efficient configuration for a given system state by examining every possible combination of task placement options.

Although this optimal offloading approach guarantees excellent choices and provides a theoretical benchmark for performance evaluation, it suffers from serious drawbacks in real-world applications. Specifically, this method requires continuous collection of large amounts of real-time data related to system parameters, such as server availability, network conditions, and hardware resources. Deploying such comprehensive and dynamic monitoring in large-scale IoT settings in the real world is computationally expensive and often impractical. Furthermore, when dealing with a large number of diverse devices and tasks, the computational complexities resulting from a comprehensive search approach can cause delays and scalability issues.

On the other hand, in order to obtain near-optimal solutions, we applying RL techniques, most especially the DQN model. This method stores action-state values in a Q-table, allowing the agent to learn from experience. After monitoring the current state $s_t$ at each time step t, the agent chooses an action $a_t$ and transitions to the next state $s_{t+1}$, earning a reward $r_{t+1}$ in the process [92]. Based on accumulated experience, the agent continuously improves its decision-making policy through this iterative process, as shown in figure 3.2.

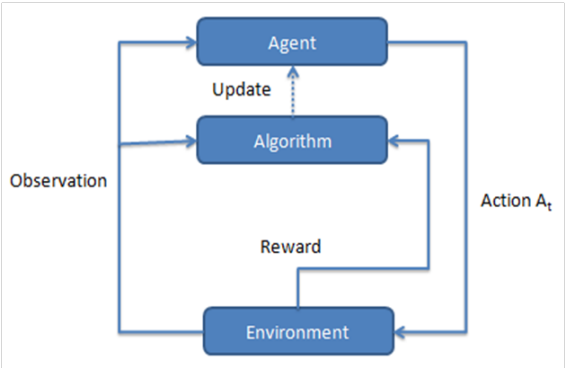


Figure 3.2: The interaction model of RL

To better understand the proposed approach, Figure 3.3 illustrates the workflow of the two task offloading approaches used in this study: the DQN-based learning approach and the optimal strategy. The DQN-based algorithm, shown on the right side of the diagram, uses deep reinforcement learning to determine dynamically efficient offloading policies through continuous interaction with the environment. The optimal strategy comprehensively searches for the best offloading decision by evaluating all possible execution locations local, fog, and cloud based on the specified objective function. This comparative diagram illustrates the conceptual differences between deterministic optimization and learning-based decision-making, highlighting the adaptability and scalability of the DQN framework in a dynamic IoT–fog–cloud environment.

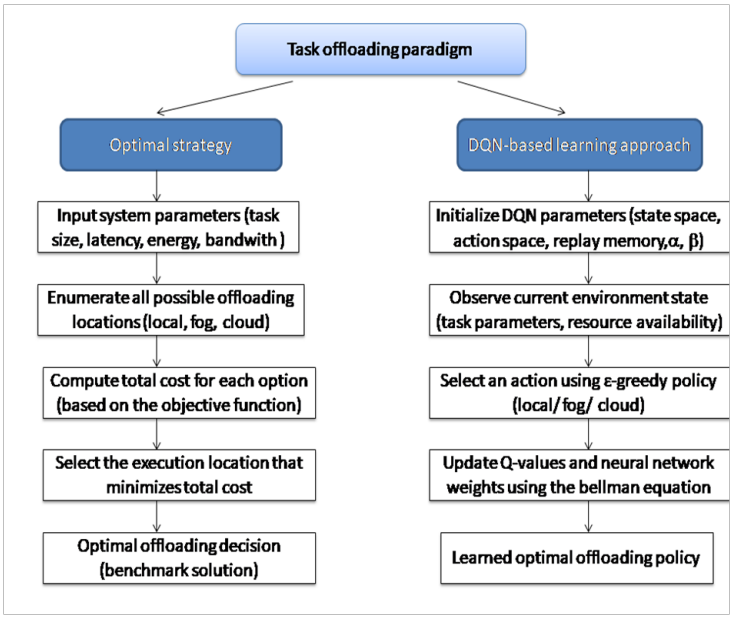


Figure 3.3: Flowchart of the proposed DQN-based and optimal offloading strategies

### 3.6.1 Optimal task offloading solution

In the context of IoT environments, this section presents a task offloading model that aims to determine the best place to execute tasks generated by MD. This approach uses an algorithm to address the optimization problem, resulting in suboptimal solutions with low processing costs, as mentioned earlier.

The process begins when a device generates a task as input, and the algorithm subsequently gathers data from the MD, FNs, and cloud servers. The algorithm takes into account the features of the underlying communication infrastructure for each task, paying special attention to the high-capacity, low-latency benefits offered by IoT networks when tasks are offloaded to distant fog nodes or the cloud. The lowest calculated cost is then used to choose the execution location: Values between 1 and 10 indicate execution at one of the fog nodes (assuming a total of 10 fog nodes), whereas (0) indicates local execution and (-1)indicates cloud execution. The value denotes the particular fog node in charge of carrying out the task via the most economical execution path when the chosen location deviates from (0) and (-1). This is usually impacted by the availability of increased bandwidth made possible by network links and the deployment of IoT components. The process begins when a device creates a task as input, and then the algorithm collects data from MD, fog nodes, and cloud servers. The algorithm takes into account the characteristics of the underlying communications infrastructure for each task, paying particular attention to the high capacity and low latency advantages offered by IoT networks when loading tasks onto remote fog nodes or the cloud. The calculated lowest cost is then used to select the execution location. For illustration: values between 1 and 10 indicate execution on one of the fog nodes (assuming a total of 10 fog nodes), while the value of (0) indicates local execution and (-1) indicates cloud execution. The value indicates the specific fog node responsible for executing the task via the most economical execution path when the selected location deviates from (0) and (-1). This is typically influenced by the increased bandwidth availability provided by network links and the deployment of IoT components [107].

For every task that is generated, the algorithm chooses the processing location by calculating the cost of each execution option local, fog, and cloud and choosing the one that minimizes expenses. The algorithm determines the processing location for each generated task by evaluating the cost associated with all possible execution alternatives local, fog, and cloud and selecting the option that minimizes this cost. Algorithm 1 provides a detailed description of the complete procedure.

**Algorithm 1** Optimal task offloading strategy

**Input** Mobile_devices MD, Fog_node F, Cloud_server C, Tasks T.

**Output** Execution_Location EL, Min_Cost MC.

**Function** Optimal Strategy:
**for** Each d in MD **do**
  **for** Each Task in T **do**
    Initialiser(EL, MC)
    //Local Execution
    MC= $C^{loc}_{d,Ti}$ //Using Equation (3.8).
    EL = 0
    // Fog Execution
  **for** Each Fog_node in F **do**
    Compute $C^{f}_{d,Ti}$ //Using Equation(3.13)
    **if** ($C^{f}_{d,Ti}$ < MC) **then**
      MC = $C^{f}_{d,Ti}$
      EL= Fog_Node
    **end if**
  **end for**
    //Cloud Execution
    Compute $C^{c}_{d,Ti}$ //Using Equation (3.18)
  **if** ($C^{c}_{d,Ti}$ < MC) **then**
      MC = $C^{c}_{d,Ti}$
      EL = -1
      Compute cost(d, $T_i$) //Using Equation (3.19)
  **end if**
  **end for**
**end for**

### 3.6.2 DQN-based task offloading

This section introduces the DQN-based task offloading technique, a RL algorithm created to tackle complicated and high-dimensional decision-making problems. Finding the best place for each activity to be executed, whether at the local device, fog node, or cloud server, while balancing latency, energy consumption, and resource usage, is the goal in the context of IoT–fog–cloud systems.

The offloading controller, which acts as the agent in the DQN architecture, continuously monitors the state of the system, including the amount of available computational resources, task sizes, bandwidth conditions, and device power levels. The agent chooses an action, which is equivalent to deciding where to offload the work, based on the observed state. The system changes states when the action is completed, and the agent is rewarded based on the quality of the decision (e.g., lower latency and energy cost).

#### 3.6.2.1 Key Components of DQN-Based Offloading

Unlike traditional Q-learning, which uses a Q-table, DQN approximates the action-value function using a deep neural network. In settings with large, continuous spaces, this allows the model to scale effectively. DQN uses replay to adjust the parameters of the neural network at each iteration. During training, random samples of past transitions (state, action, reward, and next state) are taken from a buffer memory. This system stabilizes learning by preventing correlation between training samples. To further enhance convergence, a target network is used in addition to the base network.

- The state space The state space is a representation of all the important data needed to decide when to offload tasks. It includes several characteristics that have a direct impact on system performance, such as the state of IoT devices at the moment, the processing power of fog and cloud nodes, the size and complexity of the task to be processed, and the state of network latency. In order to reflect the changing nature of resource availability, the state space also takes into account the current workload or load level of the system. The state space can capture these characteristics and provide the DQN agent with a comprehensive picture of the environment, allowing it to assess different offloading strategies and adjust its decisions in response to evolving system conditions. Formally, the state at time t can be expressed as follows:

$$s_t = \{W_{Ti}, D_{Ti}, Q_{Ti}, f_d, f_j, f_c, B_u 2f, Bf2c\}, \tag{3.27}$$

Table 3.2 summarizes the parameters that define the state space in Equation 3.27, including task characteristics, computational resources, and network bandwidth information at each layer of the IoT–fog–cloud environment.

Table 3.2: Description of parameters in the state space representation

| Parametres | Bref explication |
|---|---|
| $W_{Ti}$ | The total workload of the task $T_i$ |
| $D_T i$ | The data size of the task $T_i$ |
| $Q_{Ti}$ | The time spent waiting of task $T_i$ in the queue |
| $F_d$ | The CPU frequency of the device (d) |
| $F_j$ | The CPU frequency of fog node (j) |
| $F_c$ | The CPU frequency of the cloud server |
| $B_{u2f}$ | The available bandwidth for data transmission between the MD and the fog node |
| $B_{f2c}$ | The available bandwidth for data transmission between the fog node and the cloud |

- The action space Action space is a set of all conceivable actions that the agent can perform in the environment, which directly affects where and how tasks are carried out. In the context of task offloading, the decision to offload each task at a specific time step t is represented by the action space. This choice essentially determines whether the task should be processed locally on the IoT device, offloaded to the fog layer, or moved to the cloud server. Each option has advantages and disadvantages: fog offloading provides a balance between response time and computation but has limited resources; cloud offloading provides abundant resources but causes higher transmission delays; and local execution reduces communication delays but may overload device resources. Thus, the action space can be formally expressed as:

$$a_t = \{-1, 0, 1 \dots M\}, \tag{3.28}$$

Where $a_t$= 0 denotes local processing, $a_t$ = -1 denotes cloud processing, and $a_t$= [1 ...M] denotes fog node execution (assuming 10 fog nodes, so M = 10) [105]. As a result, the action space has the following formal expression: Table 3.3 summarizes all possible actions that the agent can take during the task offloading process. Each action corresponds to a specific execution location local device, one of the available fog nodes,

or the cloud server allowing the agent to explore and learn optimal offloading policies across different computing layers.

Table 3.3: Description of possible actions in the action space

| Values of $a_t$ | Execution location | Description |
|---|---|---|
| -1 | Cloud server | Task is offloaded to the cloud for execution |
| 0 | Local device | Task is processed locally on the IoT device |
| 1 | Fog node1 | Task is offloaded to Fog Node 1 |
| 2 | Fog node2 | Task is offloaded to Fog Node 2 |
| 10 | Fog node 10 | Task is offloaded to Fog Node 10 |

- The reward Specifically, the reward reflects the performance of the system in terms of latency and energy consumption, which are the two main optimization objectives in this work. Since lower latency and reduced energy usage correspond to better system performance, the reward function is modeled using negative values, which penalizes suboptimal offloading decisions and encourages the agent to seek policies that minimize overall cost. The reward function is a crucial component in guiding the learning process of the DQN-based offloading algorithm. It is carefully designed to be consistent with the objective function of the system model, which aims to minimize the total cost associated with task execution. It is carefully designed to be consistent with the objective function of the system model, which aims to minimize the total cost associated with task execution.

