Enhancing Disaster Management with Efficient User Allocation Using Quantum-Inspired Cuckoo Search and UAV-Supported Edge Computing

Thandava Purandeswar Reddy*, Gokarakonda Nikhil Sri Sai Teja*, Bhukya Dayanand*, A.Swamy Goud*, Banavath Balaji Naik*, Gopa Bhaumik**, Bhabani Sankar Das***

Computer Science and Engineering
*National Institute Technology Patna, Bihar, India-800005**

Computer Science and Engineering
*National Institute Technology Jamshedpur, Jharkhand, India-831014***

Civil Engineering
*National Institute Technology Patna, Bihar, India-800005****

Abstract

Efficient allocation of IoT devices and timely real-time data processing in disaster-affected areas pose significant challenges due to energy constraints, limited infrastructure, and the urgent need for rapid response. Unmanned Aerial Vehicles (UAVs), when integrated with Edge Computing EC, offer a promising solution for collecting and processing data. In this paper, we propose a novel Quantum-Inspired Cuckoo Search Optimization (QICSO) algorithm to address key challenges in disaster management, including user allocation, processor allocation, energy consumption, and time delay factors. The QICSO algorithm is designed to optimize resource utilization by addressing key objectives such as minimizing energy consumption, reducing data processing delays, and improving the allocation rates of users and processors (edge servers and UAVs). A fitness function is formulated by considering these factors to ensure efficient disaster response. The algorithm is evaluated against leading evolutionary algorithms. This paper presents a Disaster Management user allocation framework for Unmanned Aerial Vehicles (UAVs) and Edge Computing using Quantum-Inspired Cuckoo Search Optimization (QICSO). A novel representation of the quantum cuckoo and qubit has been developed, where quantum cuckoos are encoded and decoded using an innovative hashing technique. The fitness function incorporates four objectives: user allocation rate, processor allocation rate, energy consumption, and delay. The proposed approach is simulated across multiple scenarios and benchmarked against the latest existing algorithms. Results demonstrate that the proposed algorithm significantly outperforms the compared methods. Additionally, statistical analyses, including analysis of variance (ANOVA) and

Email addresses: thandavar.ug21.cs@nitp.ac.in (Thandava Purandeswar Reddy*), gokarakondat.ug21.cs@nitp.ac.in (Gokarakonda Nikhil Sri Sai Teja*), bhukyad.ug21.cs@nitp.ac.in (Bhukya Dayanand*), a.swamyg.ph23.cs@nitp.ac.in (A.Swamy Goud*), balaji.cs@nitp.ac.in (Banavath Balaji Naik*), gopabhaumik.cse@nitjsr.ac.in (Gopa Bhaumik**), bsd.ce@nitp.ac.in (Bhabani Sankar Das***)

the Friedman test, have been conducted. The Taguchi parametric statistical technique is employed to further evaluate performance. The simulation results indicate that QICSO surpassed existing approaches in terms of energy consumption by 18.3% and delay by 3.5%.

Keywords:
Internet of Things, Unmanned Aerial Vehicles, Mobile edge computing, Quantum Computing, Cuckoo Search Optimization, Edge servers, Hashing Technology

1. Introduction

Natural disasters around the world pose serious threats to environmental stability and animal welfare. These events can severely disrupt various societal operations, including transportation routes, economic systems, and communication networks. After a catastrophic event like a flood, storm, or earthquake, there is an immediate need for accurate and timely data collection and analysis to support an effective response plan. However, the extensive damage often restricts access to affected communities, complicating efforts by social service and humanitarian groups to assess and address urgent needs which are addressed in Ejaz et al. (2020). Additionally, the destruction of infrastructure can hinder unmanned ground vehicles (UGVs) from providing aid. Communication efforts are further challenged when mobile networks malfunction, especially in densely populated urban areas that are vulnerable to secondary disasters such as fires. Regrettably, it is not feasible to promptly pinpoint the exact location of the fire and initiate damage control operations due to the limited accessibility and lack of options for sharing information Maghsoudi et al. (2023). A city named Istanbul. Where the fire detection system has been developed using the concept of machine learning by Convolutional Neural Networks (CNNs) to point out the after-that disaster scenarios. These fire detection methods require computational power Which can create an impact on Internet of Things (IoT) devices Shahmoradi et al. (2020) Here the UAVs present a unique solution with their supervision of mobility and superfast computational strengths.

Drones are crucial in this situation primarily because they streamline the system and take on the labor-intensive task of fire detection from IoT devices. By knowingly placing IoT devices in areas susceptible to disasters Data collection will be stronger Zhang et al. (2019). IoT devices are essential for gathering data and sending critical information to surrounding stations, facilitating rapid damage assessments, and organizing rescue operations Yu et al. (2024). They are utilized during UAV installation. Additionally, UAVs can improve connection in disaster-affected areas by offering network services Yu et al. (2024). Therefore, using UAVs with powerful computational capabilities offers a practical and life-saving way to improve relationships and Effects in Post-Disaster Environments. Frequent execution and computation of resource-intense activities and long communication distances deplete the energy resources of IoT devicesFarahbakhsh et al. (2021). Mobile Edge Computing provides a solution for this problem by offloading the resource-intensive tasks of IoT devices to Edge Servers and UAV Li et al. (2020). This solution helps optimize task processing efficiency. However, there is a small disadvantage associated with this Ye et al. (2020). Offloading tasks from IoT devices to Edge servers and UAVs will add latency and network load, which may disturb task processing Hu et al. (2019). So, it is essential to balance the acceptable delay and energy efficiency Hu et al. (2019). Therefore, there are two major problems to solve

before the UAV-assisted EC can be comfortably integrated with disaster managementYu et al. (2024). The first one is to effectively allocate tasks to processing units (PUs) and manage computational workloads and resource optimization. The second one is to reduce overall delay or latency and energy consumption across UAVs, ESs, and IoTs, load balancing of tasks between UAVs and ESs Wang et al. (2019a). This research focuses on enhancing disaster response by optimizing task offloading, aiming to speed up response times, reducing the overall impact of disasters, and most importantly, helping save lives.

A multi-UAV-assisted MEC network is non-deterministic polynomial complete (NP-complete) Wang et al. (2019b). It is a major flooding issue, and it conveys that no algorithm can provide an ideal solution in polynomial time. Moreover, due to their vast computational demands, exponential algorithms are not the best way to find the optimal solutions. Researchers have turned to evolutionary algorithms (EAs) such as Cuckoo Search Optimization (CSO Bakshi et al. (2023)), Differential Evolution (DE Bandyopadhyay et al. (2024)), and Particle Swarm Optimization (PSO Maratha et al. (2021)) because they offer practical solutions to complex NP-complete problems within polynomial time.

It is important to keep in mind that the issue of Efficient Resource-Based User Allocation (ERUA) is NP-complete Ghosh and Kuila (2023); Balaji Naik et al. (2020). A family of optimization techniques known as Quantum-Inspired Evolutionary Algorithms (QIEAs) makes use of ideas from quantum physics to improve the effectiveness of traditional evolutionary algorithms (EAs). EAs may now search larger solution areas and resolve faster toward ideal outcomes for these quantum-inspired approachesBey et al. (2024) To tackle difficult optimization issues, novel Quantum-Inspired Cuckoo Search Optimization (QICSO) was developed. In this paper, we handle the ERUA in the edge computing environment using an algorithm based on QICSO.

In this research, we introduce Quantum Inspired Cuckoo Search Optimization (QICSO), a novel user allocation method for the Edge Computing (EC) environment. The following are this paper's primary contributions:

- A mathematical proof of the NP-completeness of the Efficient Resource-Based User Allocation (ERUA) problem.
- A new approach to solving the ERUA problem is presented, using encoded Quantum Vectors (QVs) in conjunction with a special hashing technique to enable further decoding.
- The fitness function is designed by using four objectives user allocation rate, processor allocation rate, delay, and energy consumption.
- It is shown that the proposed algorithm Efficient Resource-based user allocation by using Quantum Inspired Cuckoo Search Optimization is computed in the polynomial time.
- The Design of Experiments (DoE) method is implemented using the Taguchi approach. Extensive simulations demonstrate that the ERUA-QICSO algorithm outperforms existing algorithms, including CSOBakshi et al. (2023), PSOMaratha et al. (2021), and DEBandyopadhyay et al. (2024). Furthermore, hypothesis-based statistical analyses,

such as ANOVA and the Friedman test, are performed to validate the significance of the proposed approach.

2. Related Work

Edge Computing (EC) is a distributed computing approach that extends beyond traditional cloud computing, helping to reduce latency. To minimize computation energy and delay across the system, authors of the study Ren et al. (2021) introduced two approaches for Mobile Edge Computing (MEC) networks with using of UAVs (Unmanned Aerial Vehicles). These approaches focus upon optimizing transmission and offloading resources ratios by applying the Karush-Kuhn-Tucker conditions. And additionally, game theory is used to manage the automated behavior of UAVs, to make sure that they select the most efficient offloading methods. These results demonstrate the effectiveness of these approaches, especially in scenarios involving multiple of UAVs. In another study made by Zhang et al. (2023), a deep reinforcement learning-based approach for computational offloading (DRCOM) in MEC networks is proposed. This approach points out the two primary challenges which were efficient resource allocation across the system and determining the optimal offloading decisions for each user with the services of the UAV.

The authors of Yu et al. (2024) proposed a multi-agent deep reinforcement learning algorithm based on QMIX to state the challenges in Aerial Edge Computing (AEC) networks, where UAVs are used as mobile edge servers. with inclusion of theoretical analysis, the approach complexity reduces efficiently. The primary objective is to enable offloading across system while minimizing the distance between users and UAVs. Notably, the simulations demonstrate significant performance improvements through simultaneous in optimization of trajectory planning and resources offloading.

References	**Metrics Considered**			
	Energy Consumption	**Offloading Delay**	**User Allocation Rate**	**Edge Servers Required**
MECA Ren et al. (2021)	✓	✓	✓	✓
DRCOM Zhang et al. (2023)	✓	✗	✗	✓
QMIX Yu et al. (2024)	✓	✗	✓	✗
EUAGame He et al. (2020)	✗	✗	✓	✗
CO Sadatdiynov et al. (2023)	✓	✓	✓	✗
LOM Dai et al. (2024)	✓	✓	✓	✗
JORA Jiang et al. (2023)	✓	✗	✓	✓
EUA Wu et al. (2021)	✓	✗	✓	✓
COP Wang et al. (2019a)	✗	✓	✓	✓
ERUA-QICSO	✓	✓	✓	✓

Table 1: Overview of Related Works

The authors of He et al. (2020) presented a game-theoretic method called EUAGame, which frames the EUA issue as a game. They demonstrated through analysis that the game developed a unique distributed method to solve the EUA problem by locating a Nash equilibrium. The performance of this algorithm is assessed both problematically and theortically. Edge computing lowers compute latency and energy consumption by offloading tasks to edge servers. However, offloading is most effective for certain tasks, necessitating optimization methods. The authors of Sadatdiynov et al. (2023) reviewed six methods: convex

optimization, heuristic techniques, game theory, machine learning, and Lyapunov optimization. This research aims to assist new researchers in optimizing compute offloading in Mobile Edge Computing networks.

In Dai et al. (2024), the authors decoupled the long-term energy limitation using the Lyapunov optimization technique. This allows for real-time solutions without the requirement of future knowledge. Then they constructed a Markov chain using Markov approximation optimization to determine the near optimal UAV-assisted offloading strategies. The authors of Jiang et al. (2023) presented an online joint offloading and resource allocation (JORA) system under the long-term MEC energy constraint. They applied Lyapunov optimization to the long-term QoE maximization problem, leveraging its optimality. They built an energy deficient queue to direct energy use, so offering both distributed and centralized online JORA solutions to solve the problem in real time.

In Wu et al. (2021), the authors introduced a decentralized reactive method that makes real-time allocation decisions using a fuzzy control mechanism. Meanwhile, the authors of Wang et al. (2019a) described the problem as a multi-objective constraint optimization problem, strategically placing edge servers to balance workloads and minimize access delays between users and edge servers. They used mixed integer programming to determine the best course of action. The summary of considered parameters in each literature and our approach is given in Table: 1.

2.1. Author Contributions Compared to Existing Approaches

The following lists the unique features of the suggested work in comparison to the previous research:

- As per our best knowledge, this is the first research work on enhancing disaster management using the assistance of UAVs for edge computing systems considering all the parameters mentioned by authors in Ren et al. (2021); Zhang et al. (2023); Yu et al. (2024); He et al. (2020); Sadatdiynov et al. (2023); Dai et al. (2024); Jiang et al. (2023); Wu et al. (2021); Wang et al. (2019a) all at a time.

- Our approach incorporates quantum computing to enable super-fast computations, which are currently limited by existing technology. This forward-looking integration ensures that our methodology can leverage quantum computing for faster real-time calculations once the technology becomes widely available.

- We have taken essential and relevant parameters and techniques needed to calculate critical factors for evaluation of energy and delay consumption in the network. UAV and ES's network & physical capabilites were taken into evaluating in allocation of users to processing units in a disaster situation. Mathematical models were developed based on the above factors. Our proposed model has considered all the mentioned parameters mentioned by authors in Ren et al. (2021); Zhang et al. (2023); Yu et al. (2024); He et al. (2020); Sadatdiynov et al. (2023); Dai et al. (2024); Jiang et al. (2023); Wu et al. (2021); Wang et al. (2019a).

