Computers, Materials & Continua

Tech Science Press

https://doi.org/10.32604/cmc.2026.077465

ARTICLE

Smart Offloading of IoT Big Data for Network Resources Optimization

Afzal Badshah[1,*], Mona Eisa[2], Omar Alghushairy[2], Riad Alharbey[2], Manal Linjawi[3] and Ali Daud[4,*]

[1]Department of Software Engineering, University of Sargodha, Sargodha, Punjab, Pakistan
[2]Department of Information Systems and Technology, College of Computer Science and Engineering, University of Jeddah, Jeddah, Saudi Arabia
[3]Department of Computer Science and Artificial Intelligence, College of Computer Science and Engineering, University of Jeddah, Jeddah, Saudi Arabia
[4]Faculty of Resilience, Rabdan Academy, Abu Dhabi, United Arab Emirates

*Corresponding Authors: Afzal Badshah. Email: afzalbadshahkhattak@gmail.com; Ali Daud. Email: alimsdb@gmail.com

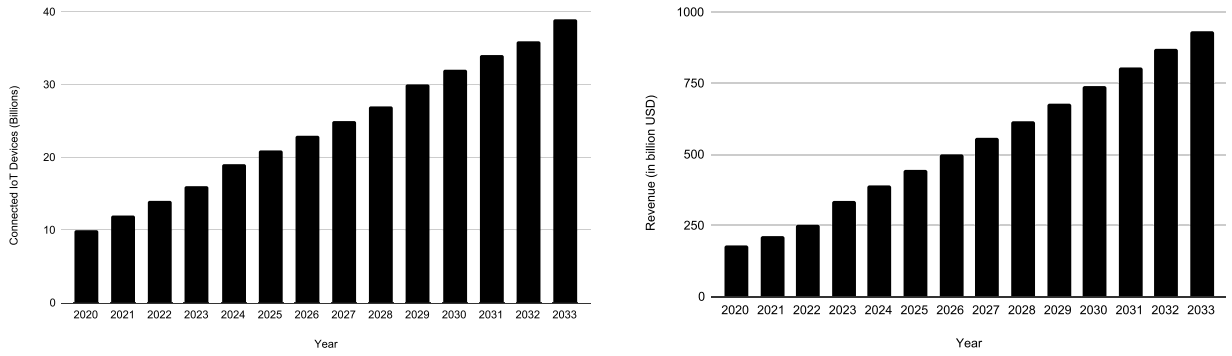
Received: 09 December 2025; Accepted: 28 February 2026; Published: 08 May 2026

ABSTRACT: The Internet of Things (IoT) devices generate massive data that leads to network congestion, propagation delays, and suboptimal resource allocation. Traditional Cloud Computing (CC) offers scalable resources required for that data; however, it has a long delay and communication overhead. On the other hand, Edge Computing (EC) guarantees low latency but has limited computational capacity. In this paper, we propose an intermediate paradigm, Regional Computing (RC), combined with a Fuzzy Logic System (FLS) for dynamic, multi-criteria offloading across edge, regional, and cloud. The FLS takes task size, cost, and computational demand as input metrics. It uses a rule-based inference engine to select the optimal offloading tier for each task. We created real-time data using an Arduino UNO R4 and ran it in our Python custom-built simulator, *RegionalEdgeSimPy*. It is specially designed to simulate IoT environments. Experimentation results show that the proposed strategy reduces average network latency by 50% as compared to CC offloading. The model also reduces costs by 30% in comparison with EC or CC. The framework enhances scalability and responsiveness in IoT big data applications and is representative of a practical solution for real-world deployment.

KEYWORDS: Internet of Things; computer networks; offloading; cloud computing; edge computing; regional computing

1 Introduction

The Internet of Things (IoT) has been spreading aggressively as a ubiquitous part of day-to-day life, with billions of IoT devices being connected. The number of connected devices has seen a sudden increase in the past decade and is forecasted to increase to 39 billion by 2033 as shown in Fig. 1a. Alongside this, the revenue from IoT is also forecasted to increase from 181.5 billion in 2020 to 934.2 billion in 2033 [1] as shown in Fig. 1b. The above circumstances lead to an unprecedented amount of data in terms of volume, speed, and variety for IoT [2]. With this high volume of data, a massive potential for data-driven automation has been produced. However, it also poses a powerful challenge to traditional networks and computing systems. Transmission as well as processing of this huge data imposes a severe overload on networks, introducing long delays as well as overworking the servers [3].



(a) Projected Growth of IoTDevices: 2020–2033

(b) Projected IoT Revenue Growth: 2020–2033

Figure 1: Projection of IoT device and revenue growth over time.

To handle this ever-growing flow of IoT data, Cloud Computing (CC) and Edge Computing (EC) two paradigms that gaining widespread usage. Both have their own advantages and drawbacks. CC has centralized scalable computational resources; therefore, it is a good fit to handle heavy workloads. However, the process of offloading data into distant CC data centres raises delays and bandwidth costs [4], which becomes impractical for time-sensitive applications. On the other hand, EC processes the data at or near the source, reducing communication delays and network congestion [5]. However, these servers have limited computational capacity [6] and cannot handle big data. Even though CC and EC each address part of the problem, neither alone could be efficient for handling that growing IoT data with low latency and high computational requirements. Regional Computing (RC) aims to bridge the gap between CC and EC [7]. It positions servers within a user's region, making them closer than remote CC centers while providing more resources than EC [8]. Due to this proximity, RC has low latency, similar to EC, but scalable resources. The RC layer can offload heavy workloads from the EC and decrease dependency on distant CC centers by managing portions of the tasks locally. This layered architecture of EC, RC, and CC resources provides a formidable solution for a large IoT environment. The main challenge is determining at which tier each task should be handled to achieve an optimal balance between latency and processing power [9].

In a multi-tier environment, finding the most suitable execution layer for each IoT task to achieve a balance between latency and processing power is a challenging problem. Many factors must be taken into consideration, such as the urgency and size of the task, network bandwidth, delay, along with the current load and availability of EC, RC, and CC resources. These metrics are dynamic and hinder the use of any fixed offloading policy. Therefore, there is an urgent requirement for a mechanism that makes a real-time decision based on multi-criteria and variable conditions to offload each task to the best computing paradigm. Fuzzy Logic (FL) offers a structured approach for this decision problem in uncertain conditions [10]. Unlike traditional rigid binary reasoning with hard thresholds, Fuzzy Logic System (FLS) can support gradual truth with many input variables in a flexible manner. Instead of treating values (e.g., network delay, task size, and server utilization) in quantitative "low," "medium," and "high" scales, in FL, these are represented in a qualitative manner. A series of fuzzy if-then rules takes a comprehensive analysis of all these variables as input for indirectly determining the most appropriate computing layer for processing a task, hence approximating human reasoning with a set of compromises between various parameters [11]. Depending on how conditions change, the FL controller dynamically adjusts decisions in real-time. This dynamic strategy works perfectly in a dynamic IoT setup [12,13]. This is unlike most present-day approaches for offloading decisions, which are usually rigid with fixed policies and naive rules. Any traditional rule-based systems

are not dynamic, practically unable to support dynamic conditions of a real-world IoT setup, which in turn results in underutilized/overutilized resources in most situations.

In this paper, we present a smart offloading framework that integrates the RC paradigm with an adaptive FL controller to enable dynamic distribution of IoT workloads across EC, RC, and CC layers. This is evaluated using real-time data generated by Arduino and a custom simulation platform (RegionalEdgeSimPy) designed to emulate multi-tier infrastructure conditions. The key contributions of this work are as follows:

- We present the RC as a new intermediary paradigm in processing IoT big data with scalability similar to that of CC but with latency comparable to that of EC.
- We propose an FL-based offloading strategy that gets the task size, cost, CPU demand, and storage demand, and dynamically selects the optimal execution tier.
- We develop RegionalEdgeSimPy a custom Python simulator integrating real-time data from Arduino IoT devices to evaluate offloading performance across EC, RC, and CC layers.

The rest of the paper is organized as follows: Section 2 reviews the related literature. Section 3 describes the proposed methodology and its working. Section 4 evaluates the proposed algorithm, which is further followed by Section 5 that presents a comparative study of edge, regional, and cloud computing. Finally, Section 6 summarizes the article and provides future directions.

2 Related Work

The growth of IoT technology has disrupted various sectors by offering a wide range of connected devices and platforms that result in huge amounts of unstructured data. IoT sensors, wearable technology, mobile apps, and immersive tech, such as virtual reality and augmented reality, are the major contributors to the accelerated adoption of IoT technology [14].

2.1 Traditional Offloading Techniques

The increase in IoT data and emerging Artificial Intelligence (AI), 6G, and mobile technology has also increased the computational complexity. CC has been widely used to handle this data due to its scalable resources. Though the CC is the core of big data management, it suffers from serious barriers (e.g., delayed response time and bandwidth limitation) [15]. These issues become more critical in continuous flow-based heterogeneous handling and managing of the raw sensor streams of IoT elements. Its scalability suffers a lot from integration complexity with other systems and a deficiency in efficient resource utilization in every case that demands real-time processing of information [16]. To address these challenges, EC and Mobile Edge Computing (MEC) emerged. Because these nodes handle the data at or closer to their generation points, the latency is reduced considerably, making them much more responsive. For example, EC has optimized resource utilization and task performance, especially in a real-time environment [17]. Similarly, MEC also handles the data at the edge of the wireless network, suitable for real-time applications, like industrial IoT systems or independent operation [18]. The performance of such systems are further optimized by cooperative edge networks; however, resource scalability is still the major challenge [19]. Fog Computing (FC) provides a near-edge layer to manage the data, minimizing the network congestion. It also serves as a bridge between the edge and cloud for real-time analytics applications in different IoT scenarios (e.g., predictive maintenance, smart city infrastructure management, among others) [20]. For example, fog systems have shown the ability to integrate local storage with embedded intelligence for rapid decisions [21]. However, FC has resource scalability challenges.

