Journal of Information Systems Engineering and Management
2025, 10(41s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

An Energy Efficient Meta-Heuristic Task Scheduling and Joint Uplink and Downlink Optimization Framework for Real-Time Mobile Edge Cloud Data

B.Teja Sree[1], G.P.S.Varma[2], Indukuri Hemalatha[3]
[1]*Research Scholar, Department of Computer Science and Engineering, Koneru Lakshmaiah Education Foundation, Vaddeswaram, India.*
[2]*Professor, Department of Computer Science and Engineering, Koneru Lakshmaiah Education Foundation, Vaddeswaram, India.*
[3]*Professor, Department of Information Technology, S.R.K.R Engineering College, Bhimavaram, India.*

ARTICLE INFO

Received: 18 Dec 2024

Revised: 10 Feb 2025

Accepted: 28 Feb 2025

ABSTRACT

Mobile edge cloud(MEC) plays a vital role in medium to large scale applications for task scheduling process. Join task uplink and downlink computations are used to optimize the cost and time computation for large scale applications. Most of the traditional mobile edge cloud based task scheduling models are independent of uplink and downlink estimations. The primary objective is to enhance energy efficiency and improve the user experience by maximizing the number of offloaded tasks during uplink communication, while ensuring that the computation resources of MEC remain at an acceptable level. In this work, an efficient joint load task scheduling approach is designed in order to improve the time, energy and load balancing properties in large scale applications. Experimental results show that the proposed approach has better efficiency in terms of runtime and energy consumption, leading to the improved energy efficiency in mobile edge cloud environments.

Keywords: Mobile, Edge , Meta heuristic model, scheduling, energy, time computation.

1. INTRODUCTION

Cloud computing is a popular system that offers on-demand services to both individuals and businesses. It has shifted how businesses view and access computing services.A cloud consists of a network of computers or servers that supply resources such as storage, networks, platforms, and software via the internet[1]. These resources are accessible by devices like smartphones, tablets, laptops, and desktops.Cloud computing is recognized as a reliable platform and business model. It provides IT infrastructure, platforms, and applications as web services with a pay-per-use system. Consumers use what they need and pay accordingly, while third-party providers handle resource management. Key technologies that support this system include virtualization, grid computing, service-oriented architecture, and metering tools. These elements work together to develop responsive and efficient systems.Mobile cloud computing integrates cloud services into wireless networks to improve mobile device performance[2]. It enhances mobile applications and functionality through service models like IaaS, and SaaS. Mobile cloud computing shifts data storage and processing away from mobile devices to external cloud infrastructure. This approach extends mobile computing beyond smartphones to a wider set of users with portable devices.A mobile device acts as a compact computing tool, capable of executing tasks typically handled by a traditional computer. Wireless communication between the mobile and cloud depends on the network layer[3]. Its performance, especially bandwidth availability, shapes the success of mobile cloud systems.The cloud solves key limitations of mobile devices by offering remote processing power, storage, and built-in security. Cloud data centers manage both the mobile and cloud environments, handling core computing needs off-device.Mobile cloud computing helps mitigate typical hardware constraints in mobile devices. These include limited storage, battery drain, weak sensing abilities, and slow processing[4]. It provides external support while managing energy use, service quality, mobility, and security concerns.To improve performance, MCC shifts heavy processing and large data operations to cloud-based nodes. This enables better service continuity and efficiency than standard mobile setups.Mobile cloud computing merges mobile and cloud computing to enhance device performance through task offloading[5]. It transfers demanding tasks like natural language processing, image editing, and object recognition to the cloud.To maintain task order and reduce load on devices, the system schedules tasks across both cloud servers and mobile units, considering which site can handle each task more effectively.Despite cloud assistance, mobile cloud systems still face hurdles. Mobile devices have limits in processing and storage, while communication networks can slow down task completion[6].

2. RELATED WORK

Mobile cloudlets act as functional extensions of cloud systems within mobile environments. They offer benefits like improved scalability and localized processing without full cloud infrastructure.A model using Markov decision processes enables dynamic offloading, where a mobile user's device chooses when to shift tasks based on energy and time tradeoffs.Different offloading strategies—such as always offloading, never offloading, or conditionally offloading—are chosen based on timing and energy constraints. The goal is to balance speed with battery savings.Experiments with Android devices have tested back-clone synchronization and nearby offloading. Communication across clients, service providers, and between provider networks addresses the unique patterns in mobile cloud environments[7-9].Three operational modes help make offloading more practical. These include methods to cut down end-to-end delay, with proactive routing protocols forming a part of Adaptive Mobile Cloud Computing (AMCC).Mobile cloudlets supported by volunteer computing and location-based services improve flexibility. A decentralized network design, using a virtual backbone in ad-hoc networks, expands task support across user devices without relying solely on centralized cloud systems.A structured architecture for mobile cloud computing includes multiple layers, each requiring specific security practices. This structure strengthens data handling and resource sharing in mobile environments.A focused strategy addresses latency-sensitive and resource-heavy tasks by optimizing resource allocation within cloudlets. These structures also include methods to secure communication and data handling within the cloud.A transaction-based interface (TI) is used to manage resource exchanges between handheld devices (buyers) and nearby cloudlets (sellers). An improved version of this interface benefits sellers more, especially when offloading occurs locally[10].Using mobile cloud computing, offloading to close-range cloudlets helps reduce energy use and decreases task response time. This makes nearby offloading both energy-efficient and faster for handheld devices.Battery life for handheld devices can be extended by merging mobile cloud computing with mobile power transfer (MPT). Using optimization models, energy usage is balanced while handling data transfers. When future channel data is known, adaptive methods adjust offloading to maintain system performance and can be expanded to support full-duplex and multi-transfer setups.Cloud-based video crowd sensing is improved by tackling three main tasks. First, an optimal video transcoding solution increases playback quality on mobile devices. Second, throughput for different file protocols is evaluated, leading to a protocol-aware, real-time transfer method[11]. Third, sensor-tagged videos are used to enhance cloud database interaction and video search capabilities.