  This reward structure allows the DQN network to repeatedly improve its policy by linking high-value decisions to positive system outcomes, such that in practice, if a task is performed in a way that causes a higher delay or energy consumption, the corresponding negative reward discourages the agent from repeating such actions. When the decision to unload the task minimizes the objective function, the agent receives the maximum reward, indicating an optimal or near-optimal decision

$$r_t(s_t, a_t) = -(\alpha \times L + \beta \times E), \tag{3.29}$$

  Where the normalized delay and energy are measured by the weighting coefficients $\alpha$ and $\beta$, which properly balance several objectives and combine them into a single reward function.

Table 3.4: Example of reward values and their interpretation in the DQN-based offloading model

| Scenario | Value of L | Value of E | Value of $\alpha$ | Value of $\beta$ | Value of reward |
|---|---|---|---|---|---|
| Low latency, low energy | 0.2 | 0.3 | 0.18 | 0.82 | -0.29 |
| Moderate latency, moderate energy | 0.5 | 0.6 | 0.18 | 0.82 | -0.75 |
| High latency, low energy | 0.8 | 0.3 | 0.18 | 0.82 | -0.42 |
| Low latency, high energy | 0.2 | 0.8 | 0.18 | 0.82 | -0.70 |
| High latency, high energy | 0.8 | 0.9 | 0.18 | 0.82 | -0.88 |

Table 3.4 illustrates how the DQN agent is inherently directed to take actions that minimize energy consumption and delay, gradually enhancing its offloading strategy through reward-based learning over a number of iterations.

#### 3.6.2.2 Weighting Factor Analysis

Based on the findings of our preliminary experiments, we systematically evaluated several combinations of the weighting parameters $\alpha$ and $\beta$ in order to capture their impact on system performance. We discovered through this approach that the best balance between communication and computation expenses is achieved when $\alpha = 0.18$ and $\beta = 0.82$. In addition to lowering the overall system expense, this arrangement guarantees that task deadlines are regularly met, enhancing system dependability and service quality.

The importance of parameters in the decision-making process is further highlighted by parameter sensitivity analysis. More specifically, the higher value of $\alpha$ emphasizes the local calculation component of the objective function. This reduces network traffic and dependence on external resources, but it can also significantly increase device-level energy consumption, which is undesirable for IoT devices with limited resources. Conversely, increasing the value of $\beta$ prioritizes the use of cloud or fog resources to complete tasks. This approach reduces the energy consumption of local devices, but also increases dependence on network conditions, communication costs, and transmission delays.

#### 3.6.2.3 Training and Execution Process of DQN Agent

Based on the current state of the system, the DQN model uses a neural network to approximate the Q-value function, which evaluates short-term execution costs as well as long-term benefits of offloading actions. DQN can overcome the shortcomings of traditional Q-learning thanks to these capabilities, especially in dynamic, high-dimensional IoT, fog, and cloud contexts. At each decision step, the model decides whether to execute the task locally, in the fog, or in the cloud. This continuous interaction with the environment is how the model is taught. Task completion time, energy usage, and overall QoS are examples of system performance indicators used to evaluate actions. Through this iterative process, the DQN agent successfully learns how to balance response time and energy efficiency, thereby improving its policy. This adaptability makes DQN especially effective in environments with fluctuating network conditions, heterogeneous device capabilities, and varying workload demands scenarios where static, rule-based strategies often fail.

The agent begins the decision-making process by monitoring the state of the environment, which records task-level characteristics such as size, computing requirements, and device power constraints, as well as infrastructure metrics such as network response time, bandwidth conditions, fog and cloud availability, and more. The agent uses its learned Q-function to determine whether offloading actions local processing, fog execution, or cloud offloading will maximize the expected cumulative reward based on these observations. When an action is executed, the environment immediately responds with a reward or penalty representing the resulting performance outcomes, such as execution delay and energy consumption. The DQN agent continuously improves its Q-value estimates using this feedback, resulting in a policy that considers the long-term efficiency of future offloading decisions in addition to optimizing the current decision. With the help of reinforcement learning, the agent can achieve continuously economical and performance-conscious offloading techniques, dynamically adapting to real-time changes in the IoT–fog–cloud system.

The agent uses temporal difference learning to change its policy ($\pi$) based on the received reward and the updated system state, successfully balancing short-term costs and long-term rewards. By use of recurrent contacts (seen in Figure 3.4 by the cyclic arrows). The agent gradually improves its decision-making skills until it can execute context-aware, adaptive offloading. Figure 3.4 shows a summary of the agent using the suggested DQN technique.

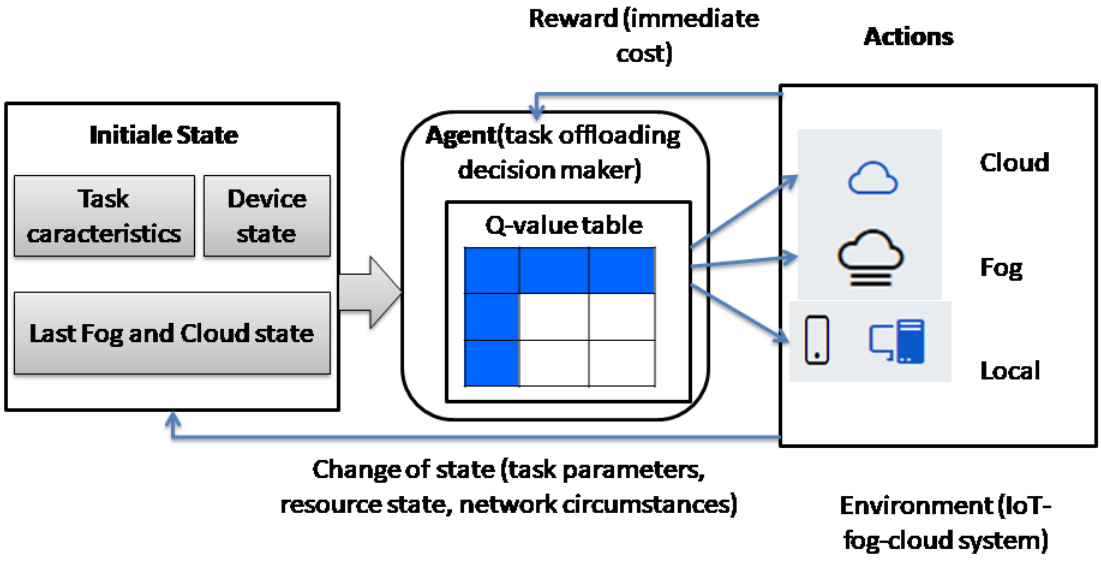


Figure 3.4: The process of task offloading decision of RL algorithms [1]

The main goal of the Q-learning framework [1] is to minimize cumulative system costs or, equivalently, maximize long-term benefits. The agent interacts with the environment by choosing actions and monitoring the ensuing state transitions and rewards. The agent gradually improves its approach through trial-and-error without needing to know the dynamics of the environment beforehand by assessing how well its action reduces metrics like latency and energy usage at each stage. In its most basic form, Q-learning keeps track of the predicted cumulative reward for every state–action pair in a Q-table [108]. This Q-table is updated repeatedly using the Bellman equation by taking into account both the estimated future values of following states and immediate rewards. As the learning process is enhanced by the series of visited states (s1, s2,...) and accompanying actions (a1,a2,...), the DQN strategy enables the agent to gradually converge on an optimal strategy. Although this tabular method is useful for small-scale problems, it is impractical for high-dimensional IoT–fog–cloud systems. For this reason, deep neural networks, such as DQN, are being integrated to approximate the Q-value function and allow for scalability in complex environments. As a result, the agent will select the action that, under a policy ($\pi$), moves the state from ($s_t$) to state ($s_{t+1}$) at each step, after which it will be rewarded ($r_t$). Additionally, the status ($s_t$) and action ($a_t$) are changed in the Q-table using the Bellman equation [109]:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \epsilon(r_t + \gamma \times a_t maxQ(s_{t+1}, a_{t+1}) - Q(s_t, a_t)), \tag{3.30}$$

Where ($\gamma$) is the discount factor and $r_t$ is the reward function obtained by going from state ($s_t$) to ($s_{t+1}$). The ($\gamma$), which has a value between 0 and 1, represents how much we value

prior rewards [1]. On the other hand, the learning rate, represented by ($\epsilon$), influences how rapidly the table will converge. The DQN-based task offloading strategy is summarized in Algorithm 2.

**Algorithm 2** DQN-based task offloading strategy

**Input** Tasks T, learning rate($\epsilon$), discount factor ($\gamma$).

**Output** Optimal offloading decision and total cost.

**Function** DQN Strategy:
Initialize replay memory D to capacity N.
Initialize total episode reward r = 0.
**for** Each episode **do**
  Reset environment to initial state s.
  **for** Each step **do**
    Observe actual state $s_t$.
    Determine feasible action.
    random = randomly choose from [0, 1]
    **if** (random < $\epsilon$) **then**
      $a_t$ = randomly select action from {0,1,2}
    **else**
      $a_t = \gamma * max_{action}$ Q($s_t, a_t$).
    **end if**
    Execute action $a_t$.
    Calculate reward $r_t$ using Equation (3.29).
    Observe next state $s_{t+1}$.
    Store $(s_t, a_t, r_t, s_{t+1})$ in replay memory.
    Update $Q(s_t, a_t)$ according to Equation (3.30)
  **end for**
**end for**

Two neural networks that are the evaluation network and the target network make up the DQN architecture used to construct the suggested algorithm. The evaluation network is regularly updated throughout training and estimates Q-values, or the anticipated future rewards for every potential course of action. On the other hand, the evaluation network and the target network are synchronized at predetermined intervals to stabilize learning and offer

steady Q-value targets. Three hidden layers, each with 64 neurons, make up the comparable design of both networks. The output layer uses a linear activation to enable accurate Q-value predictions, whereas the hidden layers use the ReLU activation function.

## 3.7 Conclusion

In conclusion, the main contribution of this work is our suggested DQN-based task offloading algorithm, which was introduced in this chapter. Deep reinforcement learning is integrated into the suggested method to accomplish intelligent and adaptive task placement decisions, successfully addressing the issues of dynamic and heterogeneous IoT–fog–cloud ecosystems. Using DQN, the offloading agent approximates the Q-value function using neural networks to balance short-term and long-term goals while continuously interacting with the environment to learn optimal policies. By constantly adapting to changes in network conditions, resource availability, and task demands, the suggested model ensures effective use of computational resources in contrast to static or heuristic approaches.

Our contribution is the development of a latency-efficient and cost-conscious DQN framework that preserves system scalability and stability while reducing response time and energy usage. The suggested approach outperforms conventional methods in terms of adaptability and robustness by providing a reward formulation that simultaneously takes energy efficiency and delay into account, along with a balanced weighting mechanism between communications and computing costs. The model's ability to lower overall system costs and achieve near-optimal performance under various network situations is further supported by the experimental evaluation. Last but not least, our contribution lays the groundwork for intelligent, context-aware scheduling and offloading algorithms, opening the door for upcoming developments incorporating multi-agent systems and extensive IoT deployments.

# Chapter 4

# Performance Analysis and Discussion

## 4.1 Introduction

The performance analysis of the proposed DQN-based task offloading algorithm is a crucial step in validating its efficiency, robustness and adaptability within the IoT–fog–cloud ecosystem. Based on the system model and the suggested approach described previously, this chapter provides a comprehensive analysis of the proposed algorithm in a variety of system configurations, focusing on important performance indicators such as response time, energy consumption, resource utilization, scalability, and overall implementation cost, which provide a direct indication of the feasibility of implementing the model in real-world environments.

The DQN-based method is compared with other fundamental techniques, such as local execution, cloud-only offloading, traditional Q-learning, and experimental or meta-experimental strategies such as DJA and bat algorithm, in order to ensure a comprehensive evaluation. Furthermore, to emphasize the unique advantages of DQN in balancing computational efficiency and communication costs, comparisons are made with more advanced reinforcement learning approaches, such as DDPG. The chapter explores the convergence behavior of the model, considering its ability to adapt to changing workloads and heterogeneous network conditions, as well as the stability and speed of the learning process. Parameter sensitivity studies are a key component of this research, as they examine trade-offs between communication and computation costs under different network dynamics, task sizes, and device capabilities. These tests demonstrate the robustness of the proposed approach, as well as its scalability and generalize ability to unknown task distributions. Along with a comprehensive explanation, the results show that even in highly dynamic and resource-constrained IoT scenarios, the DQN-based algorithm provides reliable, low-latency, and energy-efficient task

distribution. The simulation results are ultimately linked to more general practical applications in this chapter, establishing DQN-based task offloading as a viable basis for managing intelligent and scalable services in 6G and next-generation IoT systems.