Furthermore, small data centres located near data sources, called cloudlets, have been investigated as a complementary strategy to further optimize localized data processing. Cloudlets have benefits for the

localized management of IoT big data but suffer resource limitations at peak times [22]. Hybrid models utilize the strengths of both FC and CC, providing a more balanced strategy for processing and storing data in dynamic IoT environments.

2.2 Fuzzy Logic Offloading

The concept of FL in computation offloading and task scheduling has attracted interest in the past few years for the optimization of IoT applications on edge, fog, and cloud architectures. Many studies have investigated its integration with advanced algorithms to address energy consumption, task prioritization, and resource allocation issues.

Hybrid architectures, including edge, fog, and cloud layers, have been explored by different studies. For example, a fuzzy computation offloading scheme was proposed to optimize task allocation across these layers, which improved the makespan, energy consumption, and cost [23]. Similarly, a fuzzy task offloading technique (FBOT) dynamically allocated and computed data-intensive tasks in hierarchical edge, fog, and cloud, which reduced waiting times and increased efficiency in task completion by 30%. Furthermore, another hybrid computing architecture improved the Quality of Service (QoS) with an FL offloading approach that remarkably minimized energy consumption and cost [23]. Energy efficiency is another important direction explored in various IoT offloading strategies. In [14], an energy-efficient fuzzy data offloading scheme was developed with an adaptive scheduling strategy under a four-tier architecture to minimize the energy consumption and delay. Another approach proposed the use of FL combined with Reinforcement Learning (RL) for task offloading optimization in VFC, focusing on energy limitations in Road Side Units (RSUs), and it attained better energy efficiency compared to conventional techniques [24]. Recently, another Vehicle-2-Vehicle (V2V) offloading approach utilized FL for dynamic task prioritization and edge node selection while dealing with highly mobile and dynamic network topologies with reduced delays [25].

FL has been successfully adopted for prioritization and scheduling in various IoT contexts. For instance, a fuzzy MEO has optimally managed the resources in IoT applications and outperformed the benchmark in task failures and WLAN delay [26]. Another research proposed a FL technique for task prioritization in FC, taking a best-fit selection multi-population algorithm to minimize service latency in delay-sensitive applications. That results in the considerable improvement of waiting time and service latency [27]. In another study, a computation peer offloading scheme for MEC was introduced, which enhanced resource sharing and reduced task failure rates by more than 80%. FL has also been used in specific domains such as Internet of Vehicles (IoV) and industrial IoT. A Vehicle-2-Everything (V2X) fuzzy offloading scheme, which evaluated vehicles for computational power and link quality, decreased the task completion time by 37.5% compared to existing methods [28]. In Industrial Cyber-Physical Systems (CPS), a recommendation system based on FL guided the distribution of data analysis tasks between cloud and edge layers, where it attended to the time-sensitive requirements effectively [29]. Further, another work has developed a high-reliability task offloading mechanism for PIoTs, which enhances the success rate and decreases power consumption significantly [30].

Collaborative inference and context-aware decisions have also been pursued using FL. As an example, a FL controller in wireless sensor networks determined inference locations depending on energy availability and network bandwidth, reducing energy costs by about 50% [31]. Another method combined FL with deep learning techniques, such as CART, to optimize the time complexity of the inference, which enhanced the task processing time and tolerance to latency [32]. Extensive simulations over various platforms have validated the effectiveness of FL-based approaches. For instance, energy gains and improvements in QoS levels were highlighted through simulations using iFogSim [14]; while the performance of the fuzzy orchestrator, using

EdgeCloudSim, in improving service times and reducing WLAN delays has been validated [26]. A Python fog simulator called YAFS was used to demonstrate gains in MEC environments pertaining to task processing time, energy consumption, and server utilization [32].

2.3 Reinforcement Learning Offloading

RL has now emerged as one of the most powerful adaptive frameworks that can learn the optimal offloading strategies by continuously interacting with dynamic computing environments [33]. While considering multiple state variables, RL schedulers can make intelligent decisions for minimizing long-term cost and latency while adapting to changes in resource availability, workload fluctuations, and network delays.

In usual RL offloading frameworks, the task scheduling problem is modelled as a Markov Decision Process (MDP), where an agent observes the system state for instance, CPU utilization, network delay, and task priority, and selects an action that decides the offloading destination of the task. The environment provides a reward signal based on defined objectives such as delay minimization, energy efficiency, or cost optimization. Over time, an agent learns a policy that maps states to optimal actions [34,35]. Initial works utilized classical Q-learning and DQN to optimize the efficiency of task allocation and reduce delay [36,37]. These approaches indeed proved promising, but they also started to face limitations in large environments with high-dimensional state and action spaces. These challenges brought forward the proposal of actor-critic methods, along with multi-agent RL strategies [38].

Multi-agent RL enables decentralized decisions with each agent acting on local observation and contributing to a globally coordinated policy [39,40]. In latency-sensitive tasks (e.g., Autonomous Vehicles (AVs) and Unmanned Aerial Vehicles (UAVs)), multi-agent models have reported enhancements in both task latency and energy optimization [41,42]. Recent developments further augmented it with innovations in the form of attention mechanisms, graph-based state representations, and continuous control techniques. These works enable fine-grained scheduling actions at the server level and dynamic bandwidth adjustment. Informing reward shaping from task priority and congestion metrics has also improved the alignment to system-level objectives.

Despite these benefits, as shown in Table 1, several drawbacks still exist. Concerns such as high training overhead, vulnerability to non-stationary environments, and low generalizability in deployment scenarios are the major issues. Works on transfer learning, distillation of policies, and hybrid models with rule-based components hint at mitigations of the aforementioned limitations.

Table 1: Comparison of existing offloading approaches and the proposed method.

Approach	Paradigm	Objectives	Limitations
CC [15,16]	Centralized cloud	Scalable storage and analytics	High latency, congestion, poor real-time support
EC [17,18]	Device-local edge	Ultra-low latency, immediate responsiveness	Limited resources, scalability issues with large IoT networks
FC & MEC [20,21]	Intermediate tier	Local analytics, reduced bandwidth, mobility support	Synchronization and coordination overhead
Cloudlets [22,43]	Small data centers	Balanced load, improved responsiveness	Limited resources during peak, integration complexity

(Continued)

Table 1 (continued)

Approach	Paradigm	Objectives	Limitations
FL [14,24,25]	Multi-tier architectures	Energy-efficient, mobility-aware, reduced delay	Complexity, higher overhead
RL [36,38,41,42]	Multi-tier (EC/FC/CC)	Adaptive scheduling, latency and energy savings	High training cost, non-stationary performance
Proposed Method	Multi-tier (EC, Regional & CC)	Multi-metric optimization (latency, cost, network demand); scalable regional aggregation	Novel tier, further real-world deployment needed

3 Methodology

Our proposed system is based on a four-layer architecture (IoT, EC, RC, and CC) as shown in Fig. 2. These layers are integrated with a fuzzy logic engine to optimize task offloading. We explain below the role of each layer and introduce the fuzzy logic offloading system along with the algorithm.

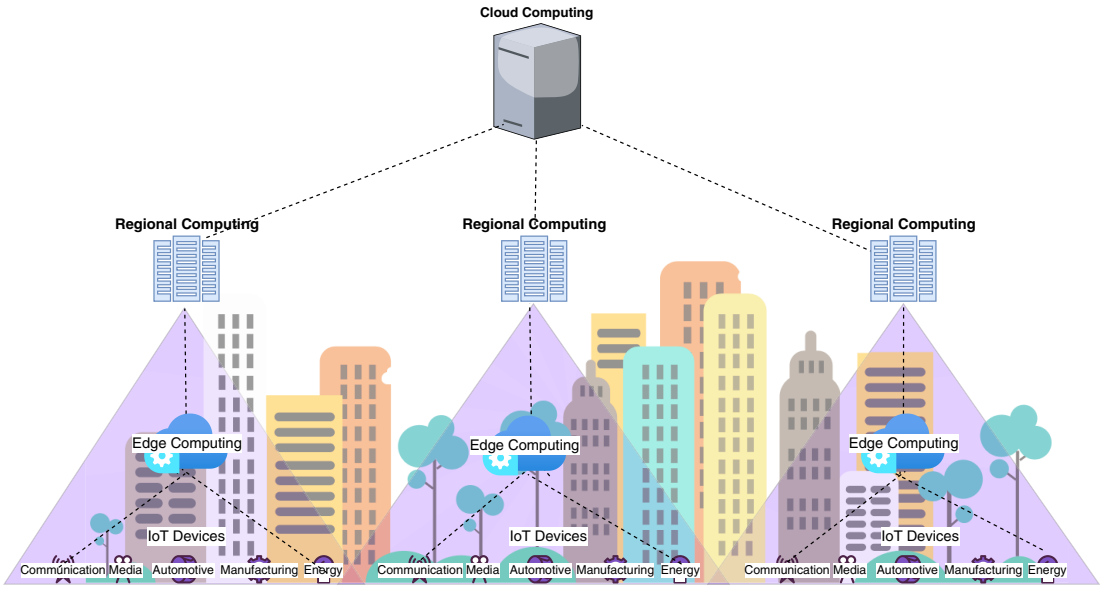


Figure 2: Structure of the proposed data offloading model.