Offloading Types in Heterogeneous Mobile Cloud Systems

Offloading decisions in heterogeneous mobile cloud setups depend on the strength of the remote connection. When low bandwidth is detected, systems aim to redistribute workloads to maintain smooth mobile operation.In areas with heavy mobile user presence, service providers can deploy local cloudlets to improve access. These nearby cloud nodes help users maintain service continuity, especially when cellular network strength drops[12].Users connected via 3G or 4G can automatically switch to nearby cloudlets when bandwidth becomes limited. Continued service is supported over Wi-Fi, with local processing handling tasks previously managed remotely.Different models for task offloading within cloudlets rely on node availability and how connections are established. Key challenges include managing setup processes and determining the optimal level for offloading.A mobile edge computing (MEC) setup is outlined with smart services and a nearby server managed using FDMA. Though actual servers may have low compute power, the model assumes access to high-capacity computing for analysis[13].

Edge-Based Execution for Real-Time Mobile Cloud Systems

In decentralized mobile-peripheral computational infrastructures, orchestrating workload allocation is fundamental for overseeing multifunctional operations and evenly distributing computational intensity across all accessible processing strata. This process is essential in delivering low-latency outcomes while optimizing energy dynamics between edge-linked handheld units and high-capacity cloud cores.Historically rooted in monolithic architectures, resource arbitration has evolved to address hybridized systems—where edge layers demand synchronization of physical nodes (CPUs, memory) and logical elements (data links, channel availability)[14]. These components must be synchronized under joint bidirectional (uplink-downlink) efficiency criteria.Each unit, mobile or cloud-based, relies on its kernel-level resource scheduler to proportionally distribute available compute cycles, tailored to runtime priorities and device constraints, especially under limited power budgets.High-density cloud-edge topologies must process diverse user requests, often concurrently. By leveraging smart task-routing algorithms, systems determine optimal task execution locations, dynamically adapting based on workload intensity and energy-saving thresholds.Responsive, algorithm-driven orchestration mechanisms now guarantee that task execution timelines are synchronized with energy-awareness goals—adapting to real-time status of bandwidth, compute load, and device health—thereby enabling intelligent task offloading decisions at runtime.When a cloudlet is active but under-resourced, it can distribute its load to nearby cloudlets using inter-cloudlet communication. This cooperation improves performance without involving the central cloud.Mobile devices discover cloudlets through a root server that keeps a live directory of cloudlets and their resources. With connectivity via Wi-Fi, 3G, or 4G, mobile clients locate the nearest available cloudlet, offload tasks, and receive results directly.If one cloudlet can't finish a task, it coordinates with others in the same area—like malls, offices, or public spaces—to complete the workload. This dynamic sharing model enhances system reliability, reduces response time, and optimizes energy usage in mobile edge computing environments[15].In the extended model of decentralized mobile computation, a novel architecture integrates API-based invocation with a remote procedure call (RPC) framework, enabling users to programmatically access and orchestrate distributed services. This integration, aligned with the Mobile Cloudlet Framework, serves as an edge layer between user devices and the cloud core. By embedding computational elasticity, the cloudlet amplifies mobility performance while preserving scalability metrics.In the extended model of decentralized mobile computation, a novel architecture integrates API-based invocation with a remote procedure call (RPC) framework, enabling users to programmatically access and orchestrate distributed services. This integration, aligned with the Mobile Cloudlet Framework, serves as an edge layer between user devices and the cloud core. By embedding computational elasticity, the cloudlet amplifies mobility performance while preserving scalability metrics[16-19].A dynamic task migration algorithm, defined via a Markov Decision Process (MDP), is formulated to optimize execution policies in fluctuating mobile environments. Let the state set be $\Omega = \{\omega_1, \omega_2, \ldots, \omega_n\}$, where each ω_i represents device-load conditions. The decision action set $A = \{a_{local}, a_{cloudlet}\}$ governs local vs. remote execution, with transition probabilities $P(\omega_j|\omega_i, a)$. The reward function $R(\omega_i, a)$ is calibrated for minimizing time-energy trade-offs.