3. System Model

Imagine a disaster zone equipped with internet-connected sensors (IoT nodes) scattered throughout the affected area. These devices continuously monitor the disaster's effects. Additionally, edge servers (ESs) are strategically placed nearby to collect data from the IoT nodes. To expedite the data collection process and relay it to the edge servers, Unmanned Aerial Vehicles (UAVs) are also deployed in the area. This disaster management system is presumed to have three layers: The first layer is the IoT layer, which consists of n number of IoTs, $S = \{d_1, d_2, \ldots d_i, \ldots d_n\}$, $1 \leq i \leq n$ generates n number of tasks $T = \{\tau_1, \tau_2, \ldots \tau_i, \ldots \tau_n\}$, $1 \leq i \leq n$. It is assumed that each IoT device generate one independent task. The second layer is the UAV layer, which consists of q number of UAVs: U $= \{u_1, u_2, \ldots, u_j, \ldots, u_q\}$, $1 \leq j \leq q$. The final layer consists of edge servers with r number of ESs: E $= \{e_1, e_2, \ldots, e_k, \ldots, e_r\}$, $1 \leq k \leq r$. Furthermore, the system has a total of $q + r = s$ processing units, which are a combination of UAVs and edge servers, represented as P $= \{p_1, p_2, .., p_m, ..., p_s\}$, where $1 \leq m \leq s$. Each processing unit p_m has a predefined amount of resources for computing which is denoted as vector $< p^m_{CPU}, p^m_{RAM}, p^m_{BAN}, p^m_{STO} >$, and it is located at geographical coordinate p^m_{COR}. Here p^m_{CPU} is the CPU capacity, p^m_{RAM} is the main memory, p^m_{BAN} is the bandwidth, p^m_{STO} is the storage capacity, $p^m_{COR} = (x_m, y_m)$ and is the geographical coordinate of the p_m. The x_m and y_m are the latitude and longitude respectively.

As previously mentioned, Let's consider that the disaster zone equipped with Internet of Things (IoTs) devices, such as mobile devices, wireless transceivers, etc, and flying unmanned aerial vehicles (UAVs) that move around the affected area to gather data, while edge servers (ESs) stay stationary in their designated locations (as shown in Figure.1). An intelligent agent (IA) acts as the brain of the operation, orchestrating communication between the IoTs, UAVs, and ESs. The IA determines the most efficient way to deliver tasks from the IoTs, either directly to the ESs or by relaying them through the UAVs. We assume that some of the IoTs are within the communicable range of some of the processing units (PUs), which can be either UAVs or ESs. The UAVs maintain a constant altitude. Additionally, each IoT has only one connection request at a time to a processing unit. Refer to Table 2 for a list of notations used throughout this manuscript.

3.1. Processing Units (PU) Status Model

PUs deployed in a specific area can service a varying number of IoTs. If a PU is servicing at least one IoT, it is considered active. This is represented in the following Eq. (1).

$$\phi_m = \begin{cases} 1; \text{If } p_m \text{ providing service for atleast one IoT} \\ 0; \text{Otherwise} \end{cases} \tag{1}$$

Consequently, the following method can be used to determine the total number of PUs currently in use in a particular area.

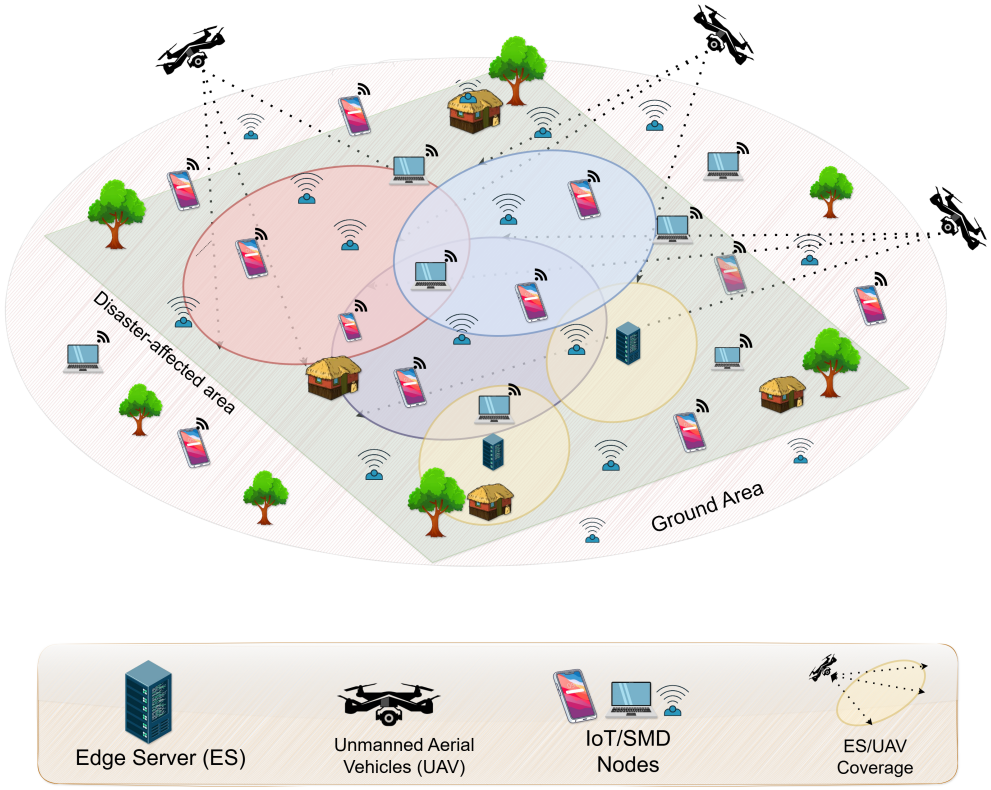


Figure 1: System Model for Disaster Zones

Notation/Symbols	Description/Meaning
IoT	Internet of Thing Device
UAV	Unmanned Aerial Vehicle
ES	Edge Server
PU	Processing Unit
d_i	i^{th} IoT
τ_i	i^{th} task
u_j	j^{th} UAV
e_k	k^{th} ES
p_m	m^{th} PU

Table 2: List of Notations and their Meanings

$$\Phi = \frac{1}{s} \sum_{m=1}^{s} \phi_m \tag{2}$$

Remark 3.1. The value of Φ lies between zero to one.

3.2. Processing Units (PU) Coverage Model

An IoT device may fall under the coverage of more than one processing unit (PU). For an PU to serve an IoT device, the IoT must be within the PU's service area. This means that

for d_i to be allocated, it needs to be within the coverage range of p_m. Let λ_i^m be a boolean variable defined as follows.

$$\lambda_i^m = \begin{cases} 1; & \text{if } p_m \text{ in coverage of } d_i \\ 0; & \text{otherwise} \end{cases} \tag{3}$$

Remark 3.2. Allocating d_i to p_m is possible only when $\lambda_i^m = 1$. Additionally, p_m must have sufficient resources to support $u_i \in \text{cov}(p_m)$.

3.3. User allocation model

The following introduces the boolean variable γ_j^i, which indicates whether d_i is allocated to p_m as defined in (4):

$$\gamma_i^m = \begin{cases} 1; & \text{if } d_i \text{ is allocated to } p_m \\ 0; & \text{otherwise} \end{cases} \tag{4}$$

You can calculate the user allocation rate (UAR) as follows (5):

$$\Gamma = \frac{1}{n} \sum_{m=1}^{s} \sum_{i=1}^{n} \gamma_i^m \tag{5}$$

Remark 3.3. It is important to note that Γ lies within the range of [0, 1].

3.4. Delay Model

Delay is a critical factor that must be considered in disaster applications. Several factors can contribute to delay, including receiving, processing, transmission, and waiting/queuing times. We will discuss each of these delays in detail below.

- Transmission Delay: This section discusses the latency involved in transferring data from an IoT device to an processing unit (PU). Assume that an IoT device d_i uses a transmission channel with a data transmission rate of R_{im} bps (bits per second) to send L_{in} bits to processing unit (PU) p_m Consequently, the transmission delay can be expressed mathematically as:

$$ti_{tran}^{im} = \frac{L_{in}}{R_{im}} \tag{6}$$

  where, $R_{im} = B_0 \times \log_2(1 + SNC)$, B_0 representing the bandwidth of the transmission channel. The Signal-to-Noise ratio at the carrier (SNC) is calculated from Wang et al. (2019a)

- Receiving Delay: The receiving delay can be calculated using the same technique as the transmission delay for L_{out} bits of received data.

$$ti_{rec}^{im} = \frac{L_{out}}{R_{im}} \tag{7}$$

- Computation Delay: The time required for p_m to process the data sent by d_i is referred to as the computation delay. It can be calculated as follows:

$$ti_{comp}^{im} = \frac{L_{in} \times C_b}{R_{im} \times f_b} \tag{8}$$

The quantity of CPU cycles required to process a single bit is denoted by (CPU cycles per bit). The computational capability, or CPU cycle count per second, of p_b is represented by f_b.

- Waiting delay:

Assume that p_m has l tasks in its job queue that need to be processed. These tasks are all offloaded from other IoT devices. In the meantime, a new task τ_i from new IoT device d_i is transferred to p_m. Consequently, d_i must wait until p_m has completed processing all the earlier jobs. The method to estimate the waiting time is as follows:

$$ti_{wait}^{im} = \sum_{i=1}^{l} ti_{comp}^{im} \tag{9}$$

Therefore, the corresponding offloading delay that occurs when τ_i is offloaded to p_m can be calculated as follows:

$$ti_{off}^{i} = \gamma_i^m \times \{ti_{tran}^{im} + ti_{rec}^{im} + ti_{comp}^{im} + ti_{wait}^{im}\} \tag{10}$$

3.5. Energy Model

The sending, receiving, and computing activities of PUs, as well as the propulsion of UAVs, all require energy consumption.

- Transmission Energy: This represents the amount of energy required to transfer data from the IoT to the relevant processing unit (PU). It can be estimated as follows:

$$e_{tran}^{im} = p_{im} \times ti_{tran}^{im} \tag{11}$$

where the power required to transfer a task τ_i from d_i to p_m is denoted by p_{im}.

- Receiving Energy: The receiving delay can be calculated in the same manner as the transmission energy.

$$e_{rec}^{mi} = p_{mi} \times ti_{tran}^{mi} \tag{12}$$

As a result, the total energy consumed for offloading is the sum of the energy used for transmission and reception. It can be expressed as follows:

$$e_{off}^{i} = \alpha_{im} \times \{e_{tran}^{im} + e_{rec}^{mi}\} \tag{13}$$

- Propulsion Energy: The energy required for an unmanned aerial vehicle (UAV) to fly and hover is primarily used by the rotary blades' rotation and the power needed to maintain a constant altitude. Let t^j_{hover} represent the duration of time that UAV u_j hovers. The energy consumption per unit of time due to the rotary blades' rotation is denoted by P_0 and the energy consumption for maintaining altitude is denoted by P_1 Therefore, the hovering energy e^j_{hover} for UAV u_j can be calculated using the following formula:

$$e^j_{hover} = (P_0 + P_1) \times t^j_{hover} \tag{14}$$

The total time the UAV spends sending, waiting, and receiving each task is represented by the hovering time, t^j_{hover}, as seen below.

$$t^j_{hover} = \sum_{v=1}^{y} \{ti^{im}_{tran} + ti^{im}_{rec} + ti^{im}_{wait}\} \tag{15}$$

The energy consumption of flying can be related to three main factors: the blade profile, the induced component, and the parasite component. The power required to rotate the rotary blades is associated with the blade profile. The power needed for induction is related to the induced component. The parasite component depends on the rotor solidity (S), air density (ρ), rotor disc area (A), and fuselage drag ratio (d_r). Additionally, there is a fourth component that addresses the UAV's vertical movement and involves power per unit time, denoted as P_2. The formula for calculating the total flying energy is as follows:

$$e^j_{fly} = f_t[P_0(1 + \frac{3||V_{U,xy}||^2}{V^2_{tip}}) + \frac{1}{2} d_r S \rho A ||V_{U,xy}||^3$$

$$+P_1 \sqrt{(\sqrt{1 + \frac{||V_{U,xy}||^4}{4V_0^2}} - \frac{||V_{U,xy}||^2}{2V_0^2})} + P_2 ||V_{U,z}||] \tag{16}$$

The flight duration of u_j is denoted by f_t, while the horizontal and vertical velocities of the UAV are represented by $V_{U,xy}$ and,$V_{U,z}$ respectively. The mean rotor-induced velocity in hover, V_0, and the rotor blade tip speed, V_{tip} are indicated accordingly. Thus, the propulsion energy of u_j can be calculated as follows:

$$e^j_{prop} = e^j_{fly} + e^j_{hover} \tag{17}$$

The total propulsion energy of all the UAVs is calculated as

$$e_{prop} = \sum_{j=1}^{q} e^j_{prop} \tag{18}$$

Keep in mind that in certain situations, the nearest ES may be assigned all the tasks,

resulting in none of the UAVs being in service. To represent these situations Let's define a Boolean variable in the following way:

$$\lambda_j = \begin{cases} 1; \text{ if } \exists \ d_i \text{ that offloaded to UAV } u_j \\ 0; \text{ otherwise} \end{cases} \tag{19}$$

Therefore, the total energy consumption can be calculated as:

$$E_{Total} = \sum_{i=1}^{n} e_{off}^{i} + \sum_{j=1}^{q} \lambda_j \times e_{prop}^{j} \tag{20}$$

3.6. Problem Formulation

Let's assume that the model is aware of user allocation to ES, ES coverage, user distribution, and analytical parameters such as energy consumption, delay across the system.