3.1 IoT Layer

The IoT sensing layer records constant environmental data and generates tasks to be processed further. Our experimental setup comprises sensor-enabled Arduino UNO R4 WiFi (Rev4) boards that collect the real-time readings and send them to the upper layers for the required processing. The measured task sizes and associated metrics enable the possibility of evaluating offloading behavior under realistic hardware conditions. The devices have only limited computing power and battery capacity. Therefore, they only do the minimum amount of processing locally and offload most of the workload to the edge layer. This sensor-generated real-time flow of information represents the very initial point of the offloading pipeline and directly feeds inputs into the fuzzy decision module.

3.2 Edge Layer

The edge computing layer comprises resources placed close to the IoT devices, receiving raw data tasks and performing local processing to achieve low latency and reduce bandwidth consumption. It handles real-time and delay-sensitive tasks by acting on them immediately, which lowers response time and relieves network pressure. As the first computational tier, it is suitable for tasks that are small or light in terms of CPU and storage requirements, and the fuzzy logic rules keep such tasks at the edge whenever possible. This prevents unnecessary traffic from moving to higher layers. When the workload exceeds the edge's capacity, tasks are passed on to the regional layer. The edge nodes also collect information from multiple IoT devices and observe network conditions, providing useful context for the offloading decision process.

3.3 Regional Layer

The regional computing layer operates between the edge and the cloud, typically represented by a stronger server or a small cluster located in a nearby data center. It handles aggregated processing and analytics that exceed the capabilities of edge nodes while avoiding the overhead of cloud offloading. With greater computational capacity, this layer efficiently manages medium-sized workloads. In the process of offloading, it acts as a mid-tier destination for tasks that are too large or demanding for the edge but do not require full resources from the cloud. The fuzzy logic engine forwards those medium complexity tasks to the regional tier, reducing the data volume towards the cloud.

3.4 Cloud Layer

CC is a high-capacity data centers that provide scalable computing resources for large workload processing and long term storage. It handles the massive workloads such as large-scale analytics, historical data processing, or training machine learning models on IoT data. Once the fuzzy logic engine marks a task as massive to handle in the lower tiers (even though it may incur higher latency and network overhead). The task is offloaded to the cloud.

3.5 Fuzzy Logic System

FLS takes key performance parameters as inputs, and it has an offloading decision as output to route tasks intelligently across the edge, regional, and cloud layers. The reason for using a FLS is to handle uncertainty and variation that can occur within dynamic IoT environments. It uses linguistic categories like low, medium, and high, along with a rule-based inference mechanism. The four metrics (task size, processing cost, CPU demand, and storage demand), which are most influential, are considered as input parameters. These inputs are converted into fuzzy values using their respective membership functions. A set of if–then rules connects a combination of input states to a recommended execution layer (edge, regional, or cloud). The rules are evaluated by the inference engine for every incoming task, followed by a defuzzification step to convert the fuzzy output into a clear decision. It continuously runs so that the system can adapt to the fluctuating workload and network conditions. More information on the design of a fuzzy controller, including the definition of membership functions and rules, is provided in Appendix A.

3.5.1 Task Size

Task size is one of the key factors in the decision process to optimize the data offloading. The increase in task size results in the congestion of public networks and overutilization of data centers, leading to long delays and increased costs.

The probability of network congestion (Con_{Prob}) may be expressed as:

$$Con_{Prob} \approx \frac{T_{total}}{B_{avail}} \tag{1}$$

where Con_{Prob} is the probability of congestion, T_{total} is the overall network traffic, i.e., task size, and B_{avail} is the available bandwidth.

As network workload increases, the bandwidth reduces:

$$B_{avail} \approx \frac{1}{T_{total}} \tag{2}$$

Bandwidth congestion leads to higher network delay.

$$delay \propto \frac{1}{B_{avail}} \tag{3}$$

These relations show the importance of managing task size to efficiently use the network and minimize delays.

In our system, the task size varies from 1000 to 2,000,000 KB, considering that data are produced directly by the IoT devices. This interval can be represented in the fuzzy logic controller by triangular membership functions, where the categories are divided by gradual transition regions. In this work, the interval that represents small tasks ranges from 1000 KB up to approximately 500,000 KB, medium tasks range from approximately 300,000 to 1,500,000 KB, and large tasks range from 1,000,000 KB to approximately 2,000,000 KB. Also, regions of overlap allow interpreting tasks with variable degrees of membership; this allows smooth fuzzification and provides a more realistic view of how task sizes behave in practice.

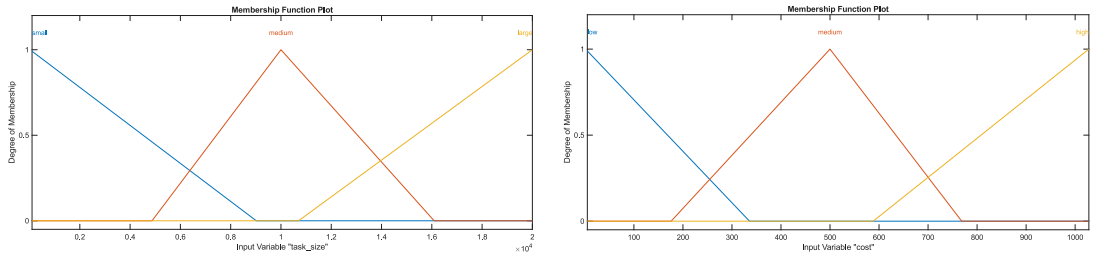
The formula for the membership functions is as follows:

$$\text{small}(x) = \max\left(0, \min\left(1, \frac{x - 1000}{500000}\right)\right) \tag{4}$$

$$\text{medium}(x) = \max\left(0, \min\left(1, \frac{x - 300000}{1200000}\right)\right) \tag{5}$$

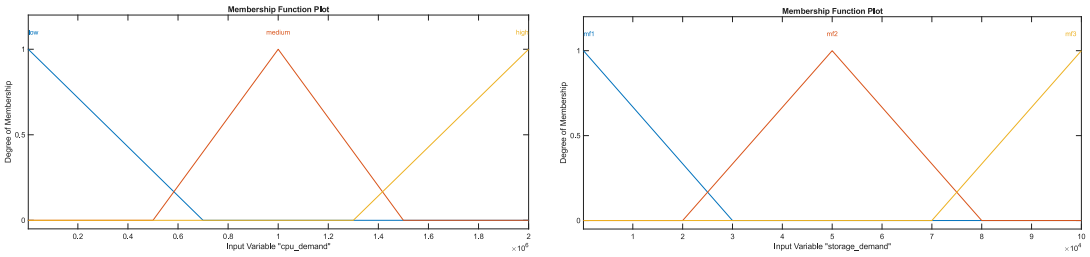
$$\text{large}(x) = \max\left(0, \min\left(1, \frac{x - 1000000}{1000000}\right)\right) \tag{6}$$

These equations are used to evaluate the degree to which a given task size fits into one of the categories (small, medium, or large) as shown in the membership function Fig. 3a.



(a) Task Size Membership Functions (b) Cost Membership Functions

Figure 3: (Continued)



(c) CPU Demand Membership Functions (d) Storage Demand Membership Functions

Figure 3: Membership functions for all input parameters.

Task size also plays a central role in the determination of the execution layer. Generally, small loads are suitable for edge processing, medium tasks would work best on the regional layer, and large tasks are routed to the cloud. This behavior is encoded within the fuzzy rule that evaluates task size considering CPU, network, and storage demands. Small tasks are kept at the edge during low demands, and for medium tasks, when demands are at a modest level, they are forwarded to the regional tier; large tasks go directly to the cloud when demands are high. It does so by continuously redistributing each task to the most suitable computing layer with respect to not only the size of the tasks but also the cost, processing load, storage requirements, and network conditions.

The system calculates the task size and evaluates its membership values to decide the appropriate computing paradigm. For example, if a task has a size of 500,000 KB, then the membership function of *small* would give it a value of 0.5, indicating that the task fits relatively in the small category. The fuzzified value is fed as input into the fuzzy inference system and computes the final offloading decision. The FL engine evaluates the combined effect of all input parameters, task size, cost, CPU demand, and storage demand to find the most suitable processing location.

3.5.2 Cost

Cost is another vital factor affecting the decision process for optimal offloading. The total cost can be represented as;

$$C_t = C_{tr} + C_p + C_{pr} + C_{col} \tag{7}$$

where C_t represents the overall cost, C_{tr} is the data transmission cost, C_p shows the propagation cost, C_{pr} shows the computational cost, and C_{col} reflects the cooling cost.

The cost for the transmission from IoT devices to RC servers can be formulated as:

$$C_{\text{trn, cp}} = \sum_{i=1}^{n} \sum_{j=1}^{m} \left(C_{\text{rate}} \cdot w_j + C_{\text{time, cp}} \cdot T_{\text{i,cp}} + C_{\text{energy, cp}} \cdot E_{\text{i,cp}} + C_{\text{B, cp}} \cdot B_{\text{i,cp}} + C_{\text{fixed, iot}} \right) \tag{8}$$

where C_{rate} is the cost per unit of transmitted data, and w_j expresses the volume of data generated by workload j. The variable $C_{\text{time,cp}}$ is the cost of every unit of transmission time, whereas $T_{\text{i,cp}}$ is the actual time needed for data transfer to the chosen computing layer. Similarly, $C_{\text{energy,cp}}$ means the cost per unit of utilized energy, while $E_{\text{i,cp}}$ is the actual expenditure of energy during data transmission. In addition, $C_{\text{B,cp}}$ is the per-unit cost of utilized bandwidth, where $B_{\text{i,cp}}$ defines actual bandwidth consumption. $C_{\text{fixed,iot}}$ is a fixed or constant cost involved in every data transfer from IoT devices to the computational infrastructure.