From an optimization standpoint, define a binary division of execution outcomes:

- Case 1: Never Offload
- Case 2: Always Offload
  These are conditional on the critical threshold T_{crit}, where

$$\text{Offload}_{\text{decision}} = \begin{cases} \text{No}, & \text{if } T_{local} < T_{crit} \\ \text{Yes}, & \text{if } T_{cloudlet} < T_{crit} \end{cases}$$

Feasibility in real-world applications was tested using Android-based mobile prototypes, where clone-synchronization mechanisms were established. Let C_i be the clone of a task instance on device D_i. The consistency across clones is preserved via:

$$\delta(C_i, C_j) \to \text{min}\text{--}(1)$$

ensuring synchronized state replication during migration.

Offloading strategies were refined further by introducing location-based proximity functions. Given a set of cloudlets $\mathcal{L} = \{L_1, L_2, \ldots, L_m\}$, the optimal offload point L^* is:

$$L^* = \arg\min_{L_i \in \mathcal{L}} [\text{Latency}(U, L_i) + \text{Load}(L_i)]\text{--}(2)$$

Lastly, [20] address the multilayered communication model, incorporating intra-cloud and inter-cloud service exchanges. Define:

- $\phi_{\text{client-CSP}}$ – client to CSP interaction latency

- $\phi_{\text{intra-CSP}}$ – communication within the CSP
- $\phi_{\text{inter-CSP}}$ – cross-provider handoff delay

The cumulative delay function becomes:

$\Phi = \phi_{\text{client-CSP}} + \phi_{\text{intra-CSP}} + \phi_{\text{inter-CSP}}$--(3)

where the objective is minimizing Φ across distributed application execution paths.

In advancing mobile cloud computing paradigms, researchers have identified three unique offloading schemas that significantly enhance task feasibility and user responsiveness. These models are tailored to dynamically respond to execution contexts while maintaining system efficiency.

The first model emphasizes latency reduction through proactive networking protocols. Let D_{e2e} be the end-to-end delay, decomposed as:

$D_{\text{e2e}} = D_{\text{send}} + D_{\text{proc}} + D_{\text{return}}$--(4)

Here, AMCC (Adaptive Mobile Cloud Communication) techniques aim to minimize D_{e2e} using preemptive routing and lightweight session handshakes.

The second strategy involves volunteer-based edge computing, where mobile cloudlets operate in a geo-localized manner. A virtual ad hoc mesh is formed:

$$\mathcal{G}(V, E), V = \{\text{devices}\}, E = \{\text{communication links}\}$$

This mesh acts as a surrogate discovery backbone, allowing tasks T_i to offload onto optimal nodes $v_j \in V$ based on minimal processing time and local resource availability.

The third contribution to offloading feasibility comes from a layered architectural model within MCC. Let the system be structured as:

$$\text{Layers } \mathcal{L} = \{L_1, L_2, L_3\}, \text{where each } L_i \text{ applies security vector } \sigma_i$$

Each layer enforces context-sensitive security protocols to protect against data breaches and ensure integrity across distributed tasks, as surveyed[21].

Addressing resource bottlenecks in cloudlet operations, Liu et al. introduced a resource-aware allocation framework. Define:

- $\mathcal{R} = \{r_1, r_2, \dots, r_n\}$ = available resources
- $\mathcal{U} = \{u_1, u_2, \dots, u_m\}$ = user demands
- $A = [a_{ij}]$, where $a_{ij} = 1$ if resource r_i is allocated to user u_j

The optimization goal becomes:

$$\max \textstyle\sum_{i,j} \left(\frac{Q_{ij}}{L_{ij}} \cdot a_{ij}\right)\text{--(5)}$$

Where Q_{ij} is the quality of service and L_{ij} is latency for each assignment. Their algorithm adapts well under varying user densities, outperforming greedy-based allocations in large-scale networks.

As user traffic $\lambda(t)$ increases, their simulations show:

$$\text{QoS}(t) \propto \lambda(t), \text{where } \lambda(t) \text{ is user arrival rate}$$

indicating a scalable solution.

Given the complexity of managing heterogeneous services, the authors suggest adopting an ad hoc cloud topology:

$$\mathcal{A}_{\text{cloud}} = \mathcal{A}_{\text{static}} \cup \mathcal{A}_{\text{mobile}}\text{--(6)}$$

The objective function for such a setup is:

$$\min \mathcal{C} = \text{Infrastructure Cost}, \max \mathcal{E} = \frac{\text{Utilized Tasks}}{\text{Resources Allocated}}$$

This model allows adaptive provisioning of services while maintaining low-cost scalability.

In evolving Mobile Edge Computing (MEC) systems, [22] developed a simplified architecture that supports diverse smart applications by integrating a MES (Mobile Edge Server) infrastructure. This server, while idealized with infinite computing power, operates under Frequency-Division Multiple Access (FDMA) communication schemes.