Consider the Efficient Resource Based User Processor Allocation (ERUA) problem. Given a set of IoT devices D= $\{d_1, d_2, \ldots d_i, \ldots, d_n\}$ and s processing units, which consist of q Unmanned Aerial Vehicles (UAVs) and r Edge Servers (ESs), the objective is to efficiently allocate the n IoT devices to the s processing units. The goal is to minimize energy consumption and delay while maximizing user allocation and coverage by the Edge Servers. Therefore, the problem can be mathematically formulated as follows:

Maximize: User Allocation Rate (Γ)
Minimize: Processor Allocation(Φ)
Minimize: Total Delay (T_{total})
Minimize: Total Energy Consumption (E_{total})

Subject to:

$$\lambda_i^m \in \{0,1\}; \forall m \in [1,s], \forall i \in [1,n] \tag{21}$$
$$\gamma_i^m \in \{0,1\}; \forall m \in [1,s], \forall i \in [1,n] \tag{22}$$
$$\gamma_i^m + (\lambda_i^m)' = 1; \forall m \in [1,s], \forall i \in [1,n] \tag{23}$$
$$\lambda_j \in \{0,1\} \forall j; 1 \leq j \leq q \tag{24}$$
$$\sum_{m=1}^{s} \gamma_i^m \leq 1; \forall m, 1 \leq i \leq n \tag{25}$$
$$\sum_{i=1}^{n} u_{CPU}^i \times \gamma_i^m \leq e_{CPU}^m \tag{26}$$
$$\sum_{i=1}^{n} u_{RAM}^i \times \gamma_i^m \leq e_{RAM}^m \tag{27}$$
$$\sum_{i=1}^{n} u_{BAN}^i \times \gamma_i^m \leq e_{BAN}^m \tag{28}$$
$$\sum_{i=1}^{n} u_{STO}^i \times \gamma_i^m \leq e_{STO}^m \tag{29}$$
$$\phi_m \in \{0,1\}; \forall 1 \leq m \leq s \tag{30}$$
$$\sum_{m=1}^{s} \phi_m \leq s \tag{31}$$

Constraints (21)-(31) are essential for the operation of the system model. Constraints (21) and (22) ensure that the values of λ_i^m and γ_i^m are either 0 or 1. According to constraint (23), d_i must be within the coverage of p_m if d_i is assigned to p_m. Constraint (24) indicates whether a UAV is utilized during the activity; UAV u_j is used if the boolean variable λ_j equals 1. Constraints (25)–(29) ensure that the resource limits of p_m are not exceeded by the processing power, memory, bandwidth, or storage needs of any IoT device within its coverage. Constraint (30) guarantees that the value of ϕ_m is either 0 or 1, while constraint (31) ensures that the number of PUs used does not exceed the total number of PUs available.

Remark 3.4. Recall that the issue raised in Section 3.6 is a 0-1 Integer Non-linear Problem (0-1 I NLP)

Theorem 1. *The Efficient Resouce-Based User Allocation (ERUA) problem is NP-complete.*

Proof. The bin packing problem (BPP) is known to be NP-complete. The BPP is generalized as ERUA i.e.(BPP $\leq$ p ERUA). Here, the reduction in polynomial time is denoted by $\leq$ p. Consequently, every NP problem can be reduced to ERUA in polynomial time. This means that for every problem B in NP, B $\leq$ p ERUA. Furthermore, it can be claimed that ERUA is NP-complete since ERUA $\in$ NP.

□

4. Overview of Quantum Inspired Cuckoo Search Optimization

4.1. Quantum Computing

Quantum computing stands at the forefront of computational science, offering a revolutionary departure from classical computing paradigms. Unlike classical computing, which relies on binary digits or classical bits, quantum computing harnesses the power of quantum bits, often denoted as qubits. These qubits leverage the principles of quantum mechanics to encode and process information in fundamentally different ways. In quantum computing, information is stored and manipulated through quantum states. While classical bits can only be in a state of 0 or 1, qubits can exist in a superposition of states, allowing them to represent both 0 and 1 simultaneously. This superposition property enables quantum computers to explore multiple computational paths concurrently, offering the potential for exponential speedups over classical algorithms. To visualize the state of qubits, we use the Bloch sphere, a two-dimensional complex vector space. The qubit's state is represented within this sphere, with the poles corresponding to the classical states $|0\rangle$ and $|1\rangle$. Utilizing Dirac notation, we can express the state of a qubit, providing a powerful mathematical framework for quantum computation as Eq.(32).

$$|\psi\rangle = \mu|0\rangle + \omega|1\rangle \tag{32}$$

In the quantum world, the qubit is defined by complex probability amplitudes, denoted by μ and ω. These μ and ω represent the probability amplitudes for finding the qubit in the states $|0\rangle$ and $|1\rangle$. The probabilities of these outcomes are given by the squared magnitudes of the respective probability amplitudes. Specifically, μ^2 represents the probability of finding the qubit in state $\|0\rangle$, while ω^2 denotes the probability of it being in state $\|1\rangle$. It is necessary to satisfy the normalization condition, ensuring that the total probability sums to 1. This condition can be expressed mathematically as:

$$|\mu^2| + |\omega^2| = 1 \tag{33}$$

Ensuring the normalization condition is crucial for establishing the validity and coherence of a qubit, thereby embodying the intrinsic probabilistic essence of quantum mechanics. For a single qubit, denoted as $|q\rangle$, its representation requires a pair of complex numbers, namely $|q\rangle = (\mu, \omega)$, where μ and ω represent the complex probability amplitudes corresponding to states $|0\rangle$ and $|1\rangle$, respectively.

When extending this representation to encompass a quantum system composed of n qubits, tensor products become indispensable for articulating the collective state. Expressing the state of an n-qubit system is encapsulated in Eq.(34):

$$\vec{Q} = [q_1, q_2 \ldots, q_k \ldots q_n] = \left[\begin{array}{c|c|c|c|c|c} \mu_1 & \mu_2 & \ldots & \mu_k & \ldots & \mu_n \\ \omega_1 & \omega_2 & \ldots & \omega_k & \ldots & \omega_n \end{array}\right] \tag{34}$$

Remark 4.1. Within a quantum system containing n qubits, the potential number of representable states expands exponentially to 2^n. Each qubit has the capability to exist in a superposition of states $|0\rangle$ and $|1\rangle$.

Consider a quantum system with 4 qubits. The qubits can be represented as shown in Eq.(35):

$$\vec{Q} = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{-1}{\sqrt{3}} & \frac{-\sqrt{3}}{2} & \frac{-1}{2} \\ \frac{-1}{\sqrt{2}} & \frac{\sqrt{2}}{\sqrt{3}} & \frac{1}{2} & \frac{\sqrt{3}}{2} \end{bmatrix} \tag{35}$$

A given qubit in a quantum system with 4 qubits can simultaneously exist in 16 (2^4) states. The binary combinations for this quantum system are: $|0000\rangle$, $|0001\rangle$, $|0010\rangle$, $|0011\rangle$, $|0100\rangle$, $|0101\rangle$, $|0110\rangle$, $|0111\rangle$, $|1000\rangle$, $|1001\rangle$, $|1010\rangle$, $|1011\rangle$, $|1100\rangle$, $|1101\rangle$, $|1110\rangle$, and $|1111\rangle$. The probability of the state $|0011\rangle$ is given by: $P|0011\rangle = (\frac{1}{\sqrt{2}})^2 \times (\frac{-1}{\sqrt{3}})^2 \times (\frac{1}{2})^2 \times (\frac{\sqrt{3}}{2})^2 = \frac{1}{32}$. Similarly, the probability of the state $|1001\rangle$ is $P|1001\rangle = (\frac{-1}{\sqrt{2}})^2 \times (\frac{-1}{\sqrt{3}})^2 \times (\frac{-\sqrt{3}}{2})^2 \times (\frac{\sqrt{3}}{2})^2 = \frac{3}{32}$.

All calculations performed on qubits adhere to the principles of quantum mechanics. In quantum computing, manipulating quantum information or updating the state of a qubit involves applying quantum gates (Q-gates) such as the NOT gate, rotation gates, Hadamard gate, and others. When a Q-gate is applied to a qubit $|q\rangle = [\mu, \omega]$, it undergoes a transformation to $|q'\rangle = [\mu', \omega']$, satisfying the normalization condition $|\mu'|^2 + |\omega'|^2 = 1$. The specific transformations depend on the type of Q-gate applied, adjusting the probability amplitudes (μ and ω) accordingly. The term "quantum angle" likely pertains to the parameters involved in these gate transformations, dictating the precise rotation or operation applied to the quantum state. The calculation of the quantum angle can be expressed in Eq.(36).

$$\theta_c = arctan(\frac{\omega_c}{\mu_c}) \tag{36}$$

The quantum angle θ_c is also used to express the quantum state of each qubit. The quantum state, depicted in terms of the quantum angle θ_c, is expressed in Eq.(37).

$$\vec{Q} = \begin{bmatrix} \cos(\theta_1) & \cos(\theta_2) & \dots & \cos(\theta_n) \\ \sin(\theta_1) & \sin(\theta_2) & \dots & \sin(\theta_n) \end{bmatrix} \tag{37}$$

4.2. Quantum Inspired Cuckoo Search Optimization(QICSO)

The Quantum-Inspired Cuckoo Search Optimization (QICSO) is a metaheuristic optimization algorithm inspired by both quantum computing principles and the behavior of cuckoo birds. Below is a detailed description of the quantum cuckoo representation, population, fitness function, updating the quantum cuckoo position, and discovering quantum cuckoos using the Lévy flight approach.

4.2.1. Quantum Cuckoo Representation

In the proposed algorithm, quantum eggs, quantum nests, and quantum cuckoos are treated similarly. In any population-based quantum evolutionary algorithm, the representation of an individual is crucial. In the QICSO algorithm, each quantum cuckoo, denoted as $\vec{Q}_p$, can be represented as a complete solution.

$$\vec{Q} = \{\vec{Q}_1, \vec{Q}_2, \dots, \vec{Q}_p, \dots, \vec{Q}_N\} \tag{38}$$

A quantum cuckoo, represented as $\vec{Q}_p$, expresses the comprehensive solution to the optimization problem. The cuckoos are described in terms of qubits in Eq. (39).

$$\vec{Q_p}(f) = [q_{p,1}(f), \dots q_{p,c}(f), \dots q_{p,N_{bits}}(f)] \tag{39}$$

Here, $q_{p,c}(f)$ denotes the state of the c^{th} qubit in the p^{th} quantum cuckoo of the f^{th} generation. Each qubit ($q_{p,c}(f)$) can be in the state of $|0\rangle$ or $|1\rangle$, representing the binary information of the solution. The combination of these qubits forms a quantum cuckoo, which, upon measurement, provides a valid solution for the optimization problem as expressed in Eq (40).

$$\vec{Q}_p(f) = [q_{p,1}(f), \dots q_{p,c}(f), \dots q_{p,N}(f)] = \begin{bmatrix} \mu_{p,1}(f) & \dots & \mu_{p,c}(f) & \dots & \mu_{p,N}(f) \\ \omega_{p,1}(f) & \dots & \omega_{p,c}(f) & \dots & \omega_{p,N}(f) \end{bmatrix} \tag{40}$$

Each component $q_{p,c}(f)$ of $Q_p(f)$ is associated with a quantum angle $\theta_{p,c}(f)$, as expressed in Eq (41).

$$\vec{\Theta}_p(f) = [\theta_{p,1}(f), \dots, \theta_{p,k}(f), \dots \theta_{p,N}(f)] \tag{41}$$

$\theta_{p,c}$ represents the quantum angle signifies the c^{th} component of the p^{th} quantum cuckoo at the f^{th} iteration.

4.2.2. Quantum Cuckoo in Terms of Quantum Cuckoo Angle

For each quantum cuckoo $\vec{Q}_p(f)$ there exist correlated Quantum Angle cuckoo $\vec{\Theta}_p(f)$. This correlation extends to all other cuckoos generated in subsequent operations. Each quantum angle $\theta_{p,c}(f)$ is connected with the c^{th} component of $\vec{Q}_p(f)$. Therefore the Quantum cuckoo expressed in quantum angles in Eq. (42)

For each quantum cuckoo $\vec{Q}_p(f)$, there exists a correlated quantum angle cuckoo $\vec{\Theta}_p(f)$. This correlation extends to all other cuckoos generated in subsequent operations. Each quantum angle $\theta_{p,c}(f)$ is associated with the c^{th} component of $\vec{Q}_p(f)$. Therefore, the quantum cuckoo can be expressed in quantum angles as shown in Eq. (42).

$$\mu_{p,c}(f) = \cos(\theta_{p,c}(f)), \omega_{p,c}(f) = \sin(\theta_{p,c}(f)) \tag{42}$$

The initial quantum population for f=1 is denoted in Eq. (43)

$$\vec{Q}_{POP}(1) = \vec{Q}_1(1), \dots, \vec{Q}_p(1) \dots \vec{Q}_{POP}(1) \tag{43}$$

For $1 \leq$ p $\leq$ POP where POP represents the size of the quantum population. Each $\vec{Q}_{p(}f)$ is derived from the quantum angles $\vec{\Theta}_p(f)$, alongside their corresponding components $\mu_{p,c}(f)$ and $\omega_{p,c}(f)$. This formulation integrates the cosine and sine values of the quantum angles to construct the quantum cuckoo within the quantum population.