Similarly, the propagation cost is calculated as;

$$C_{\text{prop}} = \sum_{i=1}^{n} \left(C_{\text{dis, dev}} \cdot Dis_{i,\text{cp}} \right) \tag{9}$$

where $C_{\text{dis,dev}}$ is the cost per unit of distance from IoT devices to the computational layer, while $Dis_{i,\text{cp}}$ is the distance from IoT device i to the layer. This illustrates that with increased distance, the propagation cost also increases. We will define the cost in terms of processing cost and network cost. These are categorized into *low*, *medium*, and *high*. These categories reflect the affordability of processing and network resources; this dictates the choice of the location for processing. The processing cost is modeled as a fuzzy variable within a range of approximately 1.4 $ to 1028 $, based on values derived from Arduino-generated data. Triangular membership functions show the uncertainty in the estimation of this cost, as shown in Fig. 3b. In this scheme, lower costs fall roughly within the range of 1.4 $ to 300 $, medium costs span the range from about 200 $ to 700 $, and higher costs extend from nearly 600 $ up to 1028 $. The overlapping regions allow each cost value to be associated with a degree of membership across the three categories. This allows the fuzzy logic system to evaluate the comparative feasibility of executing a task and to choose the most appropriate processing layer accordingly.

The formulae for the membership functions are given as:

$$\text{low}(x) = \max\left(0, \min\left(1, \frac{300 - x}{298.6}\right)\right) \tag{10}$$

$$\text{medium}(x) = \max\left(0, \min\left(1, \frac{x - 200}{500}, \frac{700 - x}{500}\right)\right) \tag{11}$$

$$\text{high}(x) = \max\left(0, \min\left(1, \frac{x - 600}{428}\right)\right) \tag{12}$$

These functions enable a smooth transition between low, medium, and high cost values, facilitating a flexible decision in the FLS.

3.5.3 CPU Demand

CPU demand is another key factor that decides the offloading of tasks. It is the required power for the computation of a particular task. We divide the demand into three kinds of states: *low*, *medium*, and *high*, representing different types of computation intensities of tasks, which could determine whether the task is processed locally or offloaded to another layer. The CPU demand is a fuzzy variable that lies within approximately 100,000 to 2,000,000 MIPS, showing the load each task exercises on the computational facilities. In representing this range within the fuzzy logic controller, triangular membership functions are applied, as shown in Fig. 3c. In this design, lower, moderate, and higher CPU demands broadly cover approximately 100,000 to 700,000 MIPS, about 500,000 to 1,500,000 MIPS, and from about 1,300,000 to 2,000,000 MIPS, respectively. Overlapping areas between all three provide a smooth mapping of the CPU value and yield a fuzzified degree of membership that the system will utilize in describing computational intensity during offloading.

The formulae for the membership functions are given by:

$$\text{low}(x) = \max\left(0, \min\left(\frac{x - 100000}{600000}, 1, \frac{700000 - x}{600000}\right)\right) \tag{13}$$

$$\text{medium}(x) = \max\left(0, \min\left(\frac{x - 500000}{500000}, 1, \frac{1500000 - x}{500000}\right)\right) \tag{14}$$

$$\text{high}(x) = \max\left(0, \min\left(\frac{x - 1300000}{700000}, 1, \frac{2000000 - x}{700000}\right)\right) \tag{15}$$

These enable a seamless transition between low, medium, and high CPU demands to enable FLS to make accurate decisions on the CPU requirements of the task.

This fuzzy rule links CPU demand to the offloading decision. When the required CPU load is low, the task can be handled at either the edge or the regional layer, as both tiers provide adequate computational capacity for lightweight workloads. When CPU demand reaches a moderate level, the regional layer becomes preferred because it offers higher scalability compared to the edge while avoiding the overhead of cloud. For high CPU requirement tasks, the FLS directs to the cloud, where high and elastic computing resources are available to manage these workloads.

3.5.4 Storage Demand

Like other factors, storage demand is a crucial factor in deciding on the computational layer. It is divided into three categories, *low*, *medium*, and *high*, in which each guides the decisions based on the available resources in every layer.

The storage demand is a fuzzy variable defined over a range of approximately 3000 to 600,000 KB, representing the amount of data a task needs to store or transfer. This range is expressed in the fuzzy controller through triangular membership functions, as explained in Fig. 3d. In this mapping, tasks with lower storage requirements span roughly 3000 to 30,000 KB, the ones requiring a fair amount of storage span a range of about 20,000 to 300,000 KB, and those that are more demanding range from about 200,000 to 600,000 KB.

The formulae for membership functions are given by:

The formulae for the membership functions are given by:

$$\text{low}(x) = \max\left(0, \min\left(\frac{x - 3000}{27000}, 1, \frac{30000 - x}{27000}\right)\right) \tag{16}$$

$$\text{medium}(x) = \max\left(0, \min\left(\frac{x - 20000}{80000}, 1, \frac{300000 - x}{200000}\right)\right) \tag{17}$$

$$\text{high}(x) = \max\left(0, \min\left(\frac{x - 200000}{400000}, 1, \frac{600000 - x}{400000}\right)\right) \tag{18}$$

These functions will allow smooth transitions between the low, medium, and high storage demand levels to make sure that FLS makes appropriate decisions about the storage requirements of a particular task.

3.6 Algorithm

Algorithm 1 addresses the complex decision of task offloading in heterogeneous computing environments. The tasks are scored considering multiple parameters, such as the number of devices, task size, processing cost, CPU demand, and storage requirements, all of which are converted to linguistic variables using membership functions.

Algorithm 1: Fuzzy logic-based offloading decision algorithm

Require: Task parameters: *task_size*, *cost*, *cpu_demand*, *storage_demand*
Ensure: Selected offloading scenario ∈ {**Edge**, **Regional**, **Cloud**}
Initialize fuzzy sets for all input variables:
    task_size ∈ {*small*, *medium*, *large*} (see Appendix A)
    cost ∈ {*low*, *medium*, *high*} Eqs. (10)–(12)
    cpu_demand ∈ {*low*, *medium*, *high*} Eqs. (13)–(15)
    storage_demand ∈ {*low*, *medium*, *high*} Eqs. (16)–(18)
Define fuzzy output variable: *scenario* ∈ {*edge*, *regional*, *cloud*}
Construct fuzzy rule base using expert knowledge (see Appendix A)
while new task data received **do**
    Read input parameters: *task_size*, *cost*, *cpu_demand*, *storage_demand*
    Fuzzification: Compute membership degrees for all input variables
    Inference: Evaluate fuzzy rules to determine the output fuzzy set for *scenario*
    Defuzzification: Use centroid method to compute crisp output score *s*
    Decision Mapping:
    if $s \leq$ threshold for *edge* (see Appendix A) **then**
        Scenario ← Edge
    else if *s* in range for *regional* (as per defined rules) **then**
        Scenario ← Regional
    else
        Scenario ← Cloud
    end if
    Return selected scenario for task offloading
end while

This algorithm receives real-time inputs regarding the task size, cost of processing, CPU demand, and storage demand. It will make use of predefined membership functions and rule bases to determine the most suitable computation paradigm. A fuzzy inference engine uses Mamdani-type reasoning and carries out defuzzification as the centroid method to translate these inputs into one of three scenarios: edge computing, regional computing, and cloud computing. The selected offloading scenario will be able to optimize the trade-off involving latency, cost, and resource utilization.

4 Evaluation

The proposed FL algorithm for offloading was evaluated for performance and cost optimization in the Edge, Regional, and Cloud stacks. Real-time data from the Arduino board and other IoT sensors was used as input for the simulator, RegionalEdgeSimPy, designed to simulate a multi-layer computing environment for analyzing offloading techniques. During experimentation, the research metrics such as size, cost, CPU usage, memory, and bandwidth were recorded. These values were then analyzed for processing time, usage, and total working costs for the proposed design to effectively understand how it reacts to changing scenarios and how our fuzzy logic controller allocates tasks to different available computing layers.

4.1 Experimental Setup

The experimental environment was composed of an integrated hardware and software platform where the data from IoT sensors was gathered in real-time using Arduino UNO R4 WiFi (Rev 4) boards

that constantly produced data streams for task size, cost, CPU requirements, storage, and network. The experimentation configuration is shown in Table 2. The data gathered in this experiment was structured in JSON format and transmitted, using serial communication, to a Python-based FLS class. On the other hand, the simulator was composed of three computing levels (Edge, Regional, and Cloud) with distinctive latency, processing, and monetary costs. The algorithm for FL in this experiment was coded using sk2fuzzy, a fuzzy-logic-based algorithm that processes input data from sensors, assesses fuzzy rules, and selects a suitable target computing platform for tasks. The code for this model can be accessed from GitHub [44]. To validate the efficiency of our proposed FLS offloading scheme, simulation of each task has been performed under three different paradigms, Edge, Regional, and Cloud, which form our internal benchmarks. These paradigms are practical offloading models that are commonly used in current IoT networks. These are performed under similar system configuration conditions, hence providing a fair and objective point of validation.

Table 2: Prototype parameters settings.