The key performance metric is average energy consumption across both mobile users and the edge server:

$$\bar{E} = \frac{1}{T}\sum_{t=1}^{T} \left(E_{\text{mobile}}(t) + E_{\text{edge}}(t)\right)\text{--(7)}$$

To optimize this system, a joint resource control model is applied:

- $f_{\text{mobile}}, f_{\text{edge}}$: CPU cycle frequency allocations
- P_{tx}: transmission power
- B: communication bandwidth

The optimization seeks to minimize:

$$\min_{f, P_{\text{tx}}, B} \bar{E} \text{ subject to latency and task constraints}$$

[23] introduced an adaptive resource allocation mechanism, incorporating mobility forecasting and dynamic resource requirements $R(t)$. Given:

- Predicted trajectory $\tau(t)$
- Required service time S_t
  The allocation decision $A(t)$ is made based on:

$$A(t) = f(\tau(t), R(t), S_t)\text{--(8)}$$

To further enhance efficiency[24] explored energy harvesting edge-enabled systems, where tasks are processed either locally or offloaded based on energy and delay costs. Each time slot t includes:

- $E_b(t)$: battery energy level
- $D_{\text{exec}}(t)$: execution delay
- $C_{\text{offload}}(t)$: cost of offloading

They demonstrate that optimal CPU frequencies f_t increase with energy:

$$\frac{df_t}{dE_b} \geq 0$$

This implies higher available energy accelerates execution and reduces latency.

In spatially dispersed computational frameworks, compute units $\Pi = \{\pi_1, \pi_2, \ldots, \pi_n\}$ are co-embedded with terminal agents Θ and sensory interfaces Ψ, enabling minimal-latency processing through proximity-optimized computation. The temporal response function τ_r is minimized as:

$$\min_{\pi_i} \tau_r = f(d(\pi_i, \Theta_j), \gamma_{\text{proc}}, \mu_{\text{mem}})\text{--(9)}$$

where d denotes spatial distance, γ_{proc} is processing frequency, and μ_{mem} is available memory per peripheral node.

In this architectural regime, **energy-constrained scheduling** of high-complexity tasks Ξ is optimized by executing near the origination point:

$$\Xi_k = \arg\min(\varepsilon_k \cdot \tau_k)$$

where ε_k is energy cost and τ_k is task latency.

Security primitives are applied directly at the edge node π_i, using embedded cryptographic gates κ and access verifiers ρ:

$$\mathcal{S}(\pi_i) = \kappa(D) \oplus \rho(\text{UID})$$

ensuring integrity $\iota(D)$ across the transformation sequence without central exposure:

$$\forall t, \iota(D_t) = \iota(D_0)$$

Each node executes lightweight inference models Λ_i, where:

$$\Lambda_i: \Phi \to \{0,1\}, \text{detecting anomalies } \alpha_t \in \Phi$$

This real-time inference enables early-stage threat mitigation at the edge, precluding compromise before central processing.Horizontal expansion is achieved via parallel delegation:

$$\Omega = \bigcup_{i=1}^{n} \Xi_i$$

where each task set Ξ_i is routed to node π_i, thus reducing global cloud pressure and improving scalability σ, measured as:

$$\sigma = \frac{\sum_{i=1}^{n} T_{\text{handled}}(\pi_i)}{T_{\text{total}}} \text{--(10)}$$

In a **modular network** structure, routing decisions ρ_t and compute path χ_t are dynamically recalculated:

$$\rho_t = f(\text{RT}_t, \nu_t), \chi_t = f(\text{QoS}_t)$$

where ν_t represents current load, and RT_t the routing table at time t. This adaptiveness supports optimized queueing $Q(t)$ across uplink/downlink paths:

$$Q(t) = \min\left(\frac{W_i(t)}{\omega_i(t)}\right), i \in \Pi\text{--(11)}$$

with $W_i(t)$ as queue weight and $\omega_i(t)$ as service rate.

3.PROPOSED MODEL

Mobile units are prepared for sending and processing tasks, checking energy usage. Edge and cloud systems are also initialized, registering their speed, storage, and connection limits.This layer handles the main logic for choosing where to process tasks and how to divide resources. It looks at size, energy use, and delay to decide if tasks stay local or go to cloud or edge servers. It models how long tasks take, how much energy they use, and builds an algorithm to balance both.
Simulates how long and how much energy tasks need if sent to mobile edge computing nodes.Spreads bandwidth and computing power across devices. Uses delay formulas to calculate time for each task and tracks how much energy is used.
Estimates how long a task takes with the formula:

$$T = \frac{D}{R}$$

where D is data size and R is the resource rate.

Estimates energy use by linking task size with device power usage.A scheduling algorithm that adjusts decisions based on both time and energy needs, choosing centralized or shared control.Takes over when the cloud handles tasks. It returns results and manages delays caused by shared access or waiting time.Predicts slowdowns or bottlenecks in cloud systems due to many tasks sharing resources.Records what happens during processing. Tracks how well IPSO and CSO algorithms perform and saves metrics for review. The overview of proposed model is shown in figure 1.

User Interface Layer

This layer initializes system components.

- **Mobile Devices Setup**
  Configuration of mobile terminals for task generation, offloading readiness, and energy profiling.
- **Edge, Cloud Devices Setup**
  Initializes edge servers and cloud datacenters, registering compute capabilities, bandwidth, and latency thresholds.