4.2.3. Updating the position of the cuckoo and locating eggs through Levy Flight

The Lévy flight serves as a potent feature within the quantum cuckoo search algorithm, facilitating the generation of new candidate solutions (quantum cuckoos). In this context, the Lévy flight approach is used as follows:

$$\vec{\Theta}_p(f+1) = \vec{\Theta}_p(f) + \vec{\chi_p} \tag{44}$$

where, $\Theta_p(f+1)$ is the quantum angle associated with the updated position of the quantum cuckoo, $\Theta_p(f)$ is the quantum angle associated with the current position of the quantum cuckoo with a change of position χ_p which is computed in Eq. (45)

$$\vec{\chi_p} = 0.001 \times S_p \odot (\vec{\Theta}_p(f) - \vec{\Theta}_{best}(f)) \tag{45}$$

In order to compute χ_p a random step denoted as S_p is generated from a symmetric Lévy distribution using element-wise multiplication.

$$S_p = \frac{\mathbf{u}}{|\mathbf{v}|^{\frac{1}{\beta}}} \tag{46}$$

where, $\mathbf{u}$ and $\mathbf{v}$ are the two-dimensional vectors are calculated using the Normal distribution $\mathbf{u} \sim \mathrm{N}(0,{\delta_\mathbf{u}}^2)$ and $\mathbf{v} \sim \mathrm{N}(0,{\delta_\mathbf{v}}^2)$. The δ can be computed in Eq. (47) sin

$$\delta_\mathbf{u} = \left(\frac{\Gamma(1+\beta).\sin(\tau \times \frac{\beta}{2})}{\Gamma\frac{1+\beta}{2}.\beta.2^{\frac{\beta-1}{2}}} \right)^{\frac{1}{\beta}} \tag{47}$$

where, $\Gamma(.)$ is the gamma distribution.

In the subsequent phase, a selection of the quantum angle of the current generation for the subsequent process of QICSO is made between $\Theta_p(f+1)$ quantum angle associated with updated quantum position and $\Theta_p(f)$ quantum angle associated with current quantum position based on the fitness values. The selection operation is performed as follows:

$$\overrightarrow{\Theta_p}(f) = \begin{cases} \overrightarrow{\Theta_p}(f+1), & \text{if } \mathrm{f}(\overrightarrow{\Theta_p}(f+1)) \geq \mathrm{f}(\overrightarrow{\Theta_p}(f)) \\ \overrightarrow{\Theta_p}(f), & \text{otherwise} \end{cases}$$

where, f(.) represents fitness function

In the subsequent phase, cuckoos are replaced through the construction of new ones as described in Eq. (48).

$$\vec{\Theta_p}(f+1) = \begin{cases} \vec{\Theta_p}(f+1) + \epsilon(\vec{\Theta_m}(f) + \vec{\Theta_n}(f), & \text{with probablity } P_a \\ \vec{\Theta_p}(f), & \text{with probability 1-}P_a \end{cases} \tag{48}$$

where, ϵ is the random distribution, and m and n are random integers chosen from 1 to N. During this phase, a subset of the quantum cuckoos is chosen with a certain probability and substituted with a new quantum cuckoo. Each quantum cuckoo is selected for potential replacement based on a probability P_a within the range of [0,1]

4.2.4. Quantum Selection Operation

The quantum selection operation in QICSO involves choosing the quantum cuckoo for the next generation (f+1). This selection is made between $\vec{\Theta}_p(f+1)$ newly generated quantum angle and $\vec{\Theta}_p(f)$ quantum angle of the current generation. The selection operation is performed as follows:

$$\vec{\Theta}_p(f+1) = \begin{cases} \vec{\Theta}_p(f+1), & \text{if f}(\vec{\Theta}_p(f+1) > \text{f}(\vec{\Theta}_p(f)) \\ \\ \vec{\Theta}_p(f+1), & \text{otherwise} \end{cases} \tag{49}$$

The selection operation, as mentioned in the previous response, would then depend on the comparison of these fitness values. Quantum selection operation results in obtaining the quantum angle vector can be expressed in Eq (50)

$$\overrightarrow{Q}_p(f+1) = \begin{bmatrix} \cos(\theta_{p,1}(f+1)) & \dots & \cos(\theta_{p,c}(f+1)) & \dots & \cos(\theta_{p,Dim}(f+1)) \\ \sin(\theta_{p,1}(f+1)) & \dots & \sin(\theta_{p,c}(f+1)) & \dots & \sin(\theta_{i,Dim}(f+1)) \end{bmatrix} \tag{50}$$

$\theta_{p,c}(f+1)$ represents the c^{th} component of the quantum cuckoo for the next generation. This Q-bit representation captures the quantum cuckoo $\overrightarrow{Q}_p(f+1)$ for the next generation. The same is illustrated as a flowchart in the Figure 2

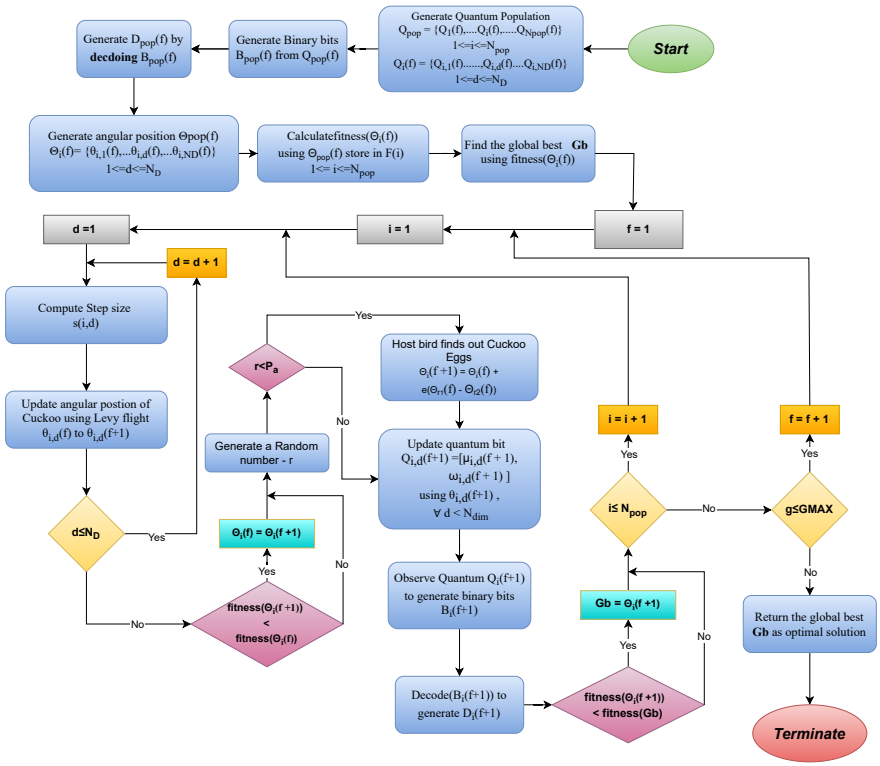


Figure 2: Flowchart of the Quantum Inspired Cuckoo Search Optimization

5. Proposed QICSO for Disaster Management

This section discusses the Quantum Inspired Cuckoo Search Optimization (QICSO) algorithm. The proposed algorithm comprises distinct stages, outlined as follows

5.1. Quantum Cuckoo Initialization and Population generation

Eq (38) and (40) describes the quantum cuckoo initializations for a system with n users and s PU. $N_{Dim} = n \times N_{Bits}$, where $N_{Bits} = \lceil log_2 s \rceil$, is the length or dimension of the QVs. Every quantum vector is a full solution to the ERUA problem. Algorithm 1 displays the pseudo-code used to create the initial quantum population.

Algorithm 1 Quantum_Population(N_{pop}, N_{Dim})

Input:N_{pop} population size and N_{Dim} dimension of QP

Output: Quantum population $Q_{pop}(1) = \left[Q_1(1), Q_2(1), \ldots, Q_p(1), \ldots Q_{N_{pop}}(1)\right]$

1: Initialize $Q_{pop}(1) = \{\ \}$ /*List to store QPs*/
2: **for** each $p \in N_{pop}$
3: Initialize $Q_p(1) = \{\ \}$ /*List to store quantum bits*/
4: **for** each $c \in N_{Dim}$
5: $\mu_{p,c}(1) = random(-1, 1)$ /*computing $\mu_{i,q}(1)$ value*/
6: $\omega_{p,c}(1) = \sqrt{1 - \mu_{p,c}(1)^2}$ /*computing $\omega_{i,q}(1) value$*/
7: $q_{p,c}(1) = \begin{bmatrix} \mu_{p,c}(1) \\ \omega_{p,c}(1) \end{bmatrix}$
8: $Q_p(1) = Q_p(1) \cup q_{p,c}(1)$ /*inserting qubit in $Q_p(1)$*/
9: **end for**
10: $Q_{pop}(1) = Q_{pop}(1) \cup Q_p(1)$ /*inserting QP in $Q_p(1)$*/
11: **end for**
12: **return** $Q_{pop}(1)$

Illustration 5.1. Suppose we have an area A with six PUs ($s = 6$) and fourteen IoTs ($n = 14$). The minimal number of qubits needed to indicate the index of s = 6 PUs is N_{Bits} = $\lceil \log_2 s \rceil = \lceil log_2 6 \rceil = 3$.

Therefore, $N_{Dim} = n \times N_{Bits} = 14 \times 3 = 42$ is the QVs dimension (N_{Dim}). The N_{Bits} length qubits are grouped for each d_i, representing the index of the PU (in this case, N_{Bits} = 3) to which the d_i can be allocated as shown in the Table 3.

IoT	d_1	d_2	d_3	...	d_{13}	d_{14}
PU_{index}	$\begin{pmatrix} q_{c,1}(f) \\ q_{c,2}(f) \\ q_{c,3}(f) \end{pmatrix}$	$\begin{pmatrix} q_{c,4}(f) \\ q_{c,5}(f) \\ q_{c,6}(f) \end{pmatrix}$	$\begin{pmatrix} q_{c,7}(f) \\ q_{c,8}(f) \\ q_{c,9}(f) \end{pmatrix}$	...	$\begin{pmatrix} q_{c,37}(f) \\ q_{c,38}(f) \\ q_{c,39}(f) \end{pmatrix}$	$\begin{pmatrix} q_{c,40}(f) \\ q_{c,41}(f) \\ q_{c,42}(f) \end{pmatrix}$

Table 3: Quantum vector representation of Illustration 1

Lemma 5.1. The time complexity of Quantum_Population (N_{pop}, N_{Dim}) is $\Theta(N_{Pop} \times n \times \log_2 s)$.

Proof. Algorithm 1 can construct a QV of dimension N_{Dim} in $\Theta(N_{Dim})$ using lines 4-8. It is possible to generate the Quantum population with N_{pop} QV's in $\Theta(N_{pop} \times N_{Dim})$ time. Given that $N_{Dim} = n \times [log_2 s]$, the worst time complexity is $\Theta(N_{pop} \times n \times [\log_2 s])$. □

Remark 5.1. Systems of quantum bits n such that a 2^n state with amplitude equal to the district probability can represent the system. A district solution of 2^n possibilities for the above problem can be represented by a single QP with N_{Dim}. We can therefore draw the conclusion that even with a smaller population, more potential solutions can be investigated.

5.2. Observation

The $\vec{\mathbf{Q}}_p$(f) is observed to generate the binary vector (BV) B_p(f) = {$b_{p,1}$(f),$b_{p,2}$(f),$b_{p,3}$(f), . . . ,$b_{p,N_{dim}}$(f)} as

$$b_{p,q}(f) = \begin{cases} 0; & rand[0,1] \leq |(\mu_{p,q}(f))|^2 \\ 1; & \text{otherwise} \end{cases} \tag{51}$$

The observation algorithm to generate the BVs is shown in Algorithm 2. Now, the BV is further decoded to generate the complete solution.

Illustration 5.2. Consider a example in Illustration 5.1. After observing the Quantum Vector (QV), the Binary Vector (BV) is generated as shown in Table 4

IoT	d_1	d_2	d_3	d_4	d_5	...	d_{12}	d_{13}	d_{14}
PU_{index}	010	101	000	001	000	...	010	111	000

Table 4: Quantum vector representation of Illustration 1

Algorithm 2 Observation($\vec{Q}_p(f)$)

Input:Quantum particle, $\vec{Q}_p(f) = [q_{p,1}(f) \ldots, q_{p,c}(f), \ldots, q_{p,N_{Dim}}(f)]$
Output: Binary particle, $\vec{B}_p(f) = [b_{p,1}(f) \ldots, b_{p,c}(f), \ldots, b_{p,N_{Dim}}(f)]$.

```
for each c ∈ N_Dim
    r = random(0, 1)
    if (r < |μ_{p,c}(f)|^2 then                    /* computing value of binary bits */
        b_{p,c}(f) = 0
    else:
        b_{p,c}(f) = 1
    end if
end for
return B_p(f)
```

Lemma 5.2. The time complexity of Observation($\vec{Q}_p(f)$) is $\Theta(N_{Dim})$.