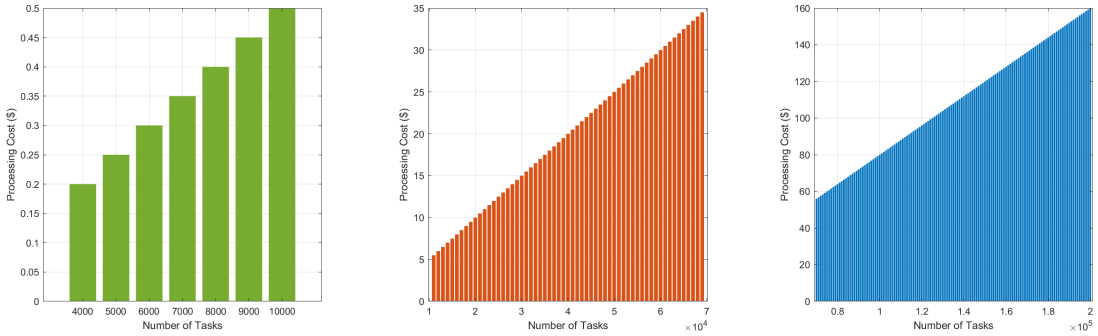
Parameter	Value	Parameter	Value
Quantity of Cloud Servers	1	Quantity of Regional Servers	1
Quantity of Edge Servers	1	Network Topology	LAN/WAN
Minimum IoT Devices	10	Maximum IoT Devices	2000
Initial Bandwidth (KB/s)	10,000	Task Size (min)	100 KB
Task Size (max)	200,000 KB		

4.2 Result Analysis

The results were used for analysis to test the performance of the algorithm in dynamic IoT devices. We did this by assessing decisions on task offloading in relation to task size, including CPU, network, storage, cost, and efficiency.

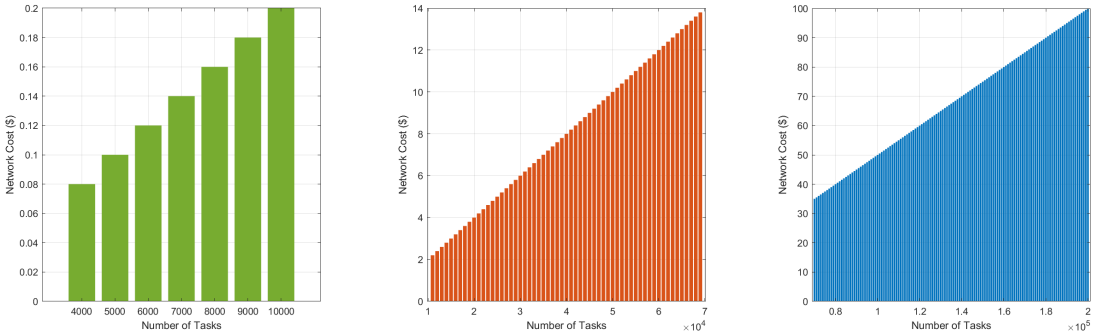
Fig. 4 shows the distribution of the processing cost across the edge, regional, and cloud paradigms. From Fig. 4a, we can observe that the edge layer has the lowest processing cost as it utilizes local resources. Therefore, it is best suited for lightweight delay-sensitive tasks. Fig. 4c shows that the cloud layer has the highest processing cost across the entire board due to its premium pricing and high overhead of infrastructure maintenance. On the other hand, Fig. 4b presents how the regional servers are positioned at an optimal point between these, offering reasonable costs with enough processing power to run mid to high intensity workloads. This further justifies the decisions from our model, which offloads most of the tasks to the regional tier to reduce costs without sacrificing performance. Therefore, the regional paradigm is an economic, scalable solution for cloud processing. This helps to further establish the intelligent task allocation strategy of the model in a dynamic vehicular environment.

Network cost of offloading tasks at the edge, regional, and cloud paradigms is shown in Fig. 5. From Fig. 5a, it can be observed that edge computing has the lowest network cost due to proximity to IoT devices and low transmission overhead. Its processing, however, has limited capacity that can support scalability. Similarly, Fig. 5b shows that the RC model has a moderate network cost (higher than edge but significantly less than cloud) and better scalability and resource provisioning. Cloud offloading results in the highest network cost due to increased transmission distance and multi-hop communication, as shown in Fig. 5c. Therefore, the regional paradigm provides a practical middle ground, having low-latency access with reduced transmission cost, while supporting larger workloads that edge computing alone cannot handle.



(a) Edge (b) Regional (c) Cloud

Figure 4: Processing cost distribution across Edge, Regional, and Cloud paradigms with increasing workloads.



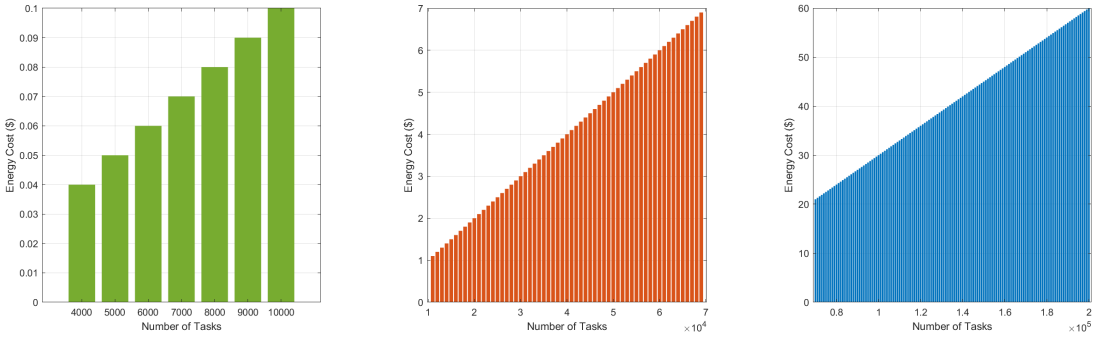
(a) Edge (b) Regional (c) Cloud

Figure 5: Network transmission cost across Edge, Regional, and Cloud paradigms as workload increases.

The energy cost of task offloading to Edge, Regional, and Cloud paradigms is shown in Fig. 6. As expected, energy cost increases with the number of tasks due to increased data transmission and computation. As we can see, cloud computing always results in the highest energy consumption, as shown in Fig. 6c. This is because longer communication distance, along with higher transmission power, are required to reach the usually centralized cloud servers that also process huge volumes of aggregated data. In contrast, edge computing consumes the least energy due to the proximity to the source devices, as shown in Fig. 6a. On the contrary, edge has limited processing capacity and is easily overutilized under heavy loads. The regional paradigm offers a practical balance, as shown in Fig. 6b. Its energy cost is higher than edge computing due to the slightly longer distance, but it is still much lower than the cloud computing paradigm. This tradeoff, along with improved scalability and task handling capability, positions the regional tier as an energy-efficient and operationally feasible choice in multtier IoT offloading scenarios.

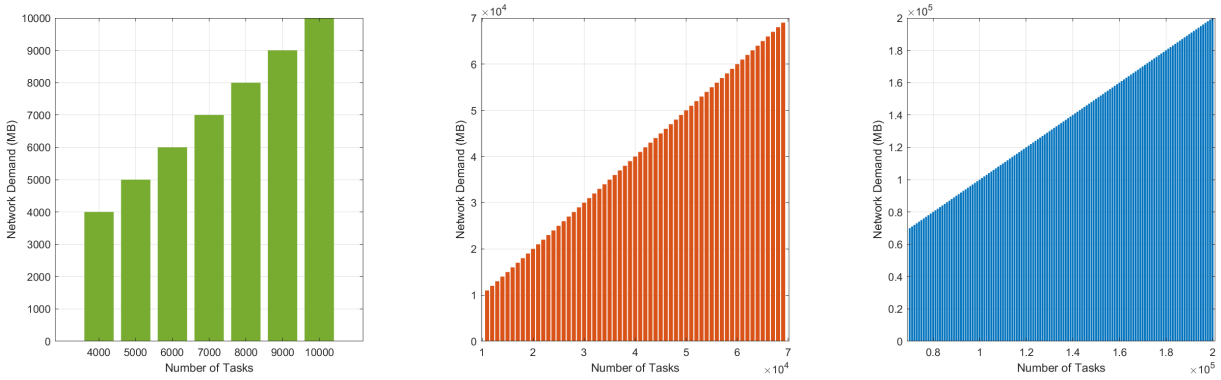
Fig. 7c shows the network demand for Edge, Regional, and Cloud computing models. CC contributes the maximum burden to the network, where the demand is ever-increasing because of the long transmission and centralized data handling, as shown in Fig. 7c. The demand reaches close to 200,000 MB with the increase in the number of tasks. Due to the centralized nature, there is bandwidth saturation and communication delays, making it not feasible for real-time IoT services. On the other hand, RC, as shown in Fig. 7b maintains a balanced and scalable network demand profile. It benefits from regional proximity to devices, reducing excessive data hops and offloading pressure on both the Edge and Cloud. The steady rise in demand confirms that regional servers effectively manage transmission without overutilizing the network. This makes RC an ideal compromise, delivering scalable bandwidth usage without compromising latency. Among them,

EC presents the minimum requirement because of the localization of tasks and very low overhead for communications. However, its scalability is limited, which reduces the processing capacity. It cannot support massive IoT loads. Therefore, for less intensive and more localized tasks, Edge is superior, while RC provides the best cost-effectiveness and scalability trade-off for wider deployments.