## 4.2 Simulation setup and parameters

In order to evaluate and compare the performance of the proposed DQN-based task offloading strategy with other benchmark solutions, we have done simulations. The performance evaluation has been done for three performance metrics: latency, energy consumption, and cost of the algorithm.

### 4.2.1 System parameters

The performance of task offloading in cloud, fog, and IoT settings depends heavily on a set of system parameters that control connectivity and computation. In addition to defining network link constraints and the nature of the tasks to be performed, these parameters also determine the capabilities of devices, fog nodes, and cloud servers. In order to analyze the trade-offs between execution cost, energy usage, and latency, these metrics must be properly defined. The system parameters can be categorized into five groups:

- **Task-related parameters:**
  Each mobile device creates computational tasks with a computational workload $W_i$, maximum allowable latency $\tau_i$, and data size $D_{Ti}$. The task size multiplied by the CPU cycles per bit (CB), which represents the application's processing intensity, yields the workload. Across several computing tiers, these characteristics have a direct impact on execution time and energy consumption.

- **Device parameters:** Mobile devices have limited CPU frequency ($f_d$) and energy efficiency ($\eta_d$), indicating resource constraints. Therefore, local task processing reduces transmission overhead, but because of its restricted processing capability, it may result in higher energy consumption. Local scheduling delays are taken into consideration by the device queue latency.

- **Fog nodes parameters:**
  Medium computing units known as FNs ($F_j$) have different energy efficiency factors ($\eta_j$) and higher CPU frequencies ($f_j$). They offer lower transmission delays than cloud servers because they are located close to IoT devices. However, to predict real-world

performance, transport power requirements ($PW_{u2f}$) and queue latency must be taken into account.

- **Cloud parameters:**
  The cloud layer has the highest processing capacity ($f_c$), but due to its remote location, it causes longer transmission and propagation delays. The energy efficiency factor ($\eta_c$) and associated communication costs ($PW_{f2c}$) of the cloud represent a trade-off between high processing power and high energy consumption in long-range communications.

- **Network parameters:**
  Communication latency and energy costs are significantly influenced by bandwidth limitations. Two main bandwidth factors are taken into account: the bandwidth between the fog node and cloud server ($B_{f2c}$) and the bandwidth between the fog node and MD ($B_{u2f}$). The transmission time of data-intensive tasks is directly impacted by these parameters.

Together, these system parameters provide the foundation for modeling the cost function, which integrates latency and energy consumption into a unified optimization objective. Their careful definition ensures that the proposed DQN-based offloading strategy can be evaluated in a setting that closely reflects the complexities and dynamics of real-world IoT–fog–cloud deployments.

### 4.2.2 Simulation model

A comprehensive simulation model was created to mimic the dynamic interactions between cloud servers, fog nodes, and IoT devices in order to test the proposed DQN-based offloading approach. The simulation environment provides a realistic yet controlled platform for evaluating the performance of algorithms in a range of scenarios.

#### 4.2.2.1 Simulation environment

The main programming environment used in the simulation studies was Python version 3.19.13, which ensures interoperability with contemporary machine learning and reinforcement learning packages. A 64-bit version of Windows running on an HP Elite Book 840 G3 laptop, manufactured by Hewlett-Packard (USA), was used for all applications. In its configuration, the computer was equipped with an Intel(R) Core (TM) i5-6300U processor running at 2.50 GHz with 8 GB of RAM. Although the configuration of this device is modest

compared to high-performance computing servers, it was specifically chosen to represent a realistic, resource constrained scenario, which is typical of many edge computing situations. Using this platform, the study highlights the effectiveness of the proposed DQN-based offloading method under typical computational conditions, as well as its potential for deployment in real-world IoT–fog–cloud scenarios.

#### 4.2.2.2 System Setup

The system configuration consists of a three-tiered IoT, fog, and cloud architecture that simulates distributed computing scenarios found in the real world. 100 IoT devices produce heterogeneous computational tasks at the edge layer, with data sizes ranging from 10 to 500 MB (mega bits) and an average workload of 500 MFLOPs (mega floating-point operations per second). This reflects the various processing and latency requirements of the applications. 10 fog nodes at the intermediate layer, each with a moderate computational capacity and unique energy consumption profile, receive tasks from these devices, allowing for effective local processing and shorter transmission delays. A single cloud server with practically infinite computational power is set up at the top layer to manage computation-intensive tasks that are too big for the fog layer; however, this requires a longer transmission distance, which increases response time and communication costs. Using this setup, the performance and flexibility of the proposed DQN-based task offloading approach can be realistically evaluated in a variety of implementation settings.

#### 4.2.2.3 Simulation parameters

A summary of the simulation parameters used to assess the suggested DQN-based task offloading framework's performance can be found in Table 4.1. During the experimental phase, these parameters specify the learning, communication, and computation environments used. The number of IoT devices and fog nodes, CPU frequencies, energy efficiency coefficients, bandwidth capacities, and reinforcement learning hyper parameters (e.g., learning rate, discount factor, and exploration rate) are significant aspects that are included in this list. By setting these parameters, the simulation environment guarantees reproducibility and consistency, enabling a fair comparison of the suggested DQN algorithm with alternative baseline techniques.

Table 4.1: Simulation parameters

| **Parameter** | **Value** |
|---|---|
| Learning rate ($\epsilon$) | 0.001 |
| Discount factor ($\gamma$) | 0.99 |
| Batch size | 32 |
| Replay memory size | 100,000 |
| Initial exploration rate ($\delta$) | 1.0 |
| Maximum Episodes | 1000 |
| Maximum Steps per Episode | 200 |
| Latency Factor ($\alpha$) | 0.18 |
| Energy Factor ($\beta$) | 0.82 |
| CPU Frequency of device($f_d$) | 2.0 GHz |
| The CPU frequency of the fog node($f_j$) | 2.5 GHz |
| The CPU frequency of the cloud server($f_c$) | 3.0 GHz |
| Energy efficiency of the device ($\eta_d$) | 0.5 |
| Energy efficiency of the fog node ($\eta_j$) | 0.4 |
| Energy efficiency of the cloud server ($\eta_c$) | 0.3 |
| Device Queue latency | 5 ms |
| Fog layer Queue latency | 10 ms |
| Cloud Server Queue latency | 15 ms |
| Bandwidth between the MD and fog node($B_{u2f}$) | 0.1 W |
| The bandwidth between the fog node and cloud server ($B_{f2c}$) | 0.05 W |
| Task data size ($D_{Ti}$) | [10–500] MB |
| Task Workload ($w_{Ti}$) | 500 MFLOPS |

## 4.3 Evaluation Metrics

This section presents the performance metrics used to quantitatively evaluate the effectiveness of the proposed DQN-based task offloading algorithm. These metrics are designed to assess the algorithm's ability to optimize computation and communication performance across IoT, fog, and cloud layers while balancing latency, energy efficiency, and resource utilization.

### 4.3.1 Average latency (L):

This metrics represents the average time needed for completing all tasks including computation time, queuing delay, and transmission time, for all layers (IoT device, fog node, and cloud server). It is expressed as:

$$L = \frac{1}{N} \sum_{i=1}^{N} \left( L_{Ti}^{comp} + L_{Ti}^{queue} + L_{Ti}^{trans} \right) \tag{4.1}$$

Where$L_{T_i}^{comp}$,$L_{T_i}^{queue}$, and $L_{T_i}^{queue}$ represents the computation time, queuing delay, and computation time for task Ti, respectively. On the other hand, the N refers to the total number of tasks in the system.

For latency-sensitive IoT applications, lower average response times mean better real-time responsiveness and faster task execution such as healthcare monitoring and autonomous vehicles.

### 4.3.2 Average energy consumption (E):

The total energy required to complete all tasks, taking into account both computing and transmission energy, is measured by the average energy consumption metric. It is defined as:

$$E = \frac{1}{N} \sum_{i=1}^{N} \left( E_{Ti}^{comp} + E_{Ti}^{trans} \right) \tag{4.2}$$

Where $E_{Ti}^{comp}$, $E_{Ti}^{trans}$ represents the execution computation energy, and the task transmission energy respectively.

Keeping E to a minimum extends the life of IoT devices, particularly those with limited battery space.

### 4.3.3 Total system cost (C):

A multi-objective cost function is established in order to assess latency and energy efficiency simultaneously. Two weighting coefficients, $\alpha$ and $\beta$, are used to combine delay and energy consumption:

$$C = (\alpha \cdot L + \beta \cdot E) \tag{4.3}$$

Where, $\alpha$ and $\beta$ are two weighting coefficients for latency and energy consumption, respectively. This composite function combines energy consumption and communication delay into a single optimization goal. A good balance between processing performance and power efficiency was ensures by the optimal weight configuration, which was found to be $\alpha$=0.18 and

$\beta$=0.82 based on experimental evaluation. The experiments were conducted by simulating multiple task offloading scenarios under various network conditions, computation workloads, and device energy levels. Different combinations of $\alpha$ and $\beta$ were tested to observe their influence on the total system performance. For each configuration, both the total execution latency and the energy consumption were measured on local devices, edge servers, and cloud platforms. The configuration achieved the best trade-off between processing speed and energy efficiency, minimizing the overall cost function.

The selected evaluation measures thoroughly evaluate energy economy and temporal performance, two crucial needs in IoT-fog-cloud systems. Through the simultaneous reduction of latency and energy consumption, the suggested DQN-based offloading model improves the balance between QoS and QoE, guaranteeing effective, dependable, and long-lasting operation in diverse IoT contexts.

## 4.4 Results and Analysis

This section provides a comprehensive evaluation of the proposed DQN-based task offloading approach, focusing on convergence properties, training stability, and relative performance compared to existing techniques. The primary goal of this analysis is to verify the model's ability to reduce response time, energy consumption, and overall system cost in an IoT-fog-cloud architecture. To achieve a comprehensive performance evaluation, multiple simulation scenarios were conducted under different network conditions, task arrival rates, and resource availability levels.

### 4.4.1 Convergence and learning stability

This section presents the evaluation of the proposed DQN-based task offloading algorithm in terms of its convergence behavior and learning stability. At the beginning of the training process, the DQN agent exhibits a wide variation in execution times, fluctuating between 24 and 36 seconds during the initial episodes. This instability characterizes the exploration phase of the algorithm, where the agent systematically tests different offloading actions local execution, fog offloading, and cloud delegation to acquire knowledge about the environment state transitions, cost functions, and network dynamics.

As the learning process progresses, the agent gradually refines its decision-making policy through interaction with the environment and feedback from accumulated rewards. Convergence is achieved after approximately 50 episodes, as evidenced by the stabilization of the

fitness value and the reduction in variance of execution time. This marks the algorithm's transition from the exploration stage to the exploitation phase, during which the DQN leverages its learned policy to consistently select near-optimal offloading actions that minimize latency and energy consumption.

As shown in Figure 4.1, the average execution time stabilizes after convergence and varies slightly between 27 and 29 seconds during successive iterations. This stability shows that the proposed DQN model can maintain effective performance even in dynamic and unstable IoT-fog-cloud scenarios, successfully reaching policy convergence. The consistent execution times after convergence prove that the DQN method is well suited for real-time adaptive task offloading in heterogeneous computing networks, where it can successfully adapt to changes in the environment.

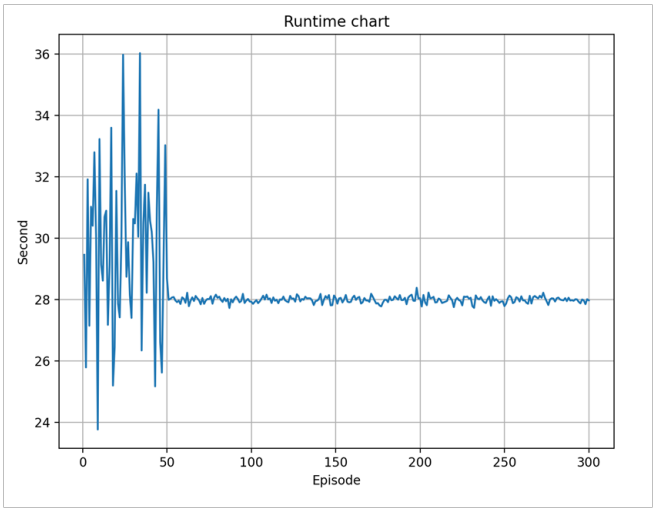


Figure 4.1: DQN approach execution time

Figure 4.2 illustrates the comparative learning curves of the proposed DQN-based task offloading algorithm and the optimal strategy in terms of total system cost over 300 training episodes. As shown, the DQN algorithm demonstrates faster learning and better adaptation to the environment by achieving convergence in significantly fewer epochs. This efficiency results from DQN's ability to continuously interact with the IoT–fog–cloud ecosystem to repeatedly refine Q-value estimates. This enables DQN to discover the best task offloading strategies without conducting an exhaustive search. As training continues, the total cost decreases steadily, demonstrating the agent's ability to reduce response time and energy consumption. In addition, the DQN model is more resilient than the best static approach in dynamically changing network conditions, as it quickly adapts to changes in resource availability and communication delays to improve overall system performance and user experience

quality.