2. Joint Optimization Layer

This is the **core layer** that balances **task offloading** and **resource scheduling**.

- **Joint Task Offloading**
  Decides which tasks should execute locally or be offloaded to MEC/cloud based on energy, delay, or task size.

  ▸ Leads to:

  - **MEC Execution Modelling**: Simulates execution time, delay, and energy if offloaded to MEC.
- **Joint Resource Allocation**
  Allocates bandwidth, CPU cycles, and memory across nodes.

  ▸ Leads to:

  - **Time Consumption Modelling**: Computes delay for task execution using:$T = \frac{D}{R}$
  - **Energy Consumption Modelling**: Calculates energy consumed based on task size and power profile.
- ▸ Outputs:
  - **Proposed CSO Model**: Centralized or Cooperative Scheduling Optimization algorithm that integrates time-energy trade-offs.

3. Cloud Services Layer

Handles execution and load balancing when cloud is involved.

- **Task Execution Result**
  Delivers final output from cloud after computation.
- **Queueing Constraints for Load Balancing**
  Models waiting time and resource contention in cloud environments.

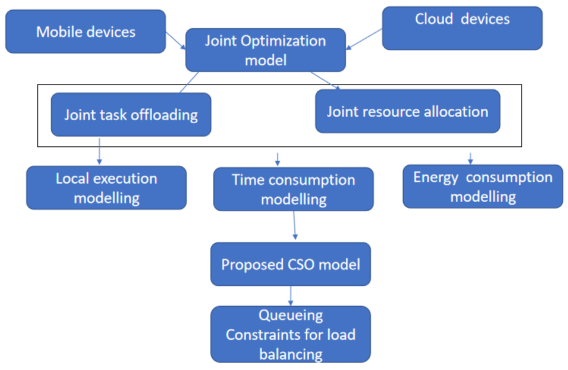

Mobile devices
Joint Optimization model
Cloud devices
Joint task offloading
Joint resource allocation
Local execution modelling
Time consumption modelling
Energy consumption modelling
Proposed CSO model
Queueing Constraints for load balancing

Figure 1: Overview of Proposed Model

Edge Execution Cost Breakdown
The total edge execution cost Q_n^e is defined as a combination of time and energy factors:

$$Q_n^e = \alpha T_n^e + \beta E_n^e$$

Here, α and β are weights for time and energy respectively.

Edge Task Timing Details
The total time to send a task from the mobile device MD_n to the edge node is:

$$T_n^e = T_n^{tr} + T_n^r + T_{n,e}^{in,p} + T_{n,e}^q$$

Transmission Delay T_n^{tr}
This is the time to transmit the task data from MD_n to the edge node:

$$T_n^{tr} = \frac{DS_{n,e}^{in}}{b_n}$$

where $DS_{n,e}^{in}$ is the task size and b_n is the bandwidth.

Response Time T_n^r
This captures the time to return the results from the edge node to MD_n:

$$T_n^r = \frac{DS_{n,e}^{out}}{b_n}$$

where $DS_{n,e}^{out}$ is the size of the result.

Execution Time $T_{n,e}^{in,p}$
This is how long it takes the edge processor to complete the task:

$$T_{n,e}^{in,p} = \frac{S_n}{f^e}$$

with S_n as the task's required computation and f^e the edge node's processing speed.

Queue Waiting Time $T_{n,e}^q$
Time the task spends waiting in the edge node's processing queue before execution begins.

Solution Setup
In this method, each egg inside a nest stands for a possible answer.
Each cuckoo creates just one answer.
Weak answers are replaced by better ones coming from other cuckoos.

Nest Placement
Nest locations are written as x_i^t, where i goes from 1 to Z (number of nests or problem size), and t is the iteration count.

Search Pattern with Lévy Flight
Lévy flight helps guide both wide and narrow searches for better answers.
New nest locations are calculated using:

$$x_{\text{new}}^{(t)} = \text{round}\left(\frac{1}{1 + e^{-\beta x_{\text{nest}}^{(t)}}}\right)$$

Generating a New Solution
A new position for a nest is computed as:

$$x_i^{(t+1)} = x_i^{(t)} + \alpha \cdot L(s, \lambda)$$

Where:

- $x_i^{(t+1)}$ is the updated position of nest i at the next step.
- $x_i^{(t)}$ is the current position.
- α adjusts the step length.
- s is the step size.
- $L(s, \lambda)$ defines the Lévy flight behavior.

Global Random Walk Update Rule

$$x_i^{t+1} = x_i' + \alpha s \otimes H(P_a - \varepsilon) \otimes (x_i' - x_k')$$

Where:

- x_i^{t+1} is the updated solution.
- s is the step size.
- H is the Heaviside function.
- ε is a random number.
- x_k' is a randomly selected solution.
- P_a is a probability that controls switching.

Edge Queue Modeling

Queue Setup
Modeled as M/M/1 queue since only one edge server is used.

- Arrival rate: λ^e
- Service rate: μ^e

Edge Server Utilization

$$\rho^e = \frac{\lambda^e}{\mu^e}$$

Measures how busy the server is.