(a) Edge (b) Regional (c) Cloud

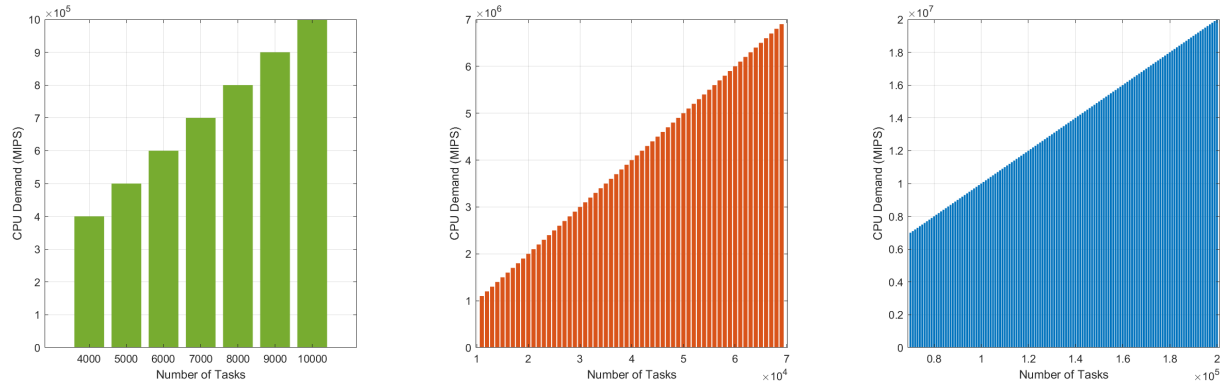
Figure 6: Energy consumption cost across computing paradigms under different workloads.



(a) Edge (b) Regional (c) Cloud

Figure 7: Network demand (in MB) observed at each paradigm tier during task offloading.

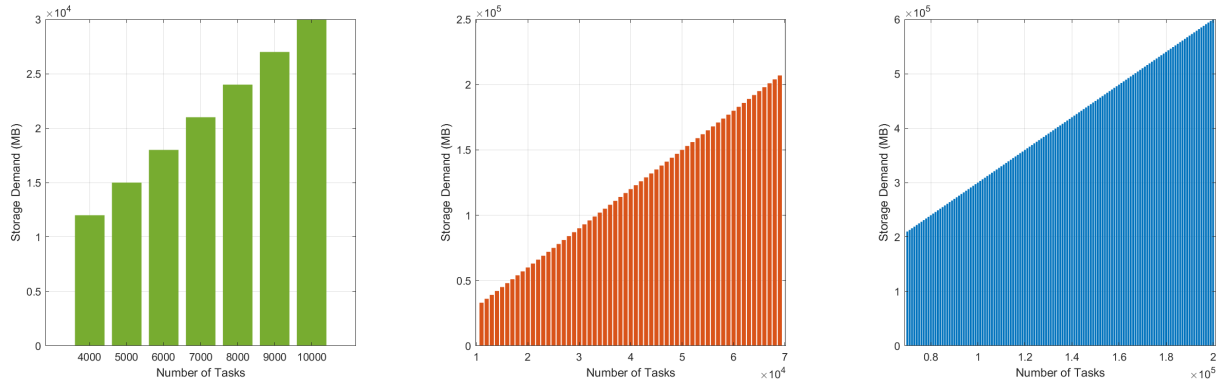
The CPU demand of the Edge, Regional, and Cloud paradigms against their increasing number of offloaded tasks is shown in Fig. 8. As anticipated, all three paradigms demonstrate a linear increase in CPU requirements with increasing workload intensity. The Cloud paradigm has the highest demand on account of processing massive amounts of remote tasks, as illustrated in Fig. 8c. On the other hand, EC has the lowest CPU demand due to its limited task intake and locally processed tasks, as shown in Fig. 8a. Its limited scalability, however, renders it unsuitable for large IoT systems. In Fig. 8b, we observe that regional computing effectively balances handling a substantial workload while maintaining control over CPU patterns. By being distributed closer to data sources, it alleviates the extreme loads typically placed on cloud servers. This not only enhances responsiveness but also improves energy efficiency. As a result, the regional computing paradigm is scalable and resource-efficient, enabling it to support higher workloads without imposing excessive computational pressure.



(a) Edge (b) Regional (c) Cloud

Figure 8: CPU usage (in MIPS) across Edge, Regional, and Cloud paradigms based on increasing workload.

Fig. 9 compares the storage demand observed at the Edge, Regional, and Cloud computing layers as the number of offloaded tasks increases. At the Edge, as shown in Fig. 9a, storage demand remains lowest due to limited computational capacity, making it suitable only for delay-sensitive tasks with minimal computational requirements. The Cloud, while offering massive resources, shows the highest storage demand due to the accumulation of massive data as shown in Fig. 9c. The Regional paradigm offers a balance, capable of handling significantly higher storage volumes than Edge while avoiding the excessive demand seen in Cloud environments, as shown in Fig. 9b. This positions the Regional layer as a scalable yet efficient intermediary, capable of supporting growing data volumes without overwhelming the infrastructure, making it well-suited for real-time applications in resource-constrained vehicular environments.

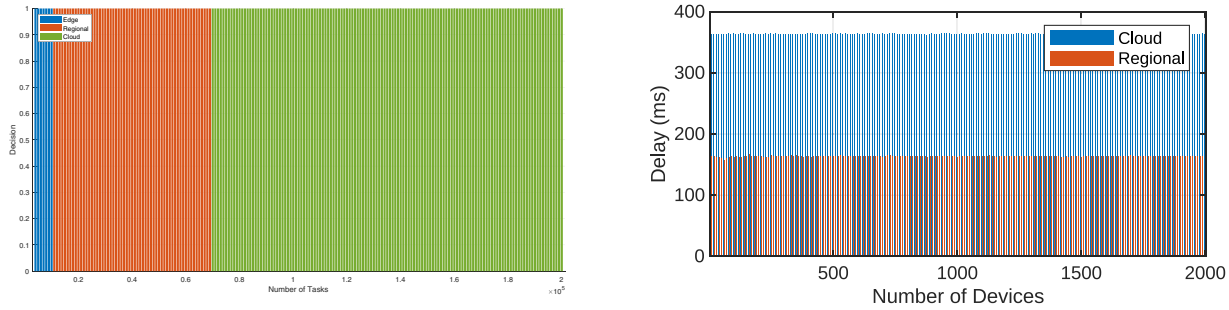


(a) Edge (b) Regional (c) Cloud

Figure 9: Storage demand trends (in MB) across offloading layers under growing task volumes.

The task size distribution across 10 to 2000 IoT devices and its offloading decisions are shown in Fig. 10a. We can see that as the workload increases, tasks are initially processed on edge devices; however, at approximately 110 tasks, the workload is offloaded to RC. This ensures efficient utilization of resources, with RC effectively managing workloads up to 6900 tasks. Tasks exceeding this capacity are further offloaded to CC, where the remaining workload of up to 200,000 tasks is processed. This hierarchical offloading strategy optimizes resource utilization across the computational layers. Fig. 10b shows that network delay varies among computing paradigms. The average network delay of CC is about 363 ms due to the long distance of data transmission to a centralized server. Such a delay will be higher under increased network traffic. On the

other hand, RC provides a much lower and more stable average delay at about 163 ms, with an increased number of devices. This advantage is due to local data processing that reduces the distances of data transfer and relieves network bottlenecks, allowing faster and timelier data processing.



(a) Offloading Decisions (across Edge, Regional and Cloud) with Increasing Task Size

(b) Comparison of Network Delay Across Computing Paradigms

Figure 10: Offloading decisions and network delay across computing paradigms.

5 Discussion

The experimental results highlight the comparative advantages of RC over edge and cloud paradigms in addressing the computational, network, and storage demands of IoT applications. Furthermore, the integration of a FL based decision mechanism plays a critical role in ensuring optimal task offloading to the most suitable computing paradigm. This section discusses the findings in detail, focusing on latency, scalability, congestion, and the decision-making capabilities of FL.

EC shows low-latency processing, as demonstrated by its proximity to IoT devices. However, as Fig. 10a illustrates, it is constrained by its limited resources, which become evident as the number of devices and workload size increase, the FL changes its decision to the RC. These limitations result in resource over-utilization, necessitating offloading to more resourceful paradigms. While EC is suitable for small-scale, latency-sensitive applications, it performs well in scenarios requiring quick response times. However, its inability to scale effectively highlights its limitations in managing the demands of large-scale IoT ecosystems. On the other hand, CC offers virtually unlimited scalability, as shown by its capacity to process workloads exceeding 200,000 tasks. However, as Fig. 10b reveals, the network delay associated with CC is substantially higher, primarily due to the longer data transmission distances to centralized cloud servers. The results in Fig. 7a–c indicate that CC suffers from severe congestion as the number of IoT devices increases, leading to significant performance degradation. These characteristics render CC less favorable for latency-sensitive applications, despite its scalability. RC, augmented by FL, emerges as the optimal paradigm for IoT applications, providing a balance between low latency and resource scalability. As shown in Fig. 10a,b, RC achieves significantly lower network delays compared to CC while offering scalability that far exceeds the limitations of EC. This is further supported by its moderate network demand in Fig. 7b, positioned between the extremes of the edge (Fig. 7a) and cloud (Fig. 7c) layers. The deployment of localized regional servers shortens transmission distances and mitigates congestion, allowing RC to effectively address edge and cloud paradigms' latency and scalability challenges.

The integration of FL further enhances the decision-making process for task offloading. The FLS dynamically evaluates multiple parameters, including task size, computational requirements, network bandwidth, and transmission costs, to determine the optimal computing paradigm for each task. By utilizing fuzzy inference rules, the system adaptively balances workload distribution across edge, regional,

and cloud resources, as clear by the transition patterns observed in Fig. 10a. This intelligent decision-making capability not only reduces latency but also ensures efficient utilization of computational and network resources. This efficiency is evident in the cost metrics. Fig. 4a–c shows the processing cost trends, highlighting that RC maintains reasonable processing costs while avoiding the steep rise associated with the cloud. Similarly, Fig. 5a–c indicates that RC incurs lower network transmission costs than the cloud due to reduced distance and fewer routing hops. The energy consumption patterns shown in Fig. 6a–c reinforce this efficiency, with RC positioned as a middle ground between low-energy edge and high-energy cloud offloading. Resource demand comparisons further support this conclusion. Fig. 8a–c presents CPU utilization trends, while Fig. 9a–c illustrates storage consumption. In both categories, RC consistently demonstrates scalable capacity without overwhelming local infrastructure or introducing the cost and delay penalties of centralized systems.