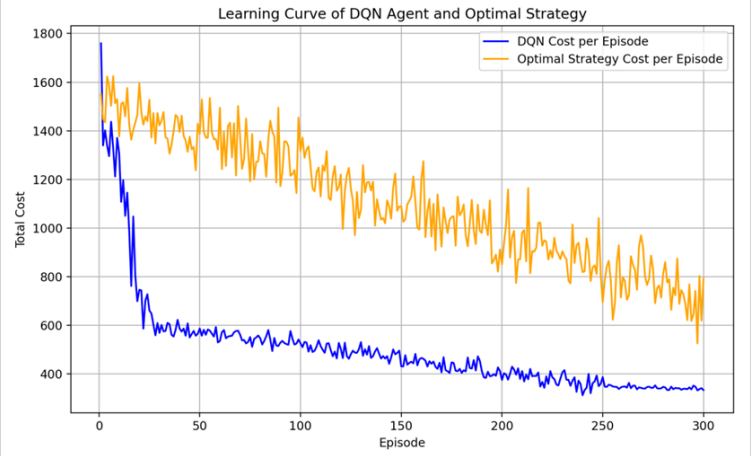


Figure 4.2: Optimal strategy and DQN training convergence process

### 4.4.2 The proposed algorithms comparative analysis

The two algorithms described in this thesis were applied in order to evaluate their performance by comparing them with other algorithms currently used in the same context. In this section, we present a comprehensive comparison of the performance and efficiency of the proposed algorithms using a set of performance indicators and application cases, with the parameters listed in Table 4.1.

#### 4.4.2.1 Energy consumption

Figure 4.3 compares the average energy consumption of the optimum method across the three execution layers (local, fog, and cloud) with the suggested DQN-based task offloading algorithm. In order to assess the scalability and flexibility of each approach, the tests were conducted for two distinct task loads, consisting of 10 tasks and 100 tasks. In both situations, the DQN-based method shows better energy efficiency at every execution tier, with the cloud layer showing the most increase in terms of energy savings.

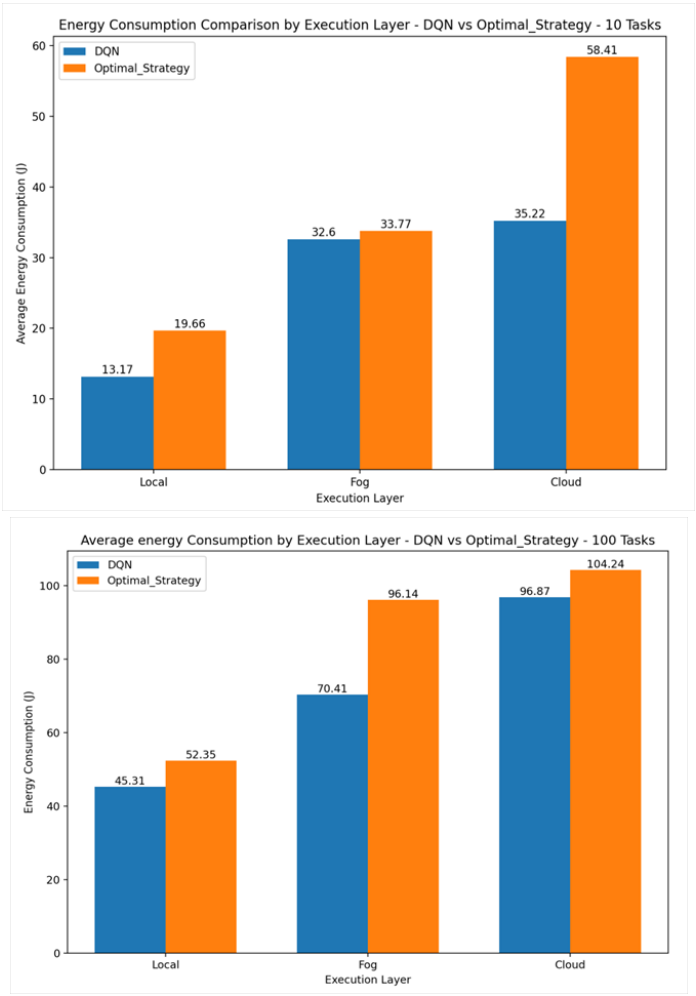


Figure 4.3: the average energy consumption for DQN and optimal strategy in different execution layers for 10 tasks and 100 tasks.

Specifically, the DQN algorithm achieves a reduction in response time of approximately 33% for 10 tasks and 30% for 100 tasks compared to the optimal strategy in the cloud layer. This significant improvement can be attributed to the parallel learning and decision-making capabilities of the DQN framework, which allows for the simultaneous evaluation of multiple pairs of states and actions and the efficient distribution of tasks across available resources. Whereas the optimal strategy typically relies on sequential computation, limiting its ability to exploit parallelism and adapt quickly to network fluctuations.

According to quantitative statistics, the DQN algorithm reduces energy consumption by approximately 25% for 10 tasks and 16% for 100 tasks compared to the best method. These results demonstrate how the model can maintain excellent energy efficiency as the number of tasks increases. The DQN-based method is a reliable option for sustainable and energy-

conscious IoT–fog–cloud systems, as it consistently saves energy across all processing levels and scales well with increasing workloads, according to the research.

#### 4.4.2.2 Latency

The average response time across the three execution levels (local device, fog, and cloud) is compared between the best method and the proposed DQN-based offloading strategy in Figure 4.4 To test scalability and responsiveness under different workloads, the evaluation is performed under two task load scenarios with 10 tasks and 100 tasks. The figure shows how the DQN-based method consistently reduces response time at each execution layer, demonstrating its exceptional ability to dynamically adapt to different network and computing conditions.

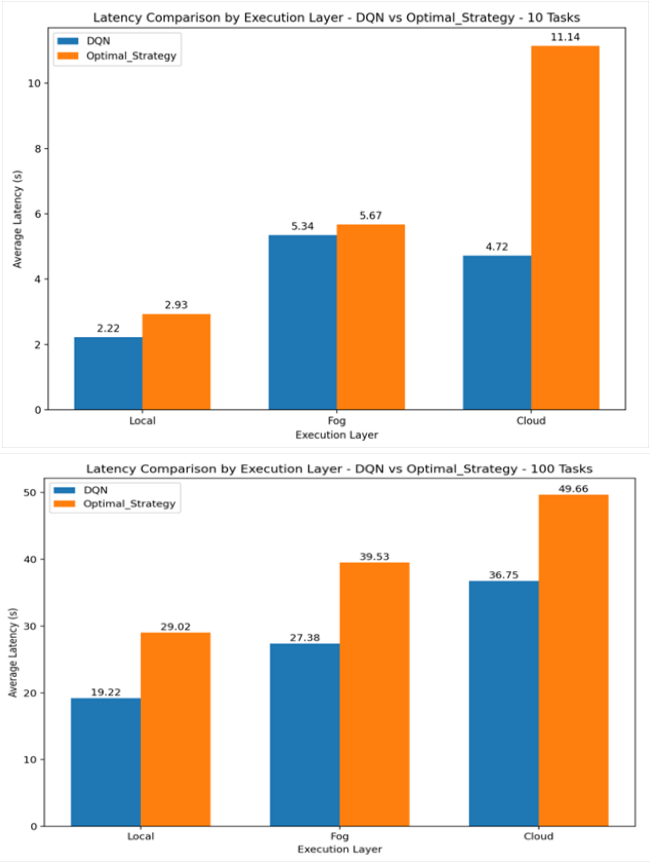


Figure 4.4: the average latency for DQN and optimal strategy in different execution layers for 10 tasks and 100 tasks.

The DQN method reduces response time by approximately 33% for 10 tasks and 30% for 100 tasks compared to the best strategy, representing a significant increase in performance

at the cloud layer level. This remarkable increase was achieved thanks to the parallel learning and decision-making capabilities of the DQN framework, which enables the simultaneous evaluation of multiple pairs of states and actions and the efficient distribution of tasks among available resources. On the other hand, the best method typically uses sequential computation, which limits its ability to take advantage of parallelism and adapt quickly to network fluctuations.

### 4.4.3 Comparative performance results

This study includes a comprehensive comparison of performance within the IoT–fog–cloud computing environment by evaluating the proposed optimal strategy and DQN-based task offloading algorithm against three standard methods: Discrete Jaya Algorithm (DJA) based algorithm [110], bat-based algorithm [11], and deep deterministic policy gradient (DDPG) based algorithm [95]. The underlying optimization operations and environmental assumptions of these algorithms differ significantly from the proposed methods, although they share some modeling parameters and operational constraints.

#### 4.4.3.1 Latency evaluation

The DJA algorithm [110] represents a metaheuristic-based task scheduling scheme designed to optimize latency and resource utilization in fog computing environments. It uses heuristic decision-making based on the Discrete Jaya Algorithm to efficiently find suboptimal solutions, but often struggles to adapt to highly dynamic network conditions. In contrast, the bat algorithm [11] proposes an AI-powered workload offloading and resource allocation framework for fog cloud systems that can be enhanced with optical networking capabilities, improving communication efficiency and transmission reliability. While both DJA and bat algorithms aim to improve response time and resource allocation, they rely on predefined or rule-based heuristics, limiting their ability to adapt to evolving environments in real time.

Furthermore, a deep reinforcement learning model specifically designed for MEC systems assisted by Unmanned Aerial Vehicles (UAV) has been introduced using DDPG technology [95]. The DDPG method operates in various dynamic aerial environments, where UAV are used to improve computational coverage, unlike DJA and bat algorithms, which focus on static or fog-centric designs. Although this method is intended for situations that differ from those of traditional IoT infrastructures fog and cloud, it shows great adaptability when movement is restricted.

Figure 4.5 presents the comparison of average latency achieved by different task offloading

strategies under various workloads. These strategies include the optimal strategy, the proposed DQN-based algorithm, and other reference techniques such as bat, DJA, and DDPG algorithms. As expected, increased communication overhead, queue delays, and increased competition for system resources lead to higher average response times for all methods as the number of tasks increases. With moderate latency, the bat algorithm outperforms DJA but performs worse than DQN, DDPG, and the optimal strategy. This is due to bat's search-based methodology, which, despite its success, may not respond to changes in dynamic tasks, resulting in increased transit time compared to the adaptive DQN approach. On the other hand, DJA has the highest latency among all the algorithms considered, as its deterministic heuristic approach lacks the adaptability and flexibility of AI-powered techniques. This limitation results in suboptimal latency performance, particularly in settings with dynamic workloads and diverse IoT demands.

The DDPG-based approach achieves better performance in terms of latency compared to benchmark algorithms because it is adept at managing continuous states and workspaces, allowing it to develop precise unloading strategies that adapt effectively to changing environmental conditions.

However, the DQN algorithm is consistently the most effective technique evaluated, achieving the lowest latency. Its strong performance is maintained as the workload increases, demonstrating its adaptability and scalability. Compared to the optimal strategy and bat, DQN improves response time by more than 30% in quantitative terms and outperforms DJA by approximately 40%. The optimal approach outperforms bat and DJA across the range studied, while starting with a slightly longer response time than DQN.