Average Waiting Load

$$L^e = \frac{(\rho^e)^2}{1-\rho^e}$$

Gives the expected number of tasks waiting in line.

Task Waiting Time in Queue

$$T_{n,q}^e = \frac{L^e}{\lambda^e}$$

Calculated using both the queue length and task arrival rate.

Overall Execution Time at Edge

$$T_n^e = T_n^{tr} + T_n^r + T_{n,e}^{in,p} + T_{n,q}^e$$

Adds up all delays: sending, result return, processing, and queue wait.

Edge Transmission Resource Calculations

Define:

- D: Volume of data in transmission
- B: Channel bandwidth
- H: Channel gain (uplink/downlink)
- P: Transmission power (uplink/downlink)
- N_0: Noise power

Uplink Rate

$$R_{\text{uplink}} = B \cdot \frac{\log_2\left(1+\frac{H_{\text{uplink}} \cdot P_{\text{uplink}}}{N_0}\right)}{\log_2 e} \text{--(13)}$$

Uplink Time

$$T_{\text{uplink}} = \frac{D}{R_{\text{uplink}}} \text{--(14)}$$

Downlink Rate

$$R_{\text{downlink}} = B \cdot \frac{\log_2\left(1+\frac{H_{\text{downlink}} \cdot P_{\text{downlink}}}{N_0}\right)}{\log_2 e} \text{--(15)}$$

Downlink Time

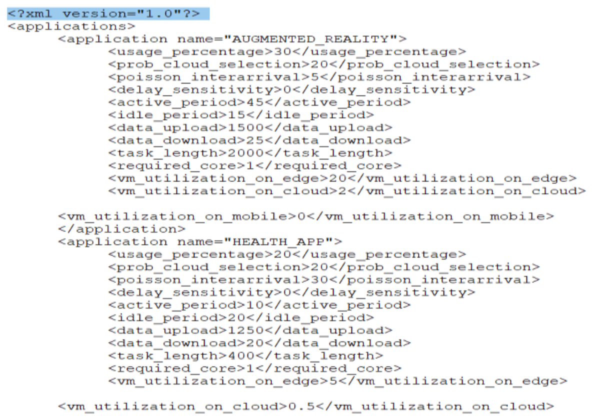
$$T_{\text{downlink}} = \frac{D}{R_{\text{downlink}}} \text{--(16)}$$

Each mobile device starts with defined settings: its task load, battery level, and whether it can offload tasks. At the same time, all edge and cloud nodes are set up with their compute strength, network capacity, and delay stats—split into upload, processing, and download times.This layer decides how to run each task—locally or offloaded—based on time and energy needs. Task time comes from dividing task size by processing speed. Energy use depends on CPU power per cycle and total cycles. Offloading is marked by a binary value. The system uses Cuckoo Search Optimization (CSO) to minimize a weighted mix of delay and energy use, adjusting based on whether users care more about speed or saving battery.Tasks that need more processing or can't handle wait times go to the cloud. The system checks the number of pending tasks and their average times to calculate delay. It also makes sure no cloud server gets overloaded by comparing all task sizes against its limits.

Journal of Information Systems Engineering and Management
2025, 10(41s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/
Research Article

4.EXPERIMENTAL RESULTS

Experimental results are simulated uisng java based mobile edge cloud simulation tool. This tool is customized usign proposed task schedluing and uplink and downlink computations. Figure 2 represents the input application configuration file which contains mobile ,edge and cloud configuration settings.



```
<?xml version="1.0"?>
<applications>
     <application name="AUGMENTED_REALITY">
          <usage_percentage>30</usage_percentage>
          <prob_cloud_selection>20</prob_cloud_selection>
          <poisson_interarrival>5</poisson_interarrival>
          <delay_sensitivity>0</delay_sensitivity>
          <active_period>45</active_period>
          <idle_period>15</idle_period>
          <data_upload>1500</data_upload>
          <data_download>25</data_download>
          <task_length>2000</task_length>
          <required_core>1</required_core>
          <vm_utilization_on_edge>20</vm_utilization_on_edge>
          <vm_utilization_on_cloud>2</vm_utilization_on_cloud>

     <vm_utilization_on_mobile>0</vm_utilization_on_mobile>
     </application>
     <application name="HEALTH_APP">
          <usage_percentage>20</usage_percentage>
          <prob_cloud_selection>20</prob_cloud_selection>
          <poisson_interarrival>30</poisson_interarrival>
          <delay_sensitivity>0</delay_sensitivity>
          <active_period>10</active_period>
          <idle_period>20</idle_period>
          <data_upload>1250</data_upload>
          <data_download>20</data_download>
          <task_length>400</task_length>
          <required_core>1</required_core>
          <vm_utilization_on_edge>5</vm_utilization_on_edge>

     <vm_utilization_on_cloud>0.5</vm_utilization_on_cloud>
```