Proof. Each quantum bit is observed, and the corresponding binary bit is generated during the observation phase. The dimension of each QP is N_{Dim}. Hence, the time complexity is $\Theta(N_{Dim})$ or $\Theta(n \times [\log_2 s])$. □

Remark 5.2. It's important to note that in this context, N_{bits} bits are utilized to represent the allocation of a $\vec{Q}$ to $\vec{B}$ within the valid range of (1,m). However, N_{bits} bits have the potential to generate decimal numbers beyond the valid range (1,m). To address this, a linear hashing technique is employed to map the range, as discussed in the following subsection.

5.3. Decoding and Hashing

The observed binary string of length N_{Dim}= n × N_{bits} is split into a task group of N_{bits} length binary string during the decoding phase, and the resulting decimal values are then decoded $\vec{\mathbf{D}}_p(f) = \{x_{p,1}(f), \ldots, x_{p,i(f)}, \ldots, x_{p,n}(f)\}$. Here,$x_{p,i}(f)$ is the equivalent decimal number of the binary string $\{b_{p,[(c-1)\times Bits+1]}(f), \ldots, b_{p,[(c)\times Bits]}(f)\}$. Following decoding, the final $x_{p,i}(f)$ indicates the PU to which the u_i is assigned; the decoded value, it should be noted, falls between 0 and $2^{N_{Bits}} - 1$.

Algorithm 3 shows the pseudocode that is used for decoding. For every binary string of length N_{Dim}, the corresponding decimal value to a decoded decimal string is indicated.

Algorithm 3 Decoding($\vec{B}_p(f)$)

Input:Observed Binary bits: $\vec{B}_p(f) = [b_{p,1}(f), \ldots, b_{p,c}(f), \ldots, b_{p,N_{Dim}}(f)]$.
Output: Decoded bits: $\vec{D}_p(f) = [x_{p,1}(f), \ldots, x_{p,i}(f), \ldots, x_{p,n}(f)]$.

Initialize $D_p(f) = \{\ \}$ /* List to store decode decimal values */
Initialize Base $= 2^{N_{bits}-1}, i = 1$
Initialize $x_{p,i}(f) = 0$ /* Initializing decimal value to zero */
for each $c \in N_{Dim}$ **do**
    $x_{p,i}(f) = x_{p,i}(f) + b_{p,c}(f) \times Base$
    $Base/ = 2$
    if $c\%N_{bits} == 0$ **then**
        $x_{p,i}(f) =$ Hashing($x_{p,i}(f)$)
        $D_p(f) = D_p(f) \cup x_{p,i}(f)$ /* Inserting decoded decimal value into $D_p(f)$ */
        $Base = 2^{N_{bits}} - 1$
        $i = i + 1$ /* Incrementing index value */
        $x_{p,i}(f) = 0$ /* Resetting decimal value */
    end if
end for
return $D_p(f)$

And to note that every u_i has a set of PUs, and that PUs in $cov(u_i)$ cover the u_i.The $([x_{p,i}(f) + 1])^{th}$ PU of the set cov(u_i) receives the u_i. Be aware that the (cov(u_i)) does not have the required number of PUs, ($[x_{p,i}(f) + 1]$), or that the assigned PU may not have enough resources to serve u_i. For additional mapping, a linear hashing algorithm is applied in both situations. The purpose of the linear hashing is to hash ($x_{p,i}(f)$) to an index number of the set (cov(u_i)). Algorithm 3 provides the BV's decoding. The hashing Algorithm 4 is called for the mapping after a Bits-length binary string has been decoded.

Illustration 5.3. The binary vector in the Table 4 is decoded using the decoding Algorithm 3 to create the decoded vector as shown in the Table 5. Its seen that the corresponding decoded value of d_1 is in the Table 5. This indicates that d_1 is allocated to the $(2 + 1)^{th} = 3^{rd}$PU of cov(d_1). Its done similarly for all IoTs according to their allocation with PUs.

Lemma 5.3. The time complexity of Decoding($\vec{B}_p(f)$) is $\Theta(n \times s \times [\log_2 s])$.

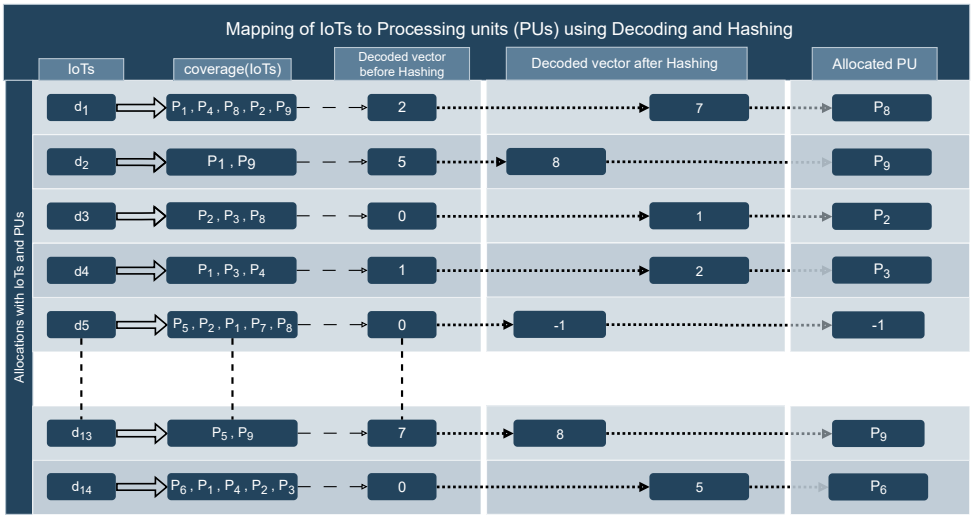


Figure 3: Mapping of IoTs with PUs with Decoding and Hashing

DV	$x_{p,1}(f)$	$x_{p,2}(f)$	$x_{p,3}(f)$	$x_{p,4}(f)$	$x_{p,5}(f)$	...	$x_{p,12}(f)$	$x_{p,13}(f)$	$x_{p,14}(f)$
IoT	d_1	d_2	d_3	d_4	d_5	...	d_{12}	d_{13}	d_{14}
PU_{index}	2	5	0	1	0	...	2	7	0

Table 5: Decoded vector representation (Before hashing) for Binary vector of Illustration 1

Proof. In the decoding phase, a binary vector of length $N_{Dim} = n \times [\log_2 s]$ is divided into binary strings of length N_{bits} and converted to their decimal equivalent. Moreover, Hashing($x_{p,i}(f)$)nis called for all $x_{p,i}(f)$ can be executed in $\Theta(s)$ time. Hence, overall time complexity of Decoding($\vec{B}_p(f)$) is $\Theta(n \times s \times [\log_2 s])$. □

Illustration 5.4. After decoding, if the $x_{p,i}(f)$ is not in the range of coverage of d_i. Then, $x_{p,i}(f)$ should be hashed accordingly such that $h(x_{p,i}(f))$ should belong to $[0, |cov(d_i)| - 1]$. This can be observed for d_2 i,e. Before hashing it was seen that $x_{p,i} = 5$ and it was hashed to 2. Therefore, d_2 is allocated to 2^{nd} PU. The decoding and hashing of all the IoTs are shown in the Figure 3

5.4. Fitness function

For the decoded vector, $\vec{D}_p(f) = [x_{p,1}(f) \ldots, x_{p,i}(f), \ldots, x_{p,N}(f)]$, the Fitness is computed. The fitness function is meant to meet the objectives outlined in Paragraph 3.6. As previously stated, the initial objective (f_1) is to maximize the UAR. The second goal (f_2) is to reduce the quantity of processors needed for task offloading. The third goal (f_3) is to reduce the system's overall delay. The last and fourth objective (f_4) is to reduce the amount of energy required. The goals taken into account for the fitness function are displayed in

Algorithm 4 Hashing($x_{p,i}(f)$)

Input: Decoded decimal, $x_{p,i}(f)$
Output: Hashed decimal, $x_{p,i}(f)$

Initialize $flag = true$ /* Boolean variable indicating sufficient resources */
Initialize $cov\_len = len(cov(u_i))$
if not sufficient resources in $[x_{p,i}(f)+1]^{th}$ PU of $cov(u_i)$ **then**
    $flag = false$
end if
if $x_{p,i}(f) < 0$ or $x_{p,i}(f) > cov\_len - 1$ or $flag \neq true$ **then**
    while $flag \neq true$ or $cov\_len > 0$ **do**
        $h(x_{p,i}(f)) = [x_{p,i}(f) \% cov\_len] + 1$
        if $[h(x_{p,i}(f))]^{th}$ PU of $cov(u_i)$ has sufficient resources **then**
            $flag = true$
            Let $p_t = [h(x_{p,i}(f))]^{th}$ PU of $cov(u_i)$
            t is the index of the PU corresponding to $[h(x_{p,i}(f))]^{th}$ in $cov(u_i)$
            $x_{p,i}(f) = t - 1$
        else
            Remove $[h(x_{p,i}(f))]^{th}$ PU from $cov(u_i)$
            $cov\_len = cov\_len - 1$
            $x_{p,i}(f) = x_{p,i}(f) - 1$
        end if
    end while
    if flag $\neq true$ **then**
        $x_{p,i}(f) = -1$
    end if
else
    Let P_t be the $[x_{p,i}(f)+1]^{th}$ PU of $cov(u_i)$
    t is the index of the PU corresponding to $[x_{p,i}(f)]^{th}$ in $cov(u_i)$
    $x_{p,i}(f) = t - 1$
end if
return $x_{p,i}(f)$

$$\text{Maximize:} \quad f_1 = \frac{1}{n}\sum_{i=1}^{n} \eta_i \tag{52}$$

$$\text{Minimize:} \quad f_2 = \frac{1}{s}\sum_{m=1}^{s} \phi_m \tag{53}$$

$$\text{Minimize:} \quad f_3 = \sum_{i=1}^{a} e^i_{off} + \sum_{j=1}^{b} \lambda_j \times e^j_{prop} \tag{54}$$

$$\text{Minimize:} \quad f_4 = ti^{im}_{tran} + ti^{im}_{rec} + ti^{im}_{comp} + ti^{im}_{wait} \tag{55}$$

In the above Eq (52)-(55), $\eta_p = 1$ if $x_{p,i}(f) \neq -1$, otherwise $\eta_p = 0$. Eq. (1) and Eq.(4), respectively, compute the variables ϕ_t and γ_i^t, with t= $x_{p,i}(f) + 1$. The fitness function is now designed using the weighted sum technique in the following manner:

$$\text{Minimize} \quad F = w_1.(1 - f_1) + w_2.f_2 + w_3.f_3 + w_4.f_4 \tag{56}$$

where, w_1, w_2, w_3 and w_4 are weight values such that, $\sum_{j=1}^{4} w_j = 1$ and $0 \leq w_j \leq 1$. The optimal solution is the one (decoded vector) with the lowest fitness value. Provided in Algorithm 5 is the pseudo-code used to compute the fitness value.

Algorithm 5 Fitness ($\vec{D}_p(f)$)

Input: Decoded bits: $\vec{D}_p(f) = [x_{p,1}(f) \ldots, x_{p,q}(f), \ldots, x_{p,a}(f)]$.
Output: Fitness

1: Initialize w_1, w_2, w_3, w_4 /* Note that sum of weights should be 1 */
2: Initialize $f_1 = f_2 = f_3 = f_4 = 0$
3: Initialize $\phi_m = 0, \forall m, 1 \leq m \leq d$ /* Initializing phi for all PU to zero */
4: **for** each $q = 1$ to a **do**
5: **if** $x_{p,q}(f) \neq -1$ **then**
6: $t = x_{p,q}(f) + 1$ /*u_q is allocated to e_t */
7: $f_1 = f_1 + \frac{1}{a}$
8: **if** $\phi_t == 0$ **then**
9: $\phi_t = 1$
10: $f_2 = f_2 + \frac{\phi_t}{d}$
11: **end if**
12: $f_3 = f_3 + \frac{L_{in}}{R} + \frac{L_{out}}{R} + (d-1) \times L_{in} \times C_b/f_b$
13: $f_4 = f_4 + P_{jb} * (\frac{L_{in}}{R} + \frac{L_{out}}{R}) + c \times (f_b^3) \times L_{in} \times C_b/f_b$
14: **if** $x_{i,d} \in$ UAV **then**
15: $f_4 = f_4 + (P_0 + P_1) f_3$
16: $f_4 = f_4 + P_0(1 + \frac{3||V_{U,xy}||^2}{V_{tip}^2})$
17: $f_4 = f_4 + \frac{1}{2} d_r S \rho G ||V_{U,xy}||^3$
18: $f_4 = f_4 + P_1 \sqrt{(\sqrt{1 + \frac{||V_{U,xy}||^4}{4V_0^2}} - \frac{||V_{U,xy}||^2}{2V_0^2})}$
19: $f_4 = f_4 + P_2 ||V_{U,z}||$
20: **end if**
21: **end if**
22: $Fitness = w_1 \times (1 - f_1) + w_2 \times f_2 + w_3 \times f_3 + w_4 \times f_4$
23: **end for**
24: **return** $Fitness$

Lemma 5.4. The time complexity of Fitness($\vec{D}_p(f)$) is $\Theta(n)$.