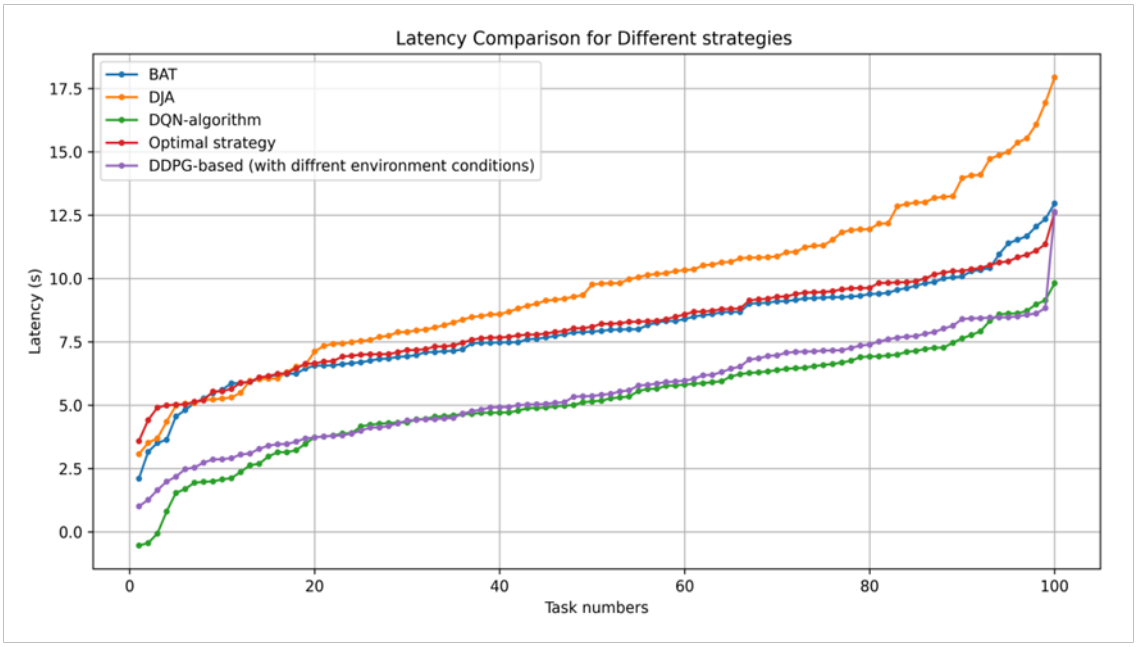


Figure 4.5: Latency Comparison across task offloading strategies.

#### 4.4.3.2 Energy consumption analysis

Figure 4.6 illustrates the average energy consumption comparison for different task offloading strategies, including the optimal strategy, the proposed DQN-based algorithm, and benchmarking techniques such as bat, DJA, and DDPG algorithms. In this context, energy consumption includes the computing energy needed to complete activities as well as the transmission energy needed to transfer data between cloud servers, fog nodes, and IoT devices. A shown in the figure, the DQN algorithm consistently uses the least amount of energy in all evaluated scenarios. Its ability to learn and adapt dynamically to changing system conditions is what makes this optimization possible. It optimizes task distribution to reduce unnecessary transport and avoid energy-intensive activities. The DQN agent achieves an effective balance between computation and communication by using reinforcement learning to continuously update the offloading policy. This allows it to select the execution layers that use the least amount of energy overall for each task.

However, across all arrival rates, the DQN algorithm consistently maintains the lowest energy consumption, demonstrating its ability to dynamically adjust offloading options depending on resource availability, network load, and system conditions. Thanks to its flexibility, the DQN model can effectively control task distribution, reducing energy consumption even in high-traffic situations. However, due to its reliance on fixed analytical optimization instead of adaptive learning, the optimal method works quite well but exhibits a more

pronounced rise in overall energy consumption at higher arrival rates. Significantly larger energy consumption are shown by the bat and DJA algorithms, suggesting less effective task distribution under increased workload. Because the DDPG method was first developed for UAV-assisted environments rather than distributed IoT–fog–cloud systems, it fails to adapt effectively to the heterogeneous and multi-layered nature of the system.

As depicted, the DQN-based strategy significantly reduces energy consumption compared to the bat and DJA algorithms. While bat and DJA start with higher energy levels of around 40 joules, DQN and optimal strategies start at around 20 joules. DQN achieves approximately 35% and 50% lower energy consumption than bat and DJA, respectively. This demonstrates the superior adaptability and efficiency of the DQN model in dynamically optimizing task offloading decisions to reduce energy usage in IoT–fog–cloud environments.

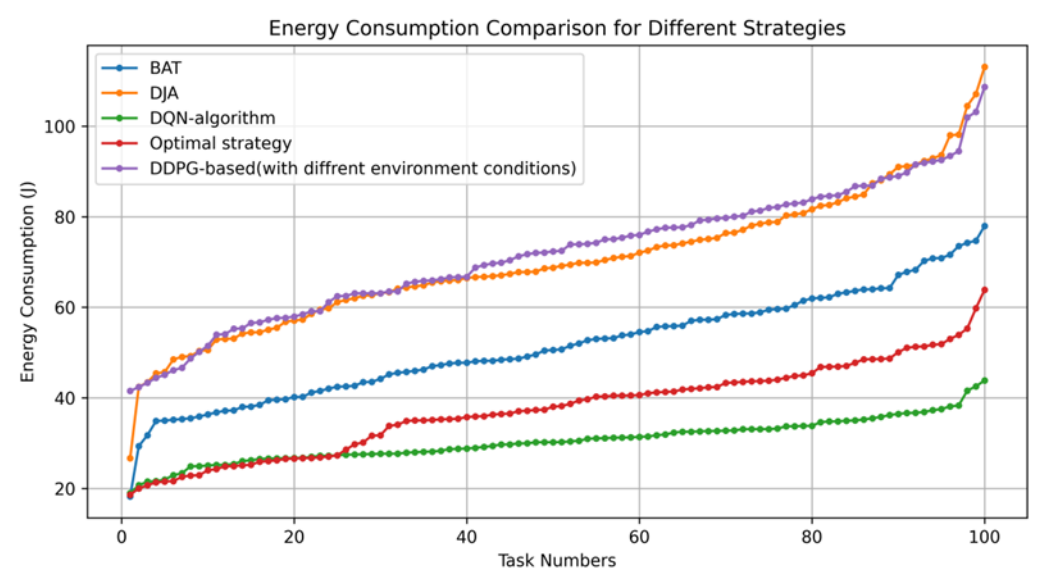


Figure 4.6: Energy consumption comparison across tasks offloading strategies

#### 4.4.3.3 Total cost comparison

Figure 4.7 illustrates how the total cost of the system changes as the number of tasks increases, revealing notable differences in the performance of the algorithms examined. Across a range of workloads, the DQN-based strategy consistently outperforms all other strategies and achieves the lowest total cost. Compared to the DJA and bat algorithms, the optimal approach shows an approximately 30% reduction in total cost as the number of tasks increases. This indicates the theoretical efficiency of the strategy but its poor adaptability in dynamic contexts. On the other hand, since it was created for UAV-assisted scenarios rather

than distributed IoT–fog–cloud scenarios, the DDPG-based approach shows moderate and less reliable results. On the other hand, the DQN algorithm's intelligent, adaptive learning mechanism continuously improves offloading decisions based on task complexity, network conditions, and available computational resources, resulting in up to 50% lower costs than both DJA and BAT. In diverse and dynamic computing settings, this demonstrates the scalability and resilience of DQN in maximizing cost efficiency.

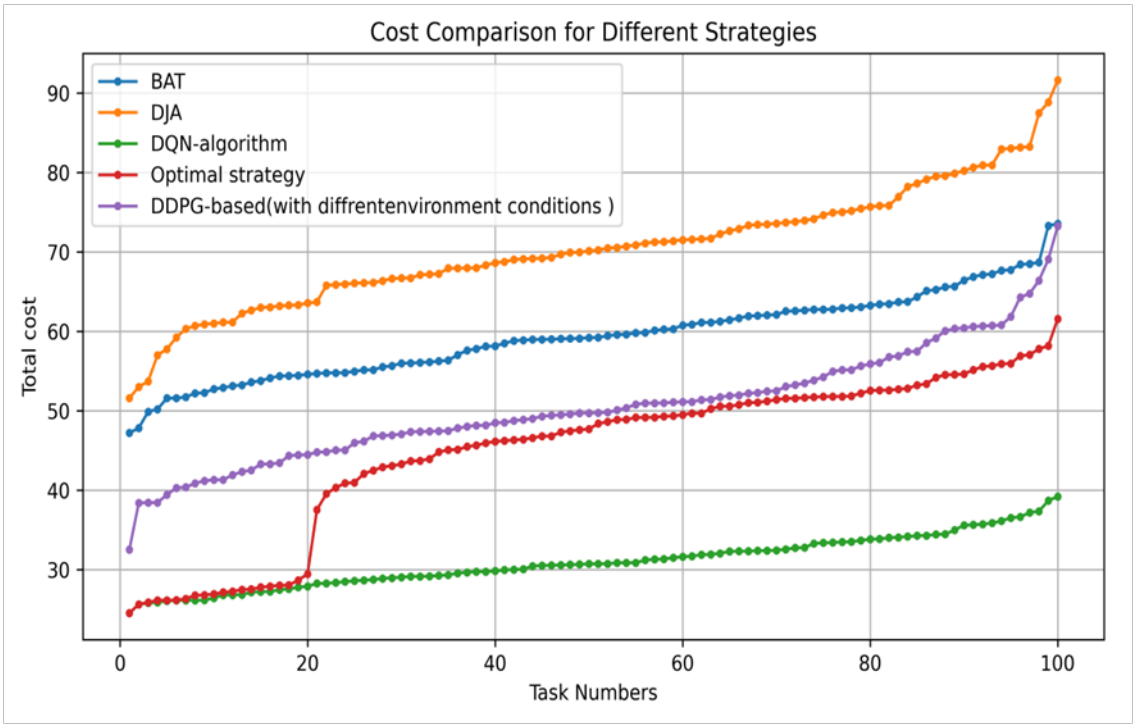


Figure 4.7: Total cost comparison across tasks offloading strategies

However, the DJA algorithm is more expensive than the other approaches because it relies on static, rule-based decision-making, which leads to less-than-ideal resource use and restricted scalability. Although dynamic in nature, DJA differs from the more adaptable systems studied in that it is unable to effectively respond to changes in system state, resource heterogeneity, and multi-objective constraints.

#### 4.4.3.4 Variability and Reliability Analysis

A statistical analysis was performed on the reliability and variability of the performance metrics of the proposed DQN-based task offloading strategy, such as response time, total cost, and energy consumption, in order to further evaluate its robustness. The results highlight significant differences in both performance efficiency and stability among the evaluated

algorithms.

The DQN-based algorithm is the best at reducing system cost (28.0), energy consumption (30.2), and latency (5.1), as evidenced by the lowest average values it obtains across all three parameters. Furthermore, all measurements show remarkably low variance, with standard deviations of less than 0.04. The stability, reliability, and consistency of the algorithm under dynamic IoT–fog–cloud environments are highlighted by the standard deviation of cost of 0.012, which shows a particularly narrow confidence interval. This stability results from the DQN model's ability to successfully adapt to changing workloads and resource conditions by continuously optimizing its decision-making policy through reinforcement learning.

On the other hand, heuristic algorithms like bat and DJA exhibit the highest mean values for every performance metric. Of these, the DJA approach records the highest latency (10.2) and energy consumption (70.3). Their static decision-making processes result in considerable performance swings and poor flexibility, as indicated by these raised values and higher standard deviations (up to 0.25). Since the DDPG-based approach was first created for UAV-assisted environments rather than distributed IoT–fog–cloud systems, it outperforms DQN despite having mean values that are somewhat higher than DQN due to its non-adaptive nature. The optimal strategy performs fairly well.

When compared to other benchmark algorithms (bat, DJA, Optimal, and DDPG), the DQN method consistently shows low standard deviation values, as shown in Table 4.2. This indicates that even with varying system conditions and workload, the proposed method continues to operate stably and predictably. The reliability of the DQN method in a variety of operating conditions is confirmed by the cost metric, which shows a standard deviation of less than 0.012, or a narrow 95% confidence interval.

Overall, the table demonstrates that the suggested DQN-based offloading technique guarantees higher dependability and predictability of results in addition to outperforming all benchmark approaches in terms of efficiency. These outcomes confirm the stability of DQN's learning framework and show that it has the potential to be a scalable and energy-efficient real time IoT–fog–cloud task management solution.