Figure 2: Input application configuration file

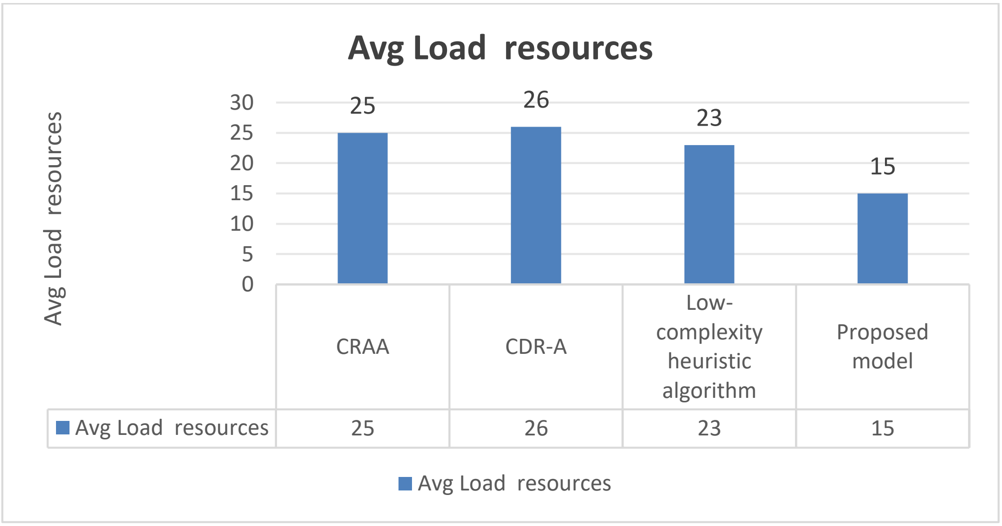


Figure 3: Comparative analysis of proposed model to existing models for average load resources.

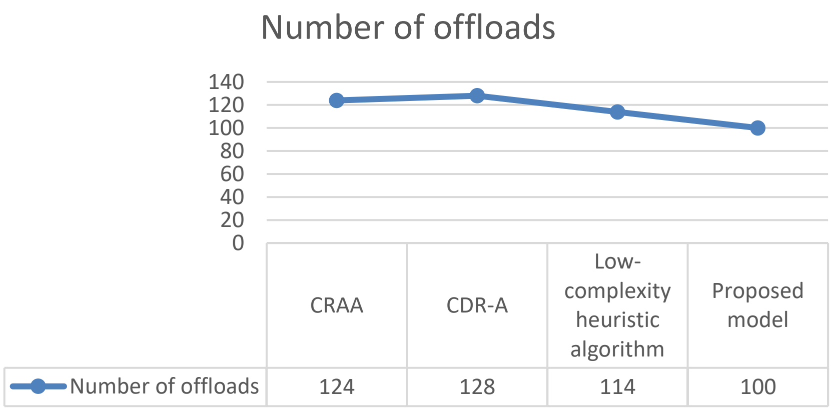


Figure 4: Comparative analysis of proposed model to existing models for number of offloads per tasks.

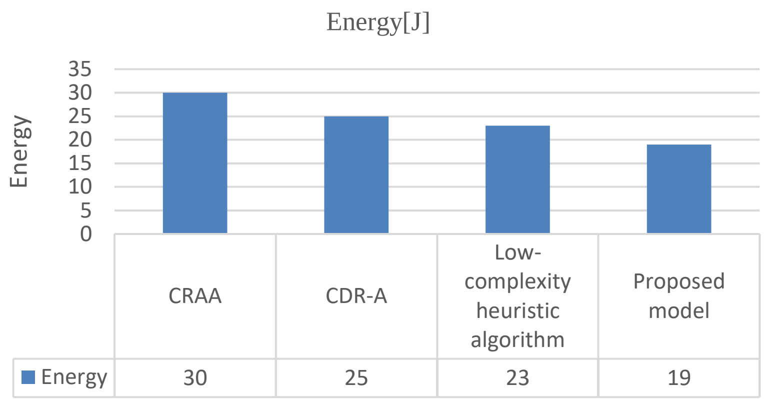


Figure 5: Comparative analysis of proposed model to existing models for energy computation.

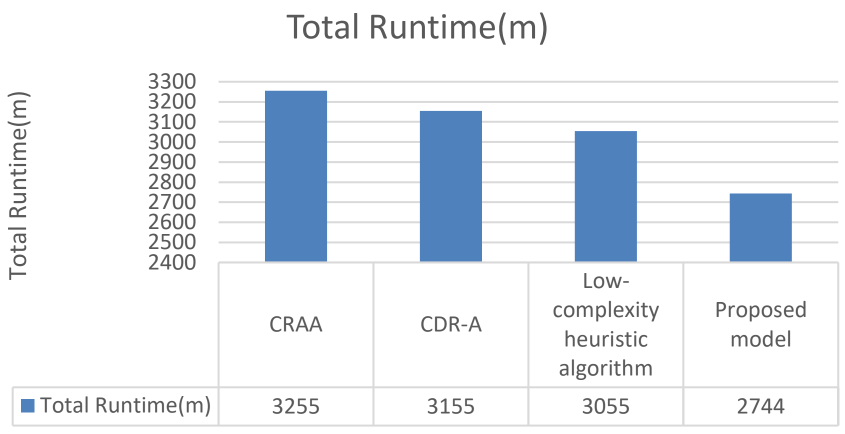


Figure 6: Comparative analysis of proposed model to existing models for total runtime(m).