Proof. Line 3 requires $\Theta(n)$ time to initialize (see Algorithm 5). Lines 4 through 10 calculate the f_1, f_2, and this takes $\Theta(n)$ time. Next, in lines 12 - 19, f_3, f_4 are calculated, using $\Theta(m)$ time. As $n > m$, the Fitness($\vec{D}_p(f)$) has a worst-case time complexity of $\Theta(a)$. □

5.5. Updation of angle

Eq. (36) is used to compute the quantum angular $\Theta(f)$. The new QP $Q(f+1)$ is updated using angular $\Theta(f+1)$, as specified by Eq. (48). Using Algorithms 2 and 3, respectively, the updated QP $Q(f+1)$ is observed and decoded. One can calculate fitness using the decoded QP. If the best fitness is found in each iteration, the quantum particle best and quantum global best are updated. Until the maximum number of iterations, F_{max}, this iteration procedure is carried out. Algorithm 6 provides the whole QICSO algorithm for the ERUA problem.

Algorithm 6 *QI_CSO()*

Input: Initial Population size N_{pop}, max iteration f_{max}, dimension N_{Dim}
Output: Decoded global best solution $D_i(f)$

Initialize $f = 1$
$\vec{Q}_{Pop}(f) = Quantum\_Population(N_{pop}, N_{Dim})$ /* Algorithm 1 */
for $i = 1$ to N_{pop} **do**
    Compute quantum angle $\vec{\Theta}_i(f)$ /* Eq (36),(37) */
    $\vec{B}_i(f) = Observation(\vec{Q}_i(f))$ /* Algorithm 2 */
    $\vec{D}_i(f) = Decoding(\vec{B}_i(f))$ /* Using Algorithm 3 */
end for
$\vec{Gb} = \vec{\Theta}_i(f)$, such that $Fitness(\vec{\Theta}_i(f)) = \min\{Fitness(\vec{\Theta}_x(f)) \,|\, \forall x, 1 \leq x \leq N_{Pop}\}$
while $g \leq F_{max}$ **do** :
    for $i = 1$ to N_{pop}:
        Compute Stepsize S_i /* Using Eq (46) */
        Calculate Levy Flight $\vec{\chi}_i$ /* Using Eq (45) */
        $\vec{\Theta}_i(f+1) = \vec{\Theta}_i(f) + \vec{\chi}_i$ /* Updating position of Cuckoo Using Levy flight */
        if Fitness$(\vec{\Theta}_i(f+1)) \leq$ Fitness$(\vec{\Theta}_i(f))$ **then** :
            $\vec{\Theta}_i(f) = \vec{\Theta}_i(f+1)$
        end if
        r = Get_Random(0,1) /*replacement and construction of a new cuckoo */
        if $r \leq P_a$ **then** : /*host bird finds the Cuckoo eggs */
            $\vec{\Theta}_i(f+1) = \vec{\Theta}_i(f) + \epsilon(\vec{\Theta}_{r1}(f) - \Theta_{r2}(f))$
        end if
        Update Cuckoo Population $\vec{Q_{pop_i}}(f+1)$ using $\vec{\Theta}_i(f+1)$ Using Eq (50)
        $B_i(f+1) = Observation\_State(\vec{Q_{pop_i}}(f+1))$ /* Algorithm 2 */
        $D_i(f+1) = Decoding(B_i(f+1))$ /* Algorithm 3 */
        if Fitness$(\vec{\Theta}i(f+1)) \leq Fitness(\vec{Gb})$ **then** :
            $Gb = \vec{\Theta}_i(f+1)$ /* Storing the Global Best Solution */
        end if
    end for$f = f + 1$
end while
$\vec{Gb} = \vec{\Theta}_i(f)$, such that $Fitness(\vec{\Theta}_i(f)) = \min\{Fitness(\vec{\Theta}_x(f)) \,|\, \forall x, 1 \leq x \leq N_{Pop}\}$
$\vec{D}_{gb} = \vec{D}_i(f)$
return $\vec{D}_{gb}$

Lemma 5.5. The time complexity of QICSO is $\Theta(N_{pop} \times N_{Dim} \times F_{max})$ or $\Theta(N_{pop} \times a \times [\log_2 s] \times F_{max})$.

Proof. Referencing the Lemma 5.2, the QV's initial population is formed in the time $\Theta(N_{pop} \times N_{Dim})$. $\Theta(N_{Dim})$ (Ref to the Lemma 5.2 and 5.3) consists of the binary vector observation stage and the QP decoding stage. In $\Theta(N_{Dim})$ time, the angular updating's QV is completed. Fitness values are calculated using the same formula from the QVs' decoded phase: $\Theta(\tau_n)$. □

6. Results

In this section, we are going to evaluate the performance of the proposed QICSO using two different scenarios. Then the hypothesis-based statistical analyses (ANOVA) and Friedman Test are going to be conducted. Finally, the Design of the Experiment (DoE) is carried out using the Taguchi method.

6.1. System specifications, parameters and Simulation setup

Python 3.11 is used for all analysis and simulations. The proposed Quantum-Inspired Cuckoo Search Optimization (QICSO) is compared with Quantum Differential Evolution (QDE Bandyopadhyay et al. (2024)), Particle Swarm Optimization (PSO Maratha et al. (2021)), and Cuckoo Search Optimization (CSO Bakshi et al. (2023)). The maximum generation count G_{MAX} for all algorithms is fixed at 100, while the population size is fixed at 100 for all algorithms.

For each algorithm, the fitness evaluation function Calculate_Fitness($\vec{A}_i(f)$) is used, as detailed in Algorithm 5 In QICSO, the Probability of abandonment(P_a) is taken as 0.5, the step-size scaling factor(ϵ) is taken as 0.002, and β as 3/2. The crossover and mutation rate for QDE are taken as 0.8 and 0.5 respectively. Table 6 shows all the variables and parameters used in research simulations.

Notation	Values	Notation	Values
B_0	1 000 000 Hz	d_r	0.3
$L_{\text{in}}, L_{\text{out}}$	400 MB	v_{tip}	120 m/s
P_0	158.76 W	v_0	4.03 m/s
P_1	88.63 W	α	20
P_2	11.46 W	ρ	1.225 kg/m^3
S	0.05	f_t	3600 s
A	0.503 m^2	f_{UAV}	4–6 GHz
P_{bj}, P_{jb}	0.2 W	f_{EC}	2 GHz
$C_{\text{UAV}}, C_{\text{EC}}$	1000 cycles/byte	k	10^{-27}
$V_{U,xy}$	20 m/s	SNC	20 db
$V_{U,z}$	5 m/s	w_1, w_2, w_3, w_4	0.2, 0.1, 0.4, 0.3

Table 6: Notation and values.

6.2. Simulation scenarios and Result Comparison

Let us consider a situation where a significant disaster event such as a tsunami, earthquake, or severe flooding occurred in a populated region resulting in widespread infrastructure damage. The disaster led to extensive road blockages, severe interruptions in communication networks, and limited access to essential resources. The area is divided into several impact zones, which makes it difficult for humanitarian aid to coordinate with the affected areas. In anticipation of such events, IoTs had previously been deployed within this disaster-prone region. Some remote ESs and UAVs are deployed and connected to the IoTs in the affected environment. For example, in this scenario, let us consider 15 different devices requests originating from 15 different IoTs, represented as $D_i = \{d_1, d_2, \ldots, d_{15}\}$, and 5 PUs deployed including 3 edge servers $\{e_1, e_2, e_3\}$ and 2 UAVs $\{u_1, u_2\}$. A small example of how the task will be offloaded to PUs using the Quantum Cuckoo Search Optimization algorithm is shown in Table 7.

IoT Device	d_1	d_2	d_3	d_4	d_5	d_6	d_7	d_8	d_9	d_{10}	d_{11}	d_{12}	d_{13}	d_{14}	d_{15}	**Fitness**
Allocated PUs	e_1	e_2	u_1	e_1	e_2	e_3	u_2	e_1	e_2	e_3	u_1	e_2	e_3	u_2	e_1	3 145 250.89

Table 7: Offloading 15 IoT Devices in 5 PUs with 3 ESs, $\{e_1, e_2, e_3\}$ and 2 UAVs, $\{u_1, u_2\}$, using QICSO.

Two more scenarios are taken for in-depth understanding and analysis of the algorithm.

6.2.1. Scenario 1: Dense urban environment

A highly populated urban area with a greater number of Edge servers. This scenario includes two experiments:

- *Experiment 1:* In this experiment, we consider a setup with 6 Processing Units (PUs) comprising 4 Edge Servers (ESs) and 2 UAVs. Between 5 and 15 IoTs generate tasks that are offloaded to the PUs. Table 8 provides a detailed distribution of tasks offloaded to processing units (PUs). It indicates that the tasks are predominantly allocated to ESs.

- *Experiment 2:* Here, the number of IoT devices generated tasks varies between 100, 200, 500, and 1000. To handle these tasks, 20 PUs are deployed, including 12 ESs and 8 UAVs. The experiment records different fitness values, specifically for delay, energy consumption, and overall fitness, using four algorithms: QICSO, PSO Maratha et al. (2021), DE Bandyopadhyay et al. (2024), and CSO Bakshi et al. (2023). Each algorithm is run for 100 iterations, with a population size of 100. Figures 4(a), 5(a), and 6(a) Illustrate the results of the experiment by showing the fitness values for delay, energy, and total fitness, respectively. Additionally, Tables 10, 11, and 12 provide the detailed values for each metric. The results show that the proposed QICSO algorithm outperforms CSO Bakshi et al. (2023), PSO Maratha et al. (2021), and DE Bandyopadhyay et al. (2024) in terms of evaluated fitness metrics.

6.2.2. Scenario 2: Deployment in a Remote Plain Area

In this scenario, the area of interest is plain land situated far from the city, characterized by a sparse distribution of edge servers. Due to the limited number of edge servers, a higher number of UAVs are required to handle task offloading efficiently.

- *Experiment 1:* Similar to Scenario 1, this experiment offloads between 5 to 15 tasks to 4 UAVs and 2 edge servers (ESs). Table 9 indicates that the tasks are mostly allocated to UAVs.

- *Experiment 2:* In this experiment, a similar number of tasks are offloaded, but the configuration differs, involving 10 PUs supported by only 6 edge servers and 4 UAVs, due to its far location from the urban areas. The experimental setup uses the same parameter values as in Experiment 1. Figures 4(b), 5(b), and 6(b) depict the results, Illustrating the fitness values for delay, energy consumption, and overall fitness, respectively. Additionally, Tables 13, 14, and 15 provide the detailed values for each metric. The results indicate that the performance in Scenario 1 is generally better for all algorithms compared to Scenario 2. This improvement can be attributed to the increased availability of resources, specifically the higher number of PUs in Scenario 1 than in Scenario 2.

IoT	d_1	d_2	d_3	d_4	d_5	d_6	d_7	d_8	d_9	d_{10}	d_{11}	d_{12}	d_{13}	d_{14}	d_{15}
6	e_2	e_3	e_1	e_4	e_4	e_4									
7	e_3	u_1	e_3	e_2	e_2	e_2	e_4								
8	u_1	e_4	e_3	e_3	e_2	e_4	e_4	e_1							
9	e_3	e_3	e_4	e_2	e_3	e_2	e_4	e_4	e_3						
10	e_3	e_3	e_2	e_4	e_4	e_3	e_3	e_1	e_4	e_1					
11	e_1	e_4	e_4	e_3	e_4	u_1	e_3	u_2	e_3	e_1	u_1				
12	e_1	e_4	e_4	e_3	u_1	e_3	e_2	e_2	e_2	e_3	e_4	e_3			
13	u_1	e_3	e_1	e_2	e_4	e_3	e_1	e_1	e_4	e_3	e_1	e_1	u_2		
14	e_1	u_1	u_1	e_2	e_4	e_1	e_3	e_2	e_3	e_4	e_3	e_3	e_1	u_2	
15	e_1	e_4	e_3	e_3	e_2	e_2	e_1	e_1	e_4	u_1	e_3	u_1	e_1	e_3	u_2

Table 8: Allocating IoTs to 4 ESs $\{e_1, e_2, e_3, e_4\}$ and 2 UAVs $\{u_1, u_2\}$ using QICSO algorithm.

6.3. Analysis of Variance

Various statistical methods are available to assess the significance of the results. In this context, Analysis of Variance (ANOVA) was employed to evaluate whether the experimental outcomes significantly differ from those produced by other algorithms such as DE Bandyopadhyay et al. (2024), PSO Maratha et al. (2021), CSO Bakshi et al. (2023), and QICSO. ANOVA helps to compare the means of the groups under consideration using the null hypothesis (H_0) and the alternative hypothesis (H_1). Null Hypothesis (H_0): All algorithms have equal performance, meaning the group means are the same, $H_{\mathcal{O}}$: δ (PSO (Maratha et al. (2021))) = δ(DE (Bandyopadhyay et al. (2024))) = δ(CSO (Bakshi et al. (2023))) = δ(QICSO).