### 4.4.4 Discussion of findings

This section discusses the results of comparing the proposed DQN-based task offloading strategy with four reference methods: optimal strategy, bat, DJA, and DDPG. The reliability, stability, and scalability of each method are examined, and the experimental results are interpreted in terms of response time, energy consumption, and total cost. This anal-

Table 4.2: Mean and standard deviation of the three metrics across100 tasks.

| **Strategy** | **Metrics** | **Mean** | **Std. dev.** |
|---|---|---|---|
| BAT | Cost | 58.7 | 0.024 |
| | Energy consumption | 55.0 | 0.18 |
| | Latency | 8.7 | 0.07 |
| DJA | Cost | 65.2 | 0.031 |
| | Energy consumption | 70.3 | 0.25 |
| | Latency | 10.2 | 0.09 |
| DQN-based algorithm | Cost | 28.0 | 0.012 |
| | Energy consumption | 30.2 | 0.01 |
| | Latency | 5.1 | 0.04 |
| Optimal strategy | Cost | 30.5 | 0.015 |
| | Energy consumption | 35.0 | 0.13 |
| | Latency | 6.0 | 0.05 |
| DDPG-based (with different environment constraints) | Cost | 45.3 | 0.029 |
| | Energy consumption | 50.5 | 0.20 |
| | Latency | 4.5 | 0.06 |

ysis clearly shows how the proposed DQN framework helps IoT–fog–cloud systems achieve efficient, intelligent, and adaptive task offloading.

#### 4.4.4.1 Comparative overview of algorithmic performance

According to the comparative results, the proposed DQN-based algorithm performs better than the baseline techniques in all three key performance metrics. Compared to experimental algorithms such as bat and DJA, DQN technology efficiently reduces overall costs, saving up to 50% while maintaining a 25-35% increase in energy efficiency and reducing average response time by 30-33%. The reason for these results is that the DQN agent can dynamically adapt to changing network conditions and workload density by repeatedly optimizing the offloading policy through reinforcement learning. The DQN model makes near-optimal choices that maintain responsiveness in complex contexts while minimizing the overall cost of the system by learning from previous interactions and feedback.

On the other hand, fixed or semi-random search techniques used by traditional heuristic algorithms such as bat and DJA are inflexible and result in unstable and energy-consuming

task distribution under high load conditions. The exhaustive search performed by the optimal strategy makes it computationally expensive and impractical in large and dynamic IoT contexts, despite its effectiveness in small-scale settings. Similarly, the DDPG-based approach initially created for MEC contexts supported by drones with distinctive communication and movement dynamics, shows poor adaptability in distributed IoT-fog-cloud systems, even if it is a deep reinforcement learning technique.

#### 4.4.4.2 Energy and latency insights

According to the energy study, the DQN-based offloading system is the most energy-efficient across all levels of implementation (local, fog, and cloud). The DQN agent's ability to dynamically partition computational workloads between fog and cloud layers reduces redundant transport costs and eliminates redundant data offloading, which is the source of this improvement. The agent may improve short-term and long-term incentives through adaptive learning, resulting in reduced cumulative energy consumption while maintaining real-time responsiveness.

Response time data supports the superiority of the DQN algorithm. Compared to alternative methods, the DQN model significantly reduces end-to-end response time through continuous learning from environmental feedback. The agent effectively determines the best places to execute activities, using cloud resources for computationally intensive tasks and fog layer as needed to reduce communication response time. Due to its adaptability, the DQN framework can operate with extremely low response times even in high-density task conditions, where static and experimental methods often show performance degradation.

- **Reliability and Stability of the DQN Model**
  The consistency of the proposed DQN-based method was verified by studying variance and reliability 4.2. With the lowest standard deviation values (less than 0.04) among all performance indicators, the method demonstrates its resilience in dealing with varying workloads and network conditions. The low 95% Confidence Interval (CI) of the DQN model results confirms the statistical significance of its stable performance. In comparison, the bat and DJA algorithms show greater variability, indicating volatile behavior and sensitivity to changes in the environment. Although the Optimal and DDPG approaches are more stable than the experimental approach, they produce more variable results than the DQN technique.

- **Overall Discussion and Implications**
  The results taken together show that the suggested DQN-based approach offers a reli-

able and clever way to offload tasks in IoT–fog–cloud systems. The DQN model efficiently adjusts to changing situations by utilizing deep reinforcement learning to make decisions in real time that minimize system cost, latency, and energy consumption. In contrast to static optimization or heuristic models, the DQN algorithm ensures adaptability, scalability, and dependability by continuously adjusting its strategy in response to environmental feedback.

These findings not only support the technical value of incorporating DQN into the offloading decision-making process, but they also demonstrate its applicability in large-scale IoT systems, where diverse devices and varying workloads present difficult problems. The DQN framework's exceptional performance indicates that it has a great chance of being used in real-time, energy-constrained, latency-sensitive Internet of Things applications including industrial automation, smart cities, and autonomous systems.

# Conclusion

In summary, this chapter has offered a thorough examination and interpretation of the experimental findings derived from comparing the suggested DQN-based task offloading framework to a number of benchmark algorithms, such as the Optimal, bat, DJA, and DDPG strategies. The findings confirm that the proposed framework achieves significant improvements in cost, energy consumption, and latency without compromising stability or reliability. The DQN model outperforms static optimization techniques and traditional heuristics that have trouble in dynamic IoT–fog–cloud environments by dynamically adjusting offloading decisions based on real-time system states through its adaptive reinforcement learning mechanism.

According to the performance analysis results, the proposed DQN-based offloading solution outperforms traditional optimization techniques and experimental methods. The DQN architecture consistently eliminated implementation costs, reduced latency, and consumed less energy across a range of task sizes and computer levels. Notably, the algorithm demonstrated rapid convergence and remained stable even in the face of highly diverse and dynamic IoT, fog, and cloud scenarios. The flexibility and resilience of the DQN approach has been demonstrated by notable advantages of up to 50% in energy savings and 40% in latency reduction compared to bat, DJA, and DDPG. Furthermore, even when compared to the theoretical optimum, DQN has demonstrated its ability to deliver near-optimal results in real-world environments where comprehensive information is rarely available. Considering

all factors, these results show that integrating DQN learning into IoT-fog-cloud systems not only improves system performance but also ensures reliable, scalable, and economical task execution, making it a viable option for future energy- and latency-sensitive IoT applications.

This chapter analyses and interprets the experimental results comparing the proposed DQN-based task offloading framework with a number of benchmark algorithms, such as Optimal, bat, DJA, and DDPG strategies. The results clearly show that the proposed approach maintains a high level of stability and reliability and significantly improves all important performance parameters, including total cost, energy consumption, and latency. The DQN model outperforms static optimization techniques and standard empirical methods that face difficulties in dynamic IoT–fog–cloud scenarios by dynamically adjusting discharge decisions based on real-time system variables through adaptive reinforcement learning.

# General conclusion

## Conclusion

In this thesis, we address the problem of efficient task offloading in collaborative Internet of Things-fog-cloud systems, where communication conditions and computing resources are highly diverse and heterogeneous. There is a growing demand for intelligent decision-making that can balance performance indicators such as response time, energy consumption, and system cost due to the explosive growth of connected devices and data-intensive applications. To meet these needs, we propose a DQN-based task offloading architecture that continuously interacts with the environment to learn the best offloading techniques. Initially, the problem was presented as a collaborative optimization model with the goal of minimizing energy consumption and response time across cloud, fog, and IoT layers [111]. The proposed approach dynamically classifies tasks and determines the best place to execute them based on network conditions, computational load, and device power constraints using reinforcement learning.

Without requiring complete prior knowledge of the environment, the proposed DQN algorithm allows the system to learn from past experiences, adapt to changing conditions, and achieve near-perfect performance. According to extensive simulations, the DQN-based approach outperforms state-of-the-art techniques, such as optimum, bat, DJA, and DDPG-based algorithms. In particular, the DQN model demonstrated its ability to provide reliable and flexible performance in complex IoT–fog–cloud systems by achieving a cost reduction of up to 50%, a response time improvement of 33%, and an energy efficiency increase of 25%. Furthermore, the proposed system includes task classification to facilitate intelligent offloading choices by determining whether the local, fog, or cloud execution layer is most suitable for each type of activity. The proposed method is suitable for real-time applications that require low latency and high reliability because this integration improves the flexibility and scalability of IoT infrastructures.

Overall, the results support the idea that reinforcement learning, particularly DQN, pro-

vides a reliable, scalable, and context-aware approach to the task unloading problem. This model is a viable path for future distributed computing systems due to its ability to continuously improve its policy, ensuring long-term system efficiency.

## Future works

Although the proposed DQN-based task offloading scheme has shown excellent results in reducing response time and energy consumption in IoT–fog–cloud settings, there are still a number of interesting research areas that need to be explored. Future computing offloading systems should be more intelligent, flexible, and scalable thanks to these insights.

The implementation of multi-agent reinforcement learning is a key avenue for further study. A single DQN agent makes all decisions regarding task offloading in the current work. The growing number of IoT devices and fog nodes may limit the scalability of this centralized solution. Distributed intelligence can be achieved through the system by deploying multiple autonomous agents that collaborate globally and learn locally, enhancing scalability and responsiveness in dynamic workload and network conditions. Furthermore, future research should focus on applying the proposed technique in the real world using advanced computing platforms such as Raspberry Pi or experimental test platforms. Verification under realistic conditions involving changing network characteristics, device movement, and workload variability will be possible by deploying the DQN framework in actual IoT and fog cloud infrastructures. The flexibility and resilience of the algorithm to changes in the environment will be further enhanced by adding online learning and transfer learning techniques. Moreover, although this study improves two important metrics, response time and energy consumption, and future research should consider multi-objective optimization by incorporating variables such as task success rate, reliability, productivity, and bandwidth. Pareto-based optimization and multi-objective reinforcement learning may be able to resolve competing performance objectives, increasing system efficiency and user satisfaction.

Finally, future studies could improve this classification method by grouping tasks in an adaptive and context-aware manner, as this thesis focuses on classifying tasks for unloading decisions. Unsupervised learning techniques such as self-organizing maps and clustering can be used to automatically classify tasks according to characteristics such as response time sensitivity, computational intensity, or data volume. When combined with reinforcement learning, this system will enable more intelligent task scheduling and prioritization, ensuring that sufficient resources are available for critical operations while maximizing overall

performance.

In conclusion, the proposed DQN-based model provides a robust foundation for task offloading in intelligent and flexible IoT–fog–cloud systems. However, by adopting multi-agent, unified, and hybrid learning methodologies, real-world deployment, multi-objective optimization, and sustainable computing, future research can expand its scope. These developments will enable the creation of next-generation intelligent systems capable of energy-efficient, context-aware, real-time judgments in dynamic and increasingly complex IoT contexts.

# Bibliography


[1] Amina Benaboura, Rachid Bechar, Walid Kadri, Tu Dac Ho, Zhenni Pan, and Shaaban Sahmoud. Latency-aware and energy-efficient task offloading in iot and cloud systems with dqn learning. *Electronics*, 14(15):3090, 2025.

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

[3] Bardia Safaei, Aliasghar Mohammadsalehi, Kimia Talaei Khoosani, Saba Zarbaf, Amir Mahdi Hosseini Monazzah, Farzad Samie, Lars Bauer, Jörg Henkel, and Alireza Ejlali. Impacts of mobility models on rpl-based mobile iot infrastructures: An evaluative comparison and survey. *IEEE access*, 8:167779–167829, 2020.

[4] Marios Avgeris. Dynamic resource allocation and computational offloading at the network edge for internet of things applications. *National Technical University of Athens*, 2021.

[5] Babak Ravandi and Ioannis Papapanagiotou. A self-learning scheduling in cloud software defined block storage. In *2017 IEEE 10th International Conference on Cloud Computing (CLOUD)*, pages 415–422. IEEE, 2017.

[6] Ejaz Ahmed, Ibrar Yaqoob, Abdullah Gani, Muhammad Imran, and Mohsen Guizani. Internet-of-things-based smart environments: state of the art, taxonomy, and open research challenges. *IEEE Wireless Communications*, 23(5):10–16, 2016.

[7] Mahadev Satyanarayanan. A brief history of cloud offload: A personal journey from odyssey through cyber foraging to cloudlets. *GetMobile: Mobile Computing and Communications*, 18(4):19–23, 2015.