The results demonstrate that RC, with a FL based task offloading mechanism, effectively bridges the gap between the low-latency advantage of edge computing and the scalability of CC. The ability of FL to adaptively allocate tasks ensures optimal resource utilization and cost-efficiency, making RC a superior choice for IoT applications with dynamic and diverse requirements.

6 Conclusion and Future Work

The growing number of IoT devices and IoT big data pose great challenges to computer networks and data centers. Although CC has scalability, it usually faces high latency. On the other hand, EC ensures minimum latency. However, it has a lack of scalability. Therefore, RC provides a medium solution. However, the key challenge involves workload allocation with respect to edge, regional, and cloud computing models. This work has proposed a decision support system using FL for overcoming this issue by dynamically migrating IoT big data to edge, regional, and cloud servers. This work uses task size, cost, CPU, and storage requirement as parameters in order to make optimal use of resources, and minimize latency as well as costs. The fuzzy inference subsystem integrates a rule-based subsystem to adaptively assign tasks as per different requirements. Evaluation outcome showed the efficiency of the system by effectively delegating tasks, promoting a good distribution of tasks among the three computing models. This model not only enhanced efficient data transmission in a network but also ensured that it has improved scalability and efficiency. All evaluation tasks were performed using our simulation tool, RegionalEdgeSimPy, which effectively simulated live data from Arduino IoT sensors. However, practical deployment may face challenges related to large-scale infrastructure provisioning, fault tolerance, and operation under dynamic real-world IoT conditions. We aim to enhance our RegionalEdgeSimPy simulator with further functionalities, including energy modeling, improved security analysis, and innovative data prioritization. Moreover, the inclusion of Machine Learning (ML) and Deep Learning (DL) techniques in our decision engine will also contribute to further improvement in our dynamic workload offloading mechanism.

Acknowledgement: Not applicable.

Funding Statement: The authors received no external funding for this research.

Author Contributions: Afzal Badshah contributed to conceptualization, methodology, software development, and writing the original draft. Mona Eisa contributed to validation, formal analysis, and writing through review and editing. Omar Alghushairy contributed to formal analysis, investigation, and manuscript review and editing. Riad Alharbey contributed to investigation, validation, and manuscript review and editing. Manal Linjawi contributed to data curation, visualization, and manuscript review and editing. Ali Daud contributed through supervision, project administration, and manuscript review and editing. All authors reviewed and approved the final version of the manuscript.

Availability of Data and Materials: The simulation code (RegionalEdgeSimPy), fuzzy controller implementation, and Arduino data generator scripts are available on GitHub [44]. The datasets generated during simulation are available from the corresponding authors upon reasonable request.

Ethics Approval: This study did not involve human participants or animals and therefore did not require ethical approval. Informed consent is not applicable.

Conflicts of Interest: The authors declare no conflicts of interest.

Appendix A Fuzzy Logic System Design Details

Appendix A.1 Fuzzy Variables and Membership Functions

This fuzzy logic controller was designed taking the data from sensors that were produced in real-time using Arduino UNO R4 WiFi modules linked to environmental sensors like temperature, humidity, and air quality. Every reading from such a sensor was then converted into a task that has quantifiable properties of size and computational demands.

The range of every fuzzy variable was determined by experiment, picking out minimum and maximum values measured in actual test scenarios. To model IoT big data traffic, these actual values were then proportionally scaled. A graphical representation of scaling was done for understanding. For instance, whereas in actual tests, the minimum measured size of a task was about 1000 KB, in simulation, it was increased to 2,000,000 KB. The controller takes five input variables and one output variable. The input and output variables are modeled with a triangular membership function as follows:

$$devices = \begin{cases} \text{few}, & [10, 10, 500] \\ \text{moderate}, & [250, 1000, 1500] \\ \text{many}, & [1000, 2000, 2000] \end{cases} \tag{A1}$$

$$task\_size = \begin{cases} \text{small}, & [1000, 1000, 500000] \\ \text{medium}, & [300000, 1000000, 1500000] \\ \text{large}, & [1000000, 2000000, 2000000] \end{cases} \tag{A2}$$

$$cost = \begin{cases} \text{low}, & [1.4, 1.4, 335.818] \\ \text{medium}, & [175.933, 500, 768.675] \\ \text{high}, & [587.993, 1028, 1028] \end{cases} \tag{A3}$$

$$cpu\_demand = \begin{cases} \text{low}, & [100000, 100000, 700000] \\ \text{medium}, & [500000, 1000000, 1500000] \\ \text{high}, & [1300000, 2000000, 2000000] \end{cases} \tag{A4}$$

$$storage\_demand = \begin{cases} \text{low}, & [3000, 3000, 30000] \\ \text{medium}, & [20000, 100000, 300000] \\ \text{high}, & [200000, 600000, 600000] \end{cases} \tag{A5}$$

The output variable *scenario* is defined as:

$$scenario = \begin{cases} \text{edge}, & [1.188, 1.188, 1.191] \\ \text{regional}, & [1.190, 1.193, 1.196] \\ \text{cloud}, & [1.195, 1.198, 1.200] \end{cases} \tag{A6}$$

Appendix A.2 Fuzzy Rule Base

The fuzzy inference system uses a set of conditional rules linking input combinations to the output decision. Selected representative rules include:

$$\text{If } \{task\_size = small,\ cost = low,\ cpu\_demand = low, \tag{A7}$$

$$storage\_demand = low\} \Rightarrow scenario = edge \tag{A8}$$

$$\text{If } \{task\_size = small,\ cost = medium,\ cpu\_demand = low, \tag{A9}$$

$$storage\_demand = medium\} \Rightarrow scenario = edge \tag{A10}$$

$$\text{If } \{task\_size = small,\ cost = high,\ cpu\_demand = medium, \tag{A11}$$

$$storage\_demand = medium\} \Rightarrow scenario = regional \tag{A12}$$

$$\text{If } \{task\_size = medium,\ cost = low,\ cpu\_demand = medium, \tag{A13}$$

$$storage\_demand = medium\} \Rightarrow scenario = regional \tag{A14}$$

$$\text{If } \{task\_size = medium,\ cost = medium,\ cpu\_demand = medium, \tag{A15}$$

$$storage\_demand = high\} \Rightarrow scenario = regional \tag{A16}$$

$$\text{If } \{task\_size = medium,\ cost = high,\ cpu\_demand = high, \tag{A17}$$

$$storage\_demand = high\} \Rightarrow scenario = cloud \tag{A18}$$

$$\text{If } \{task\_size = large,\ cost = low,\ cpu\_demand = medium, \tag{A19}$$

$$storage\_demand = medium\} \Rightarrow scenario = regional \tag{A20}$$

$$\text{If } \{task\_size = large,\ cost = medium,\ cpu\_demand = high, \tag{A21}$$

$$storage\_demand = high\} \Rightarrow scenario = regional \tag{A22}$$

$$\text{If } \{task\_size = large,\ cost = high,\ cpu\_demand = high, \tag{A23}$$

$$storage\_demand = high\} \Rightarrow scenario = cloud \tag{A24}$$

Appendix A.3 Decision Thresholds

The defuzzified *scenario* output is mapped to the final offloading layer as:

$$scenario = \begin{cases} \text{Edge}, & scenario < 1.1920, \\ \text{Regional}, & 1.1920 \leq scenario < 1.1929, \\ \text{Cloud}, & scenario \geq 1.1929 \end{cases} \tag{A25}$$

Appendix A.4 Notes on Design and Availability

The membership functions and rule base were derived and tuned using simulated IoT workloads. The fuzzy controller and Arduino data generator code are publicly available on GitHub [44].