Alternate Hypothesis(H_A): The algorithms performances are not equal, meaning their group means differ significantly, i.e.,δ(PSO (Maratha et al. (2021))) $\neq$ δ(DE (Bandyopadhyay

IoT	d_1	d_2	d_3	d_4	d_5	d_6	d_7	d_8	d_9	d_{10}	d_{11}	d_{12}	d_{13}	d_{14}	d_{15}
6	$u4$	$u1$	$u3$	$u3$	$u4$	$u2$									
7	$u3$	$e1$	$u1$	$u4$	$u4$	$u1$	$u2$								
8	$u2$	$u2$	$u2$	$e1$	$u4$	$u3$	$u3$	$u1$							
9	$u1$	$u1$	$u2$	$u4$	$u1$	$u4$	$u3$	$u4$	$u3$						
10	$e1$	$u1$	$u4$	$u2$	$u2$	$u3$	$u3$	$u1$	$e2$	$u2$					
11	$e1$	$u3$	$u3$	$u1$	$u3$	$u4$	$u1$	$e2$	$u2$	$e2$	$u1$				
12	$u3$	$u2$	$u1$	$u2$	$u1$	$u1$	$u4$	$u4$	$u4$	$u3$	$u1$	$u1$			
13	$e2$	$u4$	$u2$	$u4$	$u3$	$u1$	$u3$	$u2$	$e2$	$u1$	$e2$	$u1$	$u1$		
14	$e2$	$u1$	$u1$	$u4$	$u2$	$u1$	$e2$	$u1$	$u2$	$u3$	$e2$	$e2$	$u1$	$u1$	
15	$u4$	$u3$	$u3$	$u1$	$u4$	$u1$	$e1$	$u1$	$u2$	$u3$	$e1$	$u2$	$u1$	$e2$	$e1$

Table 9: Allocating IoTs to 4 UAVs $\{u_1, u_2, u_3, u_4\}$ and 2 ESs $\{e_1, e_2\}$ using QICSO algorithm.

Number of tasks	100	500	1000	2000	3000
PSO Maratha et al. (2021)	0.74281	0.86145	0.90994	0.89083	0.93045
CSO Bakshi et al. (2023)	0.66021	0.80290	0.90536	0.89451	0.90729
DE Bandyopadhyay et al. (2024)	0.60377	0.85855	0.89836	0.92215	0.95539
QICSO	**0.61552**	**0.81583**	**0.85978**	**0.88436**	**0.89096**

Table 10: Comparison in terms of Total Delay for scenario-1, experiment-2.

Number of tasks	100	500	1000	2000	3000
PSO Maratha et al. (2021)	0.64168	0.76983	0.79540	0.86266	0.89614
CSO Bakshi et al. (2023)	0.60367	0.82580	0.88918	0.79777	0.88744
DE Bandyopadhyay et al. (2024)	0.57086	0.76030	0.82107	0.87235	0.88428
QICSO	**0.54815**	**0.68361**	**0.70086**	**0.73083**	**0.77745**

Table 11: Comparison in terms of Energy Consumption for scenario-1, experiment-2.

Number of tasks	100	500	1000	2000	3000
PSO Maratha et al. (2021)	0.69012	0.71056	0.75076	0.85318	0.83573
CSO Bakshi et al. (2023)	0.55471	0.73512	0.73034	0.84107	0.85363
DE Bandyopadhyay et al. (2024)	0.56965	0.66740	0.71587	0.80089	0.80976
QICSO	**0.54265**	**0.66341**	**0.69797**	**0.78619**	**0.80022**

Table 12: Comparison in terms Fitness for scenario-1, experiment-2.

et al. (2024))) $\neq$ δ(CSO(Bakshi et al. (2023))) $\neq$ δ(QICSO). The ANOVA test results for the dataset, presented in Tables 16 and 17, reveal significant findings. The P-value is below the α level, and the F-statistic surpasses the F-critical value. This indicates notable differences between the proposed algorithm and other algorithms, including, PSO (Maratha et al. (2021)), DE (Bandyopadhyay et al. (2024)), CSO (Bakshi et al. (2023)), and QICSO,

Number of tasks	100	500	1000	2000	3000
PSO Maratha et al. (2021)	0.69563	0.88383	0.87258	0.94075	0.97588
CSO Bakshi et al. (2023)	0.69955	0.87116	0.86613	0.91155	0.94846
DE Bandyopadhyay et al. (2024)	0.73700	0.88580	0.90627	0.96249	0.96583
QICSO	**0.68231**	**0.91072**	**0.84335**	**0.91195**	**0.90180**

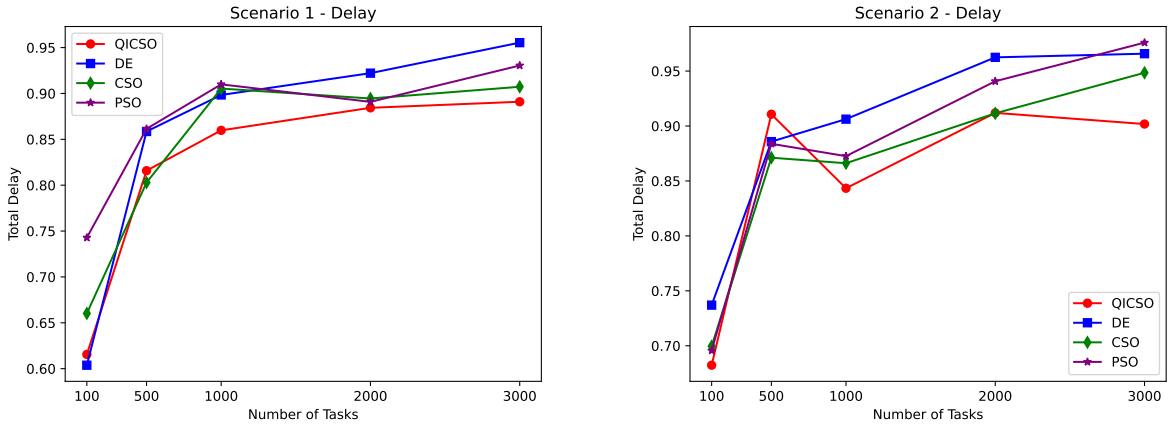
Table 13: Comparison in terms of Total Delay for scenario-2, experiment-2.

Number of tasks	100	500	1000	2000	3000
PSO Maratha et al. (2021)	0.71659	0.74431	0.78048	0.87575	0.91776
CSO Bakshi et al. (2023)	0.34180	0.51464	0.79144	0.87133	0.88714
DE Bandyopadhyay et al. (2024)	0.46532	0.56999	0.76910	0.89363	0.90054
QICSO	**0.33341**	**0.38751**	**0.62098**	**0.71407**	**0.74503**

Table 14: Comparison in terms of Energy Consumption for scenario-2, experiment-2.

Number of tasks	100	500	1000	2000	3000
PSO Maratha et al. (2021)	0.55858	0.61560	0.71447	0.73773	0.78325
CSO Bakshi et al. (2023)	0.53255	0.64324	0.68990	0.74360	0.76205
DE Bandyopadhyay et al. (2024)	0.60459	0.56770	0.64863	0.71066	0.72991
QICSO	**0.49594**	**0.55574**	**0.65983**	**0.70110**	**0.70550**

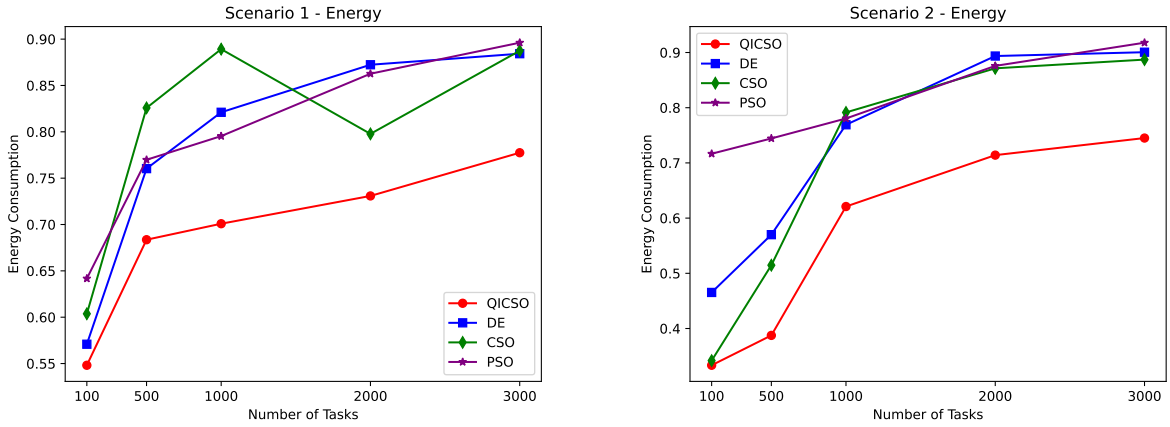
Table 15: Comparison in terms of Fitness for scenario-2, experiment-2.



(a) (b)

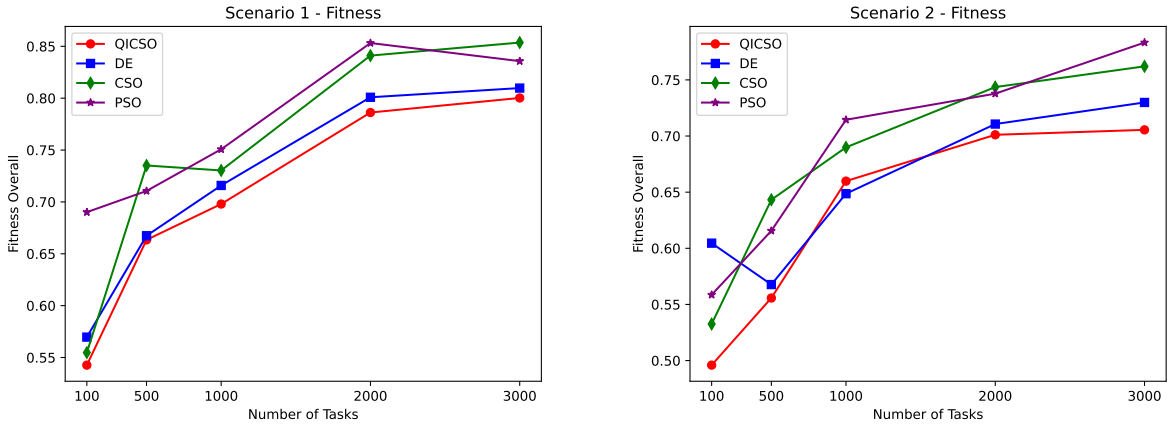
Figure 4: Comparison in terms of Total delay for experiment 2, (a)-Scenario 1 and (b)-Scenario 2

across all datasets. The consistent ANOVA outcomes across datasets further support this. The F-critical values, derived from the Table of Critical Values for the F-Distribution with degrees of freedom (DF = (3, 36)), remain consistent in all cases.



(a) (b)

Figure 5: Comparison in terms of Energy Consumption for experiment 2, (a)-Scenario 1 and (b)-Scenario 2



(a) (b)

Figure 6: Comparison in terms of Overall Fitness for experiment 2, (a)-Scenario 1 and (b)-Scenario 2

Groups	Count	Sum	Average	Variance
PSO Maratha et al. (2021)	10	7.4469	0.7446	0.0697
DE Bandyopadhyay et al. (2024)	10	8.0489	0.8048	0.0014
CSO Bakshi et al. (2023)	10	7.9799	0.7979	0.0012
QICSO	10	7.2055	0.7205	0.0005

Table 16: Input for ANOVA Test Datasets

6.4. Taguchi Experimental Method

The Taguchi Method is a statistical approach designed to improve the quality of products and processes by reducing variation and optimizing performance. It is based on robust design principles and is commonly used in manufacturing, engineering, and quality control. The

Source of Variance	Degree of Freedom	Sum of Squares	Mean Squares	F Statistical	P-value	F Critical
Between the groups	3	0.0505	0.0168	0.9233	0.4393	2.8662
Within the groups	36	0.6564	0.0182			
Total	39	0.7069				

Table 17: Result of ANOVA Test Datasets

method consists of three main stages: Design of Experiments (DoE), Signal-to-Noise (S-N) Ratio Analysis, and Optimization. The Taguchi method is included to control the variables in a manner to improve the factors to improve the performance and metrics of the finalized product. Orthogonal arrays are used to methodically change the levels of the multiple parameters or variables.

The Taguchi experimental design leverages orthogonal arrays to organize the parameters that influence a process and the levels at which they vary in a systematic manner. Unlike traditional factorial designs, which necessitate evaluating every possible combination, the Taguchi approach simplifies the process by testing selected pairs of combinations. This method is employed to examine the impact of various control factors and their levels on predefined objectives under consideration, User Allocation Rate (Γ), Processor Allocation (Φ), Total Delay (T_{total}), Total Energy Consumption (E_{total}). The factor that includes is : Factor A: Number of processing units(P). The Taguchi method classifies quality characteristics into three categories: nominal-the-best (SN_N), smaller-the-better (SN_S), and larger-the-better (SN_L). The SN_S category is used for objectives requiring minimization, while the SN_L category applies to objectives that require maximization. In this study, SN_L is applied to delay, is calculated using equation (57).

$$SN_{\mathcal{L}} = -10log\left(\frac{1}{n_{Rep}}\sum_{r=1}^{n_{Rep}}\frac{1}{(\Phi)^2}\right) \tag{57}$$

In this analysis, 'n_{Rep}' represents the number of repetitions performed for a given experimental configuration. For simplicity, we assume 'n_{Rep}' to be 1. However, if 'n_{Rep}' exceeds 1, the average value across all repetitions will be used. The parametric analysis results for P are summarized in Table 18. The Contribution Ratio (CR) of each factor affecting P with respect to SN_L is calculated using Eq. (58) as shown in Table 20.

$$CR_u = \frac{(SN_{\mathcal{L}})_{max,u} - (SN_{\mathcal{L}})_{min,u}}{\sum_{u=1}^{q}\left[(SN_{\mathcal{L}})_{max,u} - (SN_{\mathcal{L}})_{min,u}\right]} \tag{58}$$

It can be observed from Table 19 that the optimal combinations of factors for the delay are P_1 as these configurations yield the highest Systematic Normalized Latency. Therefore,p_1 is identified as the most favorable factor combination for achieving the desired performance in delay as shown in Figure 7.