[8] Ibrahim Abaker Targio Hashem, Victor Chang, Nor Badrul Anuar, Kayode Adewole, Ibrar Yaqoob, Abdullah Gani, Ejaz Ahmed, and Haruna Chiroma. The role of big data in smart city. *International Journal of information management*, 36(5):748–758, 2016.

[9] Hatem A Alharbi, Mohammad Aldossary, Jaber Almutairi, and Ibrahim A Elgendy. Energy-aware and secure task offloading for multi-tier edge-cloud computing systems. *Sensors*, 23(6):3254, 2023.

[10] Mohit Kumar, Subhash Chander Sharma, Anubhav Goel, and Santar Pal Singh. A comprehensive survey for scheduling techniques in cloud computing. *Journal of Network and Computer Applications*, 143:1–33, 2019.

[11] Mohammad Aknan, Maheshwari Prasad Singh, and Rajeev Arya. Ai and blockchain assisted framework for offloading and resource allocation in fog computing. *Journal of Grid Computing*, 21(4):74, 2023.

[12] Jinming Shi, Jun Du, Jingjing Wang, Jian Wang, and Jian Yuan. Priority-aware task offloading in vehicular fog computing based on deep reinforcement learning. *IEEE Transactions on Vehicular Technology*, 69(12):16067–16081, 2020.

[13] Liang Yu, Tao Jiang, and Yulong Zou. Fog-assisted operational cost reduction for cloud data centers. *IEEE Access*, 5:13578–13586, 2017.

[14] Branka Mikavica and Aleksandra Kostic-Ljubisavljevic. A truthful double auction framework for security-driven and deadline-aware task offloading in fog-cloud environment. *Computer Communications*, 217:183–199, 2024.

[15] Taha Alfakih, Mohammad Mehedi Hassan, Abdu Gumaei, Claudio Savaglio, and Giancarlo Fortino. Task offloading and resource allocation for mobile edge computing by deep reinforcement learning based on sarsa. *Ieee Access*, 8:54074–54084, 2020.

[16] Abdeladim Sadiki, Jamal Bentahar, Rachida Dssouli, Abdeslam En-Nouaary, and Hadi Otrok. Deep reinforcement learning for the computation offloading in mimo-based edge computing. *Ad Hoc Networks*, 141:103080, 2023.

[17] Partha Pratim Ray. A survey on internet of things architectures. *Journal of King Saud University-Computer and Information Sciences*, 30(3):291–319, 2018.

[18] Fatima Zahra AMARA. *Représentation des Connaissances dans l'Internet des Objets: Modélisation Sémantique et Raisonnement*. PhD thesis, Université d'Oum El-Bouaghi, 2024.

[19] Zaheer Allam and Zaynah A Dhunny. On big data, artificial intelligence and smart cities. *Cities*, 89:80–91, 2019.

[20] Muhammad Shoaib Farooq, Shamyla Riaz, Adnan Abid, Kamran Abid, and Muhammad Azhar Naeem. A survey on the role of iot in agriculture for the implementation of smart farming. *Ieee Access*, 7:156237–156271, 2019.

[21] Mohd Javaid, Abid Haleem, Ibrahim Haleem Khan, Ravi Pratap Singh, and Abid Ali Khan. Industry 4.0 and circular economy for bolstering healthcare sector: A comprehensive view on challenges, implementation, and futuristic aspects. *Biomedical Analysis*, 1(2):174–198, 2024.

[22] Angeles Verdejo Espinosa, Jose Luis Lopez, Francisco Mata Mata, and Macarena Espinilla Estevez. Application of iot in healthcare: keys to implementation of the sustainable development goals. *Sensors*, 21(7):2330, 2021.

[23] Zahra Mohammadzadeh, Hamid Reza Saeidnia, Aynaz Lotfata, Mohammad Hassanzadeh, and Nasrin Ghiasi. Smart city healthcare delivery innovations: a systematic review of essential technologies and indicators for developing nations. *BMC health services research*, 23(1):1180, 2023.

[24] R Dave, N Seliya, and N Siddiqui. The benefits of edge computing in healthcare, smart cities, and iot. arxiv 2021. *arXiv preprint arXiv:2112.01250*.

[25] Chamara Kasun. Securing iot ecosystems: Challenges, solutions, and future directions. 2025.

[26] Faisal Karim Shaikh, Sherali Zeadally, and Ernesto Exposito. Enabling technologies for green internet of things. *IEEE Systems Journal*, 11(2):983–994, 2015.

[27] Fadele Ayotunde Alaba. Iot architecture layers. In *Internet of Things: A Case Study in Africa*, pages 65–85. Springer, 2024.

[28] Dina Darwish. Improved layered architecture for internet of things. *Int. J. Comput. Acad. Res.(IJCAR)*, 4(4):214–223, 2015.

[29] Pallavi Sethi and Smruti R Sarangi. Internet of things: architectures, protocols, and applications. *Journal of electrical and computer engineering*, 2017(1):9324035, 2017.

[30] Muhammad Burhan, Rana Asif Rehman, Bilal Khan, and Byung-Seo Kim. Iot elements, layered architectures and security issues: A comprehensive survey. *sensors*, 18(9):2796, 2018.

[31] Guoxing Yao and Lav Gupta. A survey on the use of partitioning in iot-edge-ai applications. *arXiv preprint arXiv:2406.00301*, 2024.

[32] Wazir Zada Khan, Ejaz Ahmed, Saqib Hakak, Ibrar Yaqoob, and Arif Ahmed. Edge computing: A survey. *Future Generation Computer Systems*, 97:219–235, 2019.

[33] Tom H Luan, Longxiang Gao, Zhi Li, Yang Xiang, Guiyi Wei, and Limin Sun. Fog computing: Focusing on mobile users at the edge. *arXiv preprint arXiv:1502.01815*, 2015.

[34] Roberto Beraldi, Abderrahmen Mtibaa, and Hussein Alnuweiri. Cooperative load balancing scheme for edge computing resources. In *2017 Second International Conference on Fog and Mobile Edge Computing (FMEC)*, pages 94–100. IEEE, 2017.

[35] Ashkan Yousefpour, Genya Ishigaki, Riti Gour, and Jason P Jue. On reducing iot service delay via fog offloading. *IEEE Internet of things Journal*, 5(2):998–1010, 2018.

[36] Saif Aljanabi and Abdolah Chalechale. Improving iot services using a hybrid fog-cloud offloading. *IEEE Access*, 9:13775–13788, 2021.

[37] Harshit Gupta and Ajay Kumar Bharti. Fog computing& iot: Overview, architecture and applications. *arXiv preprint arXiv:2304.08302*, 2023.

[38] Moteb K Alasmari, Sami S Alwakeel, and Yousef A Alohali. A multi-classifiers based algorithm for energy efficient tasks offloading in fog computing. *Sensors*, 23(16):7209, 2023.

[39] Sreenivasa S Sharma Nittala, Sangeeta Shah Bharadwaj, Shiv S Tripathi, and Heiko Seif. Service innovation enabled by internet of things and cloud computing–a service-dominant logic perspective. *Technology Analysis & Strategic Management*, 34(4):433–446, 2022.

[40] Maryam Sheikh Sofla, Mostafa Haghi Kashani, Ebrahim Mahdipour, and Reza Faghih Mirzaee. Towards effective offloading mechanisms in fog computing. *Multimedia Tools and Applications*, 81(2):1997–2042, 2022.

[41] E Cao, Saira Musa, Mingsong Chen, Tongquan Wei, Xian Wei, Xin Fu, and Meikang Qiu. Energy and reliability-aware task scheduling for cost optimization of dvfs-enabled cloud workflows. *IEEE Transactions on Cloud Computing*, 11(2):2127–2143, 2022.

[42] Ibrar Yaqoob, Ejaz Ahmed, Abdullah Gani, Salimah Mokhtar, and Muhammad Imran. Heterogeneity-aware task allocation in mobile ad hoc cloud. *IEEE Access*, 5:1779–1795, 2017.

[43] Mohammad Reza Rezaee, Nor Asilah Wati Abdul Hamid, Masnida Hussin, and Zuriati Ahmad Zukarnain. Fog offloading and task management in iot-fog-cloud environment: Review of algorithms, networks, and sdn application. *IEEE Access*, 12:39058–39080, 2024.

[44] Naser Shirvanian, Maryam Shams, and Amir Masoud Rahmani. Internet of things data management: A systematic literature review, vision, and future trends. *International Journal of Communication Systems*, 35(14):e5267, 2022.

[45] Seyed Omid Azarkasb and Seyed Hossein Khasteh. Fog computing tasks management based on federated reinforcement learning. *Journal of Grid Computing*, 23(1):11, 2025.

[46] Gauvav Goel and Rajeev Tiwari. Task management in iot-fog-cloud environment employing static scheduling techniques. *ENP Engineering Science Journal*, 2(1):13–20, 2022.

[47] Chunlin Li, Jun Liu, Weigang Li, and Youlong Luo. Adaptive priority-based data placement and multi-task scheduling in geo-distributed cloud systems. *Knowledge-Based Systems*, 224:107050, 2021.

[48] Esmaeil Torabi, Mostafa Ghobaei-Arani, and Ali Shahidinejad. A learning-based data and task placement mechanism for iot applications in fog computing: a context-aware approach. *The Journal of Supercomputing*, 80(15):21726–21763, 2024.

[49] Bing Tang, Huiyuan Han, Qing Yang, and Wei Xu. Operator placement for data stream processing based on publisher/subscriber in hybrid cloud-fog-edge infrastructure. *Cluster Computing*, 27(3):2741–2759, 2024.

[50] Kavitha Dhanushkodi, Raushan Kumar, Pratyush Mittal, Saumye Saran Das, Neelam Naga Saivenkata Suryavenu, and Kiruthika Venkataramani. Enhancing resource utilization and privacy in iot data placement through fuzzy logic and pso optimization. *Cluster Computing*, 27(9):12603–12626, 2024.

[51] Shanhe Yi, Cheng Li, and Qun Li. A survey of fog computing: concepts, applications and issues. In *Proceedings of the 2015 workshop on mobile big data*, pages 37–42, 2015.

[52] Balázs Sonkoly, János Czentye, Márk Szalay, Balázs Németh, and László Toka. Survey on placement methods in the edge and beyond. *IEEE Communications Surveys & Tutorials*, 23(4):2590–2629, 2021.

[53] Hao Liang, Sharad Sinha, Rakesh Warrier, and Wei Zhang. Static hardware task placement on multi-context fpga using hybrid genetic algorithm. In *2015 25th International Conference on Field Programmable Logic and Applications (FPL)*, pages 1–8. IEEE, 2015.

[54] Xiao Ma, Ao Zhou, Shan Zhang, Qing Li, Alex X Liu, and Shangguang Wang. Dynamic task scheduling in cloud-assisted mobile edge computing. *IEEE Transactions on Mobile Computing*, 22(4):2116–2130, 2021.

[55] Indranil Sarkar, Mainak Adhikari, Neeraj Kumar, and Sanjay Kumar. Dynamic task placement for deadline-aware iot applications in federated fog networks. *IEEE Internet of Things Journal*, 9(2):1469–1478, 2021.

[56] Farooq Hoseiny, Sadoon Azizi, Mohammad Shojafar, Fardin Ahmadiazar, and Rahim Tafazolli. Pga: A priority-aware genetic algorithm for task scheduling in heterogeneous fog-cloud computing. In *IEEE INFOCOM 2021-IEEE conference on computer communications workshops (INFOCOM WKSHPS)*, pages 1–6. IEEE, 2021.

[57] Mingze Wang, Yingjie Wang, Akshita Maradapu Vera Venkata Sai, Zhaowei Liu, Yang Gao, Xiangrong Tong, and Zhipeng Cai. Task assignment for hybrid scenarios in spatial crowdsourcing: A q-learning-based approach. *Applied Soft Computing*, 131:109749, 2022.

[58] Farah Ait Salaht, Frédéric Desprez, and Adrien Lebre. An overview of service placement problem in fog and edge computing. *ACM Computing Surveys (CSUR)*, 53(3):1–35, 2020.

[59] Samodha Pallewatta, Vassilis Kostakos, and Rajkumar Buyya. Placement of microservices-based iot applications in fog computing: A taxonomy and future directions. *ACM Computing Surveys*, 55(14s):1–43, 2023.

[60] Liang Huang, Xu Feng, Cheng Zhang, Liping Qian, and Yuan Wu. Deep reinforcement learning-based joint task offloading and bandwidth allocation for multi-user mobile edge computing. *Digital Communications and Networks*, 5(1):10–17, 2019.