The performance comparisons(Figure 3-6) across the four methods[25]—CRAA, CDR-A, Low-complexity heuristic algorithm, and the Proposed model—shows that the proposed approach consistently outperforms others in efficiency and resource management. In terms of average load on resources, the proposed model shows the lowest usage at 15, compared to 25–26 in CRAA and CDR-A, indicating more efficient task distribution. Regarding the number of offloads, although the proposed model performs fewer offloads (100), it likely achieves better optimization by offloading only when beneficial, unlike CRAA and CDR-A which offload more frequently (124 and 128, respectively). In energy consumption, the proposed model again leads with the lowest value (19J), suggesting it is more power-efficient than CRAA (30J), CDR-A (25J), and the heuristic algorithm (23J). Lastly, in terms of total runtime, the proposed model has the shortest completion time at 2744m, whereas CRAA takes the longest at 3255m. Altogether, the proposed model demonstrates a balanced and optimized performance across all metrics.

V.CONCLUSION

In this paper, a novel meta-heuristic optimization model with uplink and downlink computations are proposed for large scale task scheduling applications. Proposed optimization model with uplink and downlink metrics are evaluated on large mobile edge cloud configurations. This model is evaluated with different parameters and optimization functions. Experimental results are evaluated with different MEC performance metrics on large scale applications. In future, this work is extended to federated based mobile edge computation using parallel computation.

VI.REFERENCES

[1] C. Yi, J. Cai, and Z. Su, "A Multi-User Mobile Computation Offloading and Transmission Scheduling Mechanism for Delay-Sensitive Applications," IEEE Transactions on Mobile Computing, vol. 19, no. 1, pp. 29–43, Jan. 2020, doi: 10.1109/TMC.2019.2891736.

[2] H. Lim et al., "An Empirical Study of 5G: Effect of Edge on Transport Protocol and Application Performance," IEEE Transactions on Mobile Computing, vol. 23, no. 4, pp. 3172–3186, Apr. 2024, doi: 10.1109/TMC.2023.3274708.

[3] S.-H. Park, S. Jeong, J. Na, O. Simeone, and S. Shamai, "Collaborative Cloud and Edge Mobile Computing in C-RAN Systems With Minimal End-to-End Latency," IEEE Transactions on Signal and Information Processing over Networks, vol. 7, pp. 259–274, 2021, doi: 10.1109/TSIPN.2021.3070712.

[4] T. T. Nguyen, L. B. Le, and Q. Le-Trung, "Computation Offloading in MIMO Based Mobile Edge Computing Systems Under Perfect and Imperfect CSI Estimation," IEEE Transactions on Services Computing, vol. 14, no. 6, pp. 2011–2025, Nov. 2021, doi: 10.1109/TSC.2019.2892428.

[5] H. Yuan et al., "Cost-Efficient Task Offloading in Mobile Edge Computing With Layered Unmanned Aerial Vehicles," IEEE Internet of Things Journal, vol. 11, no. 19, pp. 30496–30509, Oct. 2024, doi: 10.1109/JIOT.2024.3408216.

[6] V. Kumar, M. Mukherjee, J. Lloret, Q. Zhang, and M. Kumari, "Delay-Optimal and Incentive-Aware Computation Offloading for Reconfigurable Intelligent Surface-Assisted Mobile Edge Computing," IEEE Networking Letters, vol. 4, no. 3, pp. 127–131, Sep. 2022, doi: 10.1109/LNET.2022.3187720.

[7] C. Zhang and H. Du, "DMORA: Decentralized Multi-SP Online Resource Allocation Scheme for Mobile Edge Computing," IEEE Transactions on Cloud Computing, vol. 10, no. 4, pp. 2497–2507, Oct. 2022, doi: 10.1109/TCC.2020.3044852.

[8] Q. Zhang, L. Gui, F. Hou, J. Chen, S. Zhu, and F. Tian, "Dynamic Task Offloading and Resource Allocation for Mobile-Edge Computing in Dense Cloud RAN," IEEE Internet of Things Journal, vol. 7, no. 4, pp. 3282–3299, Apr. 2020, doi: 10.1109/JIOT.2020.2967502.

[9] S. Mukherjee and J. Lee, "Edge Computing-Enabled Cell-Free Massive MIMO Systems," IEEE Transactions on Wireless Communications, vol. 19, no. 4, pp. 2884–2899, Apr. 2020, doi: 10.1109/TWC.2020.2968897.

[10] Z. Xu, J. Liu, J. Zou, and Z. Wen, "Energy-Efficient Design for IRS-Assisted NOMA-Based Mobile Edge Computing," IEEE Communications Letters, vol. 26, no. 7, pp. 1618–1622, Jul. 2022, doi: 10.1109/LCOMM.2022.3172309.

[11] Y. K. Tun, Y. M. Park, N. H. Tran, W. Saad, S. R. Pandey, and