6.5. Friedman Test

The Friedman test is a non-parametric alternative to the repeated measures ANOVA, designed to detect variations between treatments across multiple trials. It works by ranking

Exp. Numbers	Tasks	Processing Unit	Delay	SN-ratio
01	100	10	0.615	4.222
02	100	20	0.682	3.324
03	500	10	0.815	1.776
04	500	20	0.910	0.819
05	1000	10	0.859	1.320
06	1000	20	0.843	1.483
07	2000	10	0.884	1.071
08	2000	20	0.911	0.809
09	3000	10	0.890	1.012
10	3000	20	0.901	0.905

Table 18: Corresponding SN-ratio for Delay

	Levels	P
Average	1	3.107
	2	1.207
	3	0.964
(Max-Min)		2.14

Table 19: Factorial effects for Delay

Factors	P
CR	0.50

Table 20: The main effect plots for SNs delay (Signal-to-noise: Larger the better)

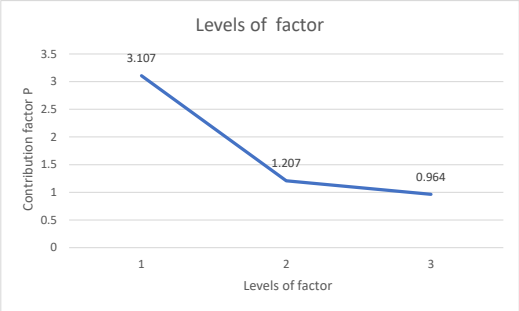


Figure 7: CR of each control factor for SN_L of Delay

the data within each row and then evaluating the ranks across the columns.

Like ANOVA, the Friedman test operates under two hypotheses: the null hypothesis (H_0) and the alternative hypothesis (H_1). The null hypothesis (H_0) is rejected if the Friedman test statistic (FM) exceeds the critical value of χ^2 at a significance level of 0.05. The hypothesis is defined as $H_{\mathcal{O}}$: δ(PSO(Maratha et al. (2021))) = δ(DE(Bandyopadhyay et al. (2024))) = δ(CSO(Bakshi et al. (2023))) = δ(QICSO), positing that , PSO Maratha et al. (2021), DE Bandyopadhyay et al. (2024), CSO Bakshi et al. (2023), and QICSO are equal. $H_{\mathcal{A}}$:

δ(PSO(Maratha et al. (2021))) $\neq$ δ(DE(Bandyopadhyay et al. (2024))) $\neq$ δ(CSO(Bakshi et al. (2023))) $\neq$ δ(QICSO), suggesting that the algorithms are not equal. In our analysis, we used 10 separate measurements for each algorithm. Based on the results, ranks were assigned, as presented in Table 21. The Friedman test statistic (FM) is then calculated using a specific formula based on these ranks.

$$\mathcal{FM} = \frac{12}{n_{subject} \times k_{treatment}(k_{treatment}+1)} \times \sum \mathcal{R}_i^2 - [3 \times n_{sample} \times (k_{treatment}+1)] \quad (59)$$

Where $n_{subjects}$ (=10) denotes the number of subjects, $k_{treatment}$ (=4) represents the number of treatments, and $\sum \mathcal{R}^2$ (=2769) signifies the squared sum of total ranks. From Eq.(59), the calculated $\mathcal{FM}$ is 16.14, which is greater than the χ^2 value at $\alpha = 0.005$ with $k_{treatment-1}$ (=3) degrees of freedom $\chi^2(0.005, 3) = 7.81$.

Input Sample						**Result of FM**			
SNo	**PSO**	**DE**	**CSO**	**QICSO**		**PSO**	**DE**	**CSO**	**QICSO**
1	0.867	0.842	0.834	0.691		4	3	2	1
2	0.782	0.846	0.744	0.708		3	4	2	1
3	0.878	0.798	0.800	0.746		4	1	3	1
4	0.800	0.874	0.829	0.686		2	4	3	1
5	0.809	0.771	0.826	0.718		3	2	4	1
6	0.835	0.747	0.823	0.721		4	2	3	1
7	0.784	0.793	0.772	0.741		3	4	2	1
8	0.822	0.789	0.793	0.705		4	2	3	1
9	0.806	0.793	0.819	0.752		3	2	4	1
10	0.873	0.796	0.740	0.736		4	3	2	1
Total : R_i						34	27	28	10
$(R_i)^2$						1156	729	784	100

Table 21: Computed Friedman Test

7. Conclusion and Future Work

The paper proposes a novel framework for Disaster Management user allocation for Unmanned Aerial Vehicles (UAVs) supported by Edge Computing (EC) using Quantum-Inspired Cuckoo Search Optimization (QICSO). This approach addresses critical challenges in disaster scenarios, including optimal user allocation, efficient processor utilization, minimizing energy consumption, and reducing data processing delays. Unmanned Aerial Vehicles (UAVs), when integrated with Edge Computing (EC), offer a promising solution for collecting and processing data efficiently in such high-stakes environments. The proposed fitness function is derived by considering the factors of allocation rates for users and processors, delay, and energy consumption. Comprehensive analysis and simulations of diverse scenarios with multiple experimental setups, compared against benchmark evolutionary algorithms such as Differential Evolution (DE) and Particle Swarm Optimization (PSO), demonstrate the efficiency and applicability of this meta-heuristic approach in catastrophic scenarios.

Imagine a disaster-affected region devastated by events such as earthquakes or floods, where roads, infrastructure, and network connectivity are heavily damaged, making it impossible for human rescue teams to immediately cover the affected area. IoT devices deployed in these regions play a critical role in capturing and transmitting data related to disasters, such as fire outbreaks, toxic gas leaks, or structural damages. Additionally, trapped individuals may generate task data using their mobile devices, further contributing to the information pool. However, these IoT devices have limited energy and processing capabilities, so offloading the tasks is necessary. The tasks should be offloaded to nearby ESs or UAVs to send humanitarian aid to the affected area. In contrast, ESs with their fixed locations can make it challenging to serve IoT devices trapped in unreachable areas. UAVs, with their mobility and ability to adjust their coverage, communicate with ESs to provide processing support in those areas. Together, UAVs and ESs process the offloaded data, assess the damage levels and share the information with rescue stations to initiate relief efforts for the affected individuals. The proposed framework with the QICSO algorithm ensures efficient decision-making and accelerates the delivery of humanitarian aid in disaster scenarios. However, there are specific operational challenges in deploying the proposed QICSO-based offloading approach. Several parameters of UAVs, ESs are mentioned. and QICSO-specific parameters, such as weight values, population size, and other algorithm settings must be appropriately configured before executing the algorithm. Future work will involve integrating machine learning and deep learning with quantum computing to address dynamic changes in disaster-affected areas. Further efforts will focus on developing a hybrid optimization technique by combining various approaches. Additional parameters, scenarios, and challenges will be considered, with an emphasis on deployment in real-life scenarios.

References

Bakshi, M., Chowdhury, C., and Maulik, U. (2023). Cuckoo search optimization-based energy efficient job scheduling approach for iot-edge environment. *The Journal of Supercomputing*, 79(16):18227–18255.

Balaji Naik, B., Singh, D., and Samaddar, A. B. (2020). Fhcs: Hybridised optimisation for virtual machine migration and task scheduling in cloud data center. *IET Communications*, 14(12):1942–1948.

Bandyopadhyay, B., Kuila, P., Govil, M. C., and Bey, M. (2024). Delay-sensitive task offloading and efficient resource allocation in intelligent edge–cloud environments: A discretized differential evolution-based approach. *Applied Soft Computing*, 159:111637.

Bey, M., Kuila, P., Naik, B. B., and Ghosh, S. (2024). Quantum-inspired particle swarm optimization for efficient iot service placement in edge computing systems. *Expert Systems with Applications*, 236:121270.

Dai, X., Xiao, Z., Jiang, H., and Lui, J. C. S. (2024). Uav-assisted task offloading in vehicular edge computing networks. *IEEE Transactions on Mobile Computing*, 23(4):2520–2534.

Ejaz, W., Ahmed, A., Mushtaq, A., and Ibnkahla, M. (2020). Energy-efficient task scheduling and physiological assessment in disaster management using uav-assisted networks. *Computer Communications*, 155:150–157.

Farahbakhsh, F., Shahidinejad, A., and Ghobaei-Arani, M. (2021). Multiuser context-aware computation offloading in mobile edge computing based on bayesian learning automata. *Transactions on Emerging Telecommunications Technologies*, 32(1):e4127.

Ghosh, S. and Kuila, P. (2023). Efficient offloading in disaster-affected areas using unmanned aerial vehicle-assisted mobile edge computing: A gravitational search algorithm-based approach. *International Journal of Disaster Risk Reduction*, 97:104067.

He, Q., Cui, G., Zhang, X., Chen, F., Deng, S., Jin, H., Li, Y., and Yang, Y. (2020). A game-theoretical approach for user allocation in edge computing environment. *IEEE Transactions on Parallel and Distributed Systems*, 31(3):515–529.

Hu, X., Wong, K.-K., Yang, K., and Zheng, Z. (2019). Task and bandwidth allocation for uav-assisted mobile edge computing with trajectory design. In *2019 IEEE Global Communications Conference (GLOBECOM)*, pages 1–6. IEEE.

Jiang, H., Dai, X., Xiao, Z., and Iyengar, A. (2023). Joint task offloading and resource allocation for energy-constrained mobile edge computing. *IEEE Transactions on Mobile Computing*, 22(7):4000–4015.

Li, M., Cheng, N., Gao, J., Wang, Y., Zhao, L., and Shen, X. (2020). Energy-efficient uav-assisted mobile edge computing: Resource allocation and trajectory optimization. *IEEE Transactions on Vehicular Technology*, 69(3):3424–3438.

Maghsoudi, A., Harpring, R., Piotrowicz, W. D., and Kedziora, D. (2023). Digital technologies for cash and voucher assistance in disasters: A cross-case analysis of benefits and risks. *International Journal of Disaster Risk Reduction*, 96:103827.

Maratha, P., Gupta, K., and Kuila, P. (2021). Energy balanced, delay aware multi-path routing using particle swarm optimisation in wireless sensor networks. *International Journal of Sensor Networks*, 35(1):10–22.

Ren, Y., Xie, Z., Ding, Z., Sun, X., Xia, J., and Tian, Y. (2021). Computation offloading game in multiple unmanned aerial vehicle-enabled mobile edge computing networks. *IET Communications*, 15(10):1392–1401.

Sadatdiynov, K., Cui, L., Zhang, L., Huang, J. Z., Salloum, S., and Mahmud, M. S. (2023). A review of optimization methods for computation offloading in edge computing networks. *Digital Communications and Networks*, 9(2):450–461.

Shahmoradi, J., Talebi, E., Roghanchi, P., and Hassanalian, M. (2020). A comprehensive review of applications of drone technology in the mining industry. *Drones*, 4(3).

Wang, S., Zhao, Y., Xu, J., Yuan, J., and Hsu, C.-H. (2019a). Edge server placement in mobile edge computing. *Journal of Parallel and Distributed Computing*, 127:160–168.

Wang, Y., Ru, Z.-Y., Wang, K., and Huang, P.-Q. (2019b). Joint deployment and task scheduling optimization for large-scale mobile users in multi-uav-enabled mobile edge computing. *IEEE transactions on cybernetics*, 50(9):3984–3997.

Wu, C., Peng, Q., Xia, Y., Ma, Y., Zheng, W., Xie, H., Pang, S., Li, F., Fu, X., Li, X., and Liu, W. (2021). Online user allocation in mobile edge computing environments:a decentralized reactive approach. *Journal of Systems Architecture*, 113:101904.

Ye, W., Luo, J., Shan, F., Wu, W., and Yang, M. (2020). Offspeeding: Optimal energy-efficient flight speed scheduling for uav-assisted edge computing. *Computer Networks*, 183:107577.

Yu, H., Leng, S., and Wu, F. (2024). Joint cooperative computation offloading and trajectory optimization in heterogeneous uav-swarm-enabled aerial edge computing networks. *IEEE Internet of Things Journal*, 11(10):17700–17711.

Zhang, P., Su, Y., Li, B., Liu, L., Wang, C., Zhang, W., and Tan, L. (2023). Deep reinforcement learning based computation offloading in uav-assisted edge computing. *Drones*, 7(3):213.

Zhang, T., Xu, Y., Loo, J., Yang, D., and Xiao, L. (2019). Joint computation and communication design for uav-assisted mobile edge computing in iot. *IEEE Transactions on Industrial Informatics*, 16(8):5505–5516.