[61] Jiadai Wang, Lei Zhao, Jiajia Liu, and Nei Kato. Smart resource allocation for mobile edge computing: A deep reinforcement learning approach. *IEEE Transactions on emerging topics in computing*, 9(3):1529–1541, 2019.

[62] S Kumar, P Tiwari, and M Zymbler. Internet of things is a revolutionary approach for future technology enhancement: a review. j. big data. 6, 111 (2019), 2019.

[63] Fan Wu, Feng Lyu, Huaqing Wu, Ju Ren, Yaoxue Zhang, and Xuemin Shen. Characterizing user association patterns for optimizing small-cell edge system performance. *IEEE Network*, 37(3):210–217, 2022.

[64] Haojun Li, Xiaoming Li, and Jingxin Xiao. Estimating gnss satellite clock error to provide a new final product and real-time services. *GPS Solutions*, 28(1):17, 2024.

[65] M Mohamed and N El-Saber. Toward energy transformation: Intelligent decision-making model based on uncertainty neutrosophic theory. *Neutrosophic Systems with Applications*, 9:13–23, 2023.

[66] Mainak Adhikari, Abhishek Hazra, and Sudarshan Nandy. Deep transfer learning for communicable disease detection and recommendation in edge networks. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 20(4):2468–2479, 2022.

[67] Utkarsh Rana, Srajan Saxena, and Shweta Sinha. Iot with edge computing: Data offloading strategies. 2024.

[68] Sina Shahhosseini, Arman Anzanpour, Iman Azimi, Sina Labbaf, DongJoo Seo, Sung-Soo Lim, Pasi Liljeberg, Nikil Dutt, and Amir M Rahmani. Exploring computation offloading in iot systems. *Information Systems*, 107:101860, 2022.

[69] Zhao Tong, Xiaomei Deng, Feng Ye, Sunitha Basodi, Xueli Xiao, and Yi Pan. Adaptive computation offloading and resource allocation strategy in a mobile edge computing environment. *Information Sciences*, 537:116–131, 2020.

[70] Hongchang Ke, Jian Wang, Lingyue Deng, Yuming Ge, and Hui Wang. Deep reinforcement learning-based adaptive computation offloading for mec in heterogeneous vehicular networks. *IEEE Transactions on Vehicular Technology*, 69(7):7916–7929, 2020.

[71] Siqi Zhang, Na Yi, and Yi Ma. A survey of computation offloading with task types. *IEEE transactions on intelligent transportation systems*, 2024.

[72] Haole Hou, Zhengyi Chai, Xu Liu, Yalun Li, and Yue Zeng. A task offloading algorithm using multi-objective optimization under hybrid mode in mobile edge computing. *Mobile Networks and Applications*, 29(5):1577–1593, 2024.

[73] Firdose Saeik, Marios Avgeris, Dimitrios Spatharakis, Nina Santi, Dimitrios Dechouniotis, John Violos, Aris Leivadeas, Nikolaos Athanasopoulos, Nathalie Mitton, and Symeon Papavassiliou. Task offloading in edge and cloud computing: A survey on mathematical, artificial intelligence and control theory solutions. *Computer Networks*, 195:108177, 2021.

[74] Suzhi Bi and Ying Jun Zhang. Computation rate maximization for wireless powered mobile-edge computing with binary computation offloading. *IEEE Transactions on Wireless Communications*, 17(6):4177–4190, 2018.

[75] Abednego Acheampong, Yiwen Zhang, Xiaolong Xu, and Daniel Kumah. A review of the current task offloading algorithms, strategies and approach in edge computing systems. *Computer Modeling in Engineering & Sciences*, 134(1):35, 2023.

[76] Quoc-Viet Pham, Tuan Leanh, Nguyen H Tran, Bang Ju Park, and Choong Seon Hong. Decentralized computation offloading and resource allocation for mobile-edge computing: A matching game approach. *IEEE Access*, 6:75868–75885, 2018.

[77] Ali Shakarami, Mostafa Ghobaei-Arani, and Ali Shahidinejad. A survey on the computation offloading approaches in mobile edge computing: A machine learning-based perspective. *Computer Networks*, 182:107496, 2020.

[78] Issam El Naqa and Martin J Murphy. What is machine learning? In *Machine learning in radiation oncology: theory and applications*, pages 3–11. Springer, 2015.

[79] Nadine Abbas, Wissam Fawaz, Sanaa Sharafeddine, Azzam Mourad, and Chadi Abou-Rjeily. Svm-based task admission control and computation offloading using lyapunov

optimization in heterogeneous mec network. *IEEE Transactions on Network and Service Management*, 19(3):3121–3135, 2022.

[80] Sihua Wang, Mingzhe Chen, Walid Saad, and Changchuan Yin. Federated learning for energy-efficient task computing in wireless networks. In *ICC 2020-2020 IEEE international conference on communications (ICC)*, pages 1–6. IEEE, 2020.

[81] Mohammad Zolghadri, Parvaneh Asghari, Seyed Ebrahim Dashti, and Alireza Hedayati. Dynamic task offloading for iot-fog-cloud systems: a network traffic-aware decision tree approach. *Computing*, 107(4):1–30, 2025.

[82] Fernando AM Trinta, Masum Z Hasan, and Jose N de Souza. Enhancing offloading systems with smart decisions. *Adaptive Monitoring, and Mobility Support. hindawi*, pages 1–18, 2019.

[83] MP Sujatha and S Niresh Kumar. Anomaly detection of industrial control systems based on transfer learning. *International journal of health sciences*, 6(S4):5782–5792, 2022.

[84] Karanam venkata Sai. Security issues and defensive approaches in deep learning frameworks.

[85] Sheharyar Khan, Zheng Jiangbin, Muhammad Irfan, Farhan Ullah, and Sohrab Khan. An expert system for hybrid edge to cloud computational offloading in heterogeneous mec–mcc environments. *Journal of Network and Computer Applications*, 225:103867, 2024.

[86] Mengyuan Zhang, Juan Fang, Ziyi Teng, Yaqi Liu, and Shen Wu. Joint dnn partitioning and task offloading based on attention mechanism-aided reinforcement learning. *IEEE Transactions on Network and Service Management*, 2025.

[87] Chenyi Yang, Xiaolong Xu, Muhammad Bilal, Yiping Wen, and Tao Huang. Deep-deterministic-policy-gradient-based task offloading with optimized k-means in edge-computing-enabled iomt cyber-physical systems. *IEEE Systems Journal*, 17(4):5195–5206, 2023.

[88] Ihsan Ullah and Hee Yong Youn. Task classification and scheduling based on k-means clustering for edge computing. *Wireless Personal Communications*, 113(4):2611–2624, 2020.

[89] Anirudh Yadav, Prasanta K Jana, Shashank Tiwari, and Abhay Gaur. Clustering-based energy efficient task offloading for sustainable fog computing. *IEEE Transactions on Sustainable Computing*, 8(1):56–67, 2022.

[90] Md Sajjad Hossain, Cosmas Ifeanyi Nwakanma, Jae Min Lee, and Dong-Seong Kim. Edge computational task offloading scheme using reinforcement learning for iiot scenario. *ICT Express*, 6(4):291–299, 2020.

[91] Chen Chen, Yuru Zhang, Zheng Wang, Shaohua Wan, and Qingqi Pei. Distributed computation offloading method based on deep reinforcement learning in icv. *Applied Soft Computing*, 103:107108, 2021.

[92] Manoj Penmetcha and Byung-Cheol Min. A deep reinforcement learning-based dynamic computational offloading method for cloud robotics. *IEEE Access*, 9:60265–60279, 2021.

[93] Nanliang Shan, Xiaolong Cui, and Zhiqiang Gao. "drl+ fl": An intelligent resource allocation model based on deep reinforcement learning for mobile edge computing. *Computer Communications*, 160:14–24, 2020.

[94] Hongchang Ke, Jian Wang, Lingyue Deng, Yuming Ge, and Hui Wang. Deep reinforcement learning-based adaptive computation offloading for mec in heterogeneous vehicular networks. *IEEE Transactions on Vehicular Technology*, 69(7):7916–7929, 2020.

[95] Muhammad Usman Hadi and Ahmed Bashir Abbasi. Optimizing uav computation offloading via mec with deep deterministic policy gradient. *Authorea Preprints*, 2023.

[96] Xiaoyan Zhao, Ming Li, and Peiyan Yuan. An online energy-saving offloading algorithm in mobile edge computing with lyapunov optimization. *Ad Hoc Networks*, 163:103580, 2024.

[97] Vandna Rani Verma, Anshul Verma, Vishnu Sharma, DK Nishad, et al. Energy-proficient task offloading in multi-access edge computing using lyapunov optimization. *International Energy Journal*, 25, 2025.

[98] Yixin Liu, Shaoling Liang, Kunlun Wang, Wen Chen, Yonghui Li, and George K Karagiannidis. Distributed massive mimo-aided task offloading in satellite-terrestrial integrated multi-tier vec networks. *IEEE Transactions on Vehicular Technology*, 2025.

[99] Latif U Khan, Maryam Alghfeli, Mohsen Guizani, Nasir Saeed, and Sami Muhaidat. Tcs: A joint task offloading, communication, and sensing framework for vehicular metaverse. *IEEE Internet of Things Journal*, 2025.

[100] Yuan Chai, Xiao-Jun Zeng, Quan Chen, and Lianglun Cheng. Stackelberg game-based joint computing resource allocation and task offloading method in edge computing. *IEEE Transactions on Vehicular Technology*, 2025.

[101] Ying Duan and Chunmao Jiang. Binary task offloading strategy for cloud robots using improved game theory in cloud-edge collaboration. *Journal of supercomputing*, 80(10), 2024.

[102] DeGan Zhang, Shuai Li, Jie Zhang, and Ting Zhang. Novel offloading approach of computing task for internet of vehicles based on particle swarm optimization strategy. *Cluster Computing*, 28(3):156, 2025.

[103] Jargis Ahmed. Task offloading and execution in edge-cloud collaborative network using genetic algorithm. *GUB Journal of Science and Engineering*, 9(1), 2022.

[104] Qianhua Luo, Bo Xie, Jiahuan Wang, Jiaqi Shuai, and Haixia Cui. A hybrid approach to task offloading optimization: integrating hybrid whale genetic algorithm and reinforcement learning. *The Computer Journal*, page bxaf033, 2025.

[105] Li Ma, Peng Wang, Chunlai Du, and Yang Li. Energy-efficient edge caching and task deployment algorithm enabled by deep q-learning for mec. *Electronics*, 11(24):4121, 2022.

[106] B Vijayaram and V Vasudevan. Energy aware intelligent task offloading in mobile edge computing using hyper heuristics. *Engineering Research: Perspectives on Recent Advances Vol. 4*, pages 73–108, 2025.

[107] Aravindh Krishnamoorthy, Hossein Safi, Othman Younus, Hossein Kazemi, Isaac NO Osahon, Mingqing Liu, Yi Liu, Sina Babadi, Rizwana Ahmad, Asim Ihsan, et al. Optical wireless communications: enabling the next generation network of networks. *IEEE Vehicular Technology Magazine*, 2025.

[108] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3):279–292, 1992.

[109] Carmen Chicone. Stability theory of ordinary differential equations. In *Encyclopedia of Complexity and Systems Science*, pages 1–31. Springer, 2013.

[110] Nidhi Kumari and Prasanta K Jana. A metaheuristic-based task offloading scheme with a trade-off between delay and resource utilization in iot platform. *Cluster computing*, 27(4):4589–4603, 2024.

[111] Rachid Bechar, Mounir Tahar Abbes, Freha Mezzoudj, and Ladjel Bellatreche. On formal modeling and validation of wireless sensor network protocols. *Wireless Personal Communications*, 114(4):2855–2888, 2020.

[112] Shuai Yu. *Multi-user computation offloading in mobile edge computing*. PhD thesis, Sorbonne université, 2018.

[113] S Kumar, P Tiwari, and M Zymbler. Internet of things is a revolutionary approach for future technology enhancement: a review. j. big data. 6, 111 (2019), 2019.

[114] Ahmad Zendebudi and Salimur Choudhury. Designing a deep q-learning model with edge-level training for multi-level task offloading in edge computing networks. *Applied Sciences*, 12(20):10664, 2022.