References

1. Statista. Internet of Things (IoT) revenue worldwide. 2024 [cited 2024 Dec 26]. Available from: https://www.statista.com/statistics/1194709/iot-revenue-worldwide/.
2. Badshah A, Daud A, Khan HU, Alghushairy O, Bukhari A. Optimizing the over and underutilization of network resources during peak and off-peak hours. IEEE Access. 2024;12(2):82549–59. doi:10.1109/access.2024.3402396.
3. Zhou Z, Abawajy J, Chowdhury M, Hu Z, Li K, Cheng H, et al. Minimizing SLA violation and power consumption in cloud data centers using adaptive energy-aware algorithms. Future Gen Comput Syst. 2018;86(6):836–50. doi:10.1016/j.future.2017.07.048.
4. Sun G, Wang Z, Su H, Yu H, Lei B, Guizani M. Profit maximization of independent task offloading in MEC-enabled 5G internet of vehicles. IEEE Trans Intell Transp Syst. 2024;25(11):16449–61. doi:10.1109/tits.2024.3416300.
5. Jiang H, Cai J, Xiao Z, Yang K, Chen H, Liu J. Vehicle-assisted service caching for task offloading in vehicular edge computing. IEEE Trans Mob Comput. 2025;24(7):6688–700. doi:10.1109/tmc.2025.3545444.
6. Wu G, Chen X, Gao Z, Zhang H, Yu S, Shen S. Privacy-preserving offloading scheme in multi-access mobile edge computing based on MADRL. J Parallel Distr Comput. 2024;183:104775. doi:10.1016/j.jpdc.2023.104775.
7. Badshah A, Daud A, Alhajlah M, Alsahfi T, Alshemaimri B, Ur-Rehman G. Smart cities' big data: performance and cost optimization with regional computing. IEEE Access. 2024;12(70):128896–908. doi:10.1109/access.2024.3457269.
8. Badshah A, Rehman GU, Farman H, Ghani A, Sultan S, Zubair M, et al. Transforming educational institutions: harnessing the power of internet of things, cloud, and fog computing. Future Internet. 2023;15(11):367. doi:10.3390/fi15110367.
9. Song L, Sun G, Yu H, Niyato D. ESPD-LP: edge service pre-deployment based on location prediction in MEC. IEEE Trans Mob Comput. 2025;24(6):5551–68.
10. Khan S, Jiangbin Z, Ali H. Soft computing approaches for dynamic multi-objective evaluation of computational offloading: a literature review. Clust Comput. 2024;27(9):12459–81. doi:10.1007/s10586-024-04543-y.
11. Tang HH, Ahmad NS. Fuzzy logic approach for controlling uncertain and nonlinear systems: a comprehensive review of applications and advances. Syst Sci Control Eng. 2024;12(1):2394429. doi:10.1080/21642583.2024.2394429.
12. Deng X, Yin J, Guan P, Xiong NN, Zhang L, Mumtaz S. Intelligent delay-aware partial computing task offloading for multiuser industrial Internet of Things through edge computing. IEEE Internet Things J. 2021;10(4):2954–66. doi:10.1109/jiot.2021.3123406.
13. Alashjaee AM, Irshad A, Daud A, Alhomoud A, Altowaijri SM, Alshdadi AA. Resots: rfid/iot-enabled secure object tracking key exchange for trustworthy smart logistics. Wirel Pers Commun. 2024;139(2):777–99. doi:10.1007/s11277-024-11598-y.
14. Singh N, Das AK. Energy-efficient fuzzy data offloading for IoMT. Comput Netw. 2022;213(4):109127. doi:10.1016/j.comnet.2022.109127.
15. Sasubilli SM, Kumar A, Dutt V. Improving health care by help of internet of things and bigdata analytics and cloud computing. In: 2020 International Conference on Advances in Computing and Communication Engineering (ICACCE). Piscataway, NJ, USA: IEEE; 2020. p. 1–4.
16. Akherfi K, Gerndt M, Harroud H. Mobile cloud computing for computation offloading: issues and challenges. Appl Comput Inform. 2018;14(1):1–16. doi:10.1016/j.aci.2016.11.002.
17. Abirami S, Chitra P. Energy-efficient edge based real-time healthcare support system. In: Advances in computers. Amsterdam, The Netherlands: Elsevier; 2020. p. 339–68. doi: 10.1016/bs.adcom.2019.09.007.
18. Awad AI, Fouda MM, Khashaba MM, Mohamed ER, Hosny KM. Utilization of mobile edge computing on the Internet of Medical Things: a survey. ICT Express. 2023;9(3):473–85. doi:10.1016/j.icte.2022.05.006.
19. Raj JS, Jennifer S. Optimized mobile edge computing framework for IoT based medical sensor network nodes. J Ubiquitous Comput Commun Technol. 2021;3(1):33–42.
20. Kumar D, Maurya AK, Baranwal G. IoT services in healthcare industry with fog/edge and cloud computing. In: IoT-based data analytics for the healthcare industry. Amsterdam, The Netherlands: Elsevier; 2021. p. 81–103. doi: 10.1016/b978-0-12-821472-5.00017-x.

21. Rahmani AM, Gia TN, Negash B, Anzanpour A, Azimi I, Jiang M, et al. Exploiting smart e-Health gateways at the edge of healthcare Internet-of-Things: a fog computing approach. Future Gen Comput Syst. 2018;78(7):641–58. doi:10.1016/j.future.2017.02.014.
22. Lo'ai AT, Mehmood R, Benkhlifa E, Song H. Mobile cloud computing model and big data analysis for healthcare applications. IEEE Access. 2016;4:6171–80. doi:10.1109/access.2016.2613278.
23. Soni D, Kumar N. Fuzzy logic-based computation offloading technique in fog computing. Concurr Comput. 2024;36(20):e8198. doi:10.1002/cpe.8198.
24. Vemireddy S, Rout RR. Fuzzy reinforcement learning for energy efficient task offloading in vehicular fog computing. Comput Netw. 2021;199(6):108463. doi:10.1016/j.comnet.2021.108463.
25. Trabelsi Z, Ali M, Qayyum T. Fuzzy-based task offloading in Internet of Vehicles (IoV) edge computing for latency-sensitive applications. Internet Things. 2024;28(3):101392. doi:10.1016/j.iot.2024.101392.
26. Nguyen V, Khanh TT, Nguyen TD, Hong CS, Huh EN. Flexible computation offloading in a fuzzy-based mobile edge orchestrator for IoT applications. J Cloud Comput. 2020;9(1):1–18. doi:10.1186/s13677-020-00211-9.
27. Chakraborty C, Mishra K, Majhi SK, Bhuyan HK. Intelligent latency-aware tasks prioritization and offloading strategy in distributed fog-cloud of things. IEEE Trans Ind Inform. 2023;19(2):2099–106. doi:10.1109/tii.2022.3173899.
28. Cheng X, Chen Y, Yang X, Zhou H, Luo L, Guo D. Memory-efficient programmable packet parsing for multi-tenant terabit networks. Comput Netw. 2025;264(3):111240. doi:10.2139/ssrn.4865842.
29. Queiroz J, Leitão P, Oliveira E. A fuzzy logic recommendation system to support the design of cloud-edge data analysis in cyber-physical systems. IEEE Open J Ind Electron Soc. 2022;3(2):174–87. doi:10.1109/ojies.2022.3152725.
30. Wang S, Qin T, Chen T, Guo W, Hu Y, Sun H. A high-reliability small-area task offloading mechanism with trust evaluation and fuzzy logic in power IoTs. IEEE Trans Mob Comput. 2025;24(4):2935–48. doi:10.1109/tmc.2024.3502167.
31. Zenasni N, Habib C, Nassar J. A fuzzy logic based offloading system for distributed deep learning in wireless sensor networks. In: 2022 International Joint Conference on Neural Networks (IJCNN). Piscataway, NJ, USA: IEEE; 2022. p. 1–8.
32. Behera SR, Panigrahi N, Bhoi SK, Bilal M, Sahoo KS, Kwak D. A distributed fuzzy optimal decision making strategy for task offloading in edge computing environment. IEEE Access. 2023;11(4):33189–204. doi:10.1109/access.2023.3262611.
33. Feng S, Li N, Liu K, Li B, Dong C, Wu Q. A cross q-learning assisted resource allocation for user-centric optical wireless communication networks. IEEE Trans Green Commun Netw. 2025;9(4):2264–78. doi:10.1109/tgcn.2025.3553202.
34. Wang J, Zhao L, Liu J, Kato N. Smart resource allocation for mobile edge computing: a deep reinforcement learning approach. IEEE Trans Emerg Top Comput. 2021;9(3):1529–41. doi:10.1109/tetc.2019.2902661.
35. Li Y, Wu B. Software-defined heterogeneous edge computing network resource scheduling based on reinforcement learning. Appl Sci. 2023;13(1):426.
36. Amini P, Kalbasi A. An adaptive task scheduling approach for cloud computing using deep reinforcement learning. In: Proceedings of the 2024 3rd International Conference on Distributed Computing and High Performance Computing, DCHPC 2024; 2024 May 14–15; Tehran, Iran. p. 1–9.
37. Chen P, Luo L, Guo D, Wu J, Chi K, Yan C, et al. QoS-oriented task offloading in NOMA-based multi-UAV cooperative MEC systems. IEEE Trans Wirel Commun. 2026;25(1):1965–80. doi:10.1109/twc.2025.3593884.
38. Ding A, Hass A, Chan M, Sehatbakhsh N, Zonouz S. Resource-aware DNN partitioning for privacy-sensitive edge-cloud systems. In: Neural information processing (ICONIP 2023). Singapore: Springer; 2024. p. 188–201. doi:10.1007/978-981-99-8073-4_15.
39. Ho TM, Nguyen KK, Cheriet M. Federated deep reinforcement learning for task scheduling in heterogeneous autonomous robotic system. IEEE Trans Autom Sci Eng. 2024;21(1):528–40. doi:10.1109/tase.2022.3221352.
40. Liu W, Li B, Xie W, Dai Y, Fei Z. Energy efficient computation offloading in aerial edge networks with multi-agent cooperation. IEEE Trans Wirel Commun. 2023;22(9):5725–39. doi:10.1109/twc.2023.3235997.

41. Li P, Xiao Z, Wang X, Huang K, Huang Y, Gao H. EPtask: deep reinforcement learning based energy-efficient and priority-aware task scheduling for dynamic vehicular edge computing. IEEE Trans Intell Veh. 2024;9(1):1830–46.
42. Binh TH, Son DB, Vo H, Nguyen BM, Binh HTT. Reinforcement learning for optimizing delay-sensitive task offloading in vehicular edge-cloud computing. IEEE Internet Things J. 2024;11(2):2058–69. doi:10.1109/jiot.2023.3292591.
43. Alshammari H, El-Ghany SA, Shehab A. Big IoT healthcare data analytics framework based on fog and cloud computing. J Inform Process Syst. 2020;16(6):1238–49. doi:10.1002/9781394175345.ch12.
44. Badshah A. FuzzyLogic-for-smart-offloading-of-IoT-Big-Data. 2025 [cited 2025 Jun 28]. Available from: https://github.com/afzalbadshah/FuzzyLogic-for-Smart-Offlaoding-of-IoT-Big-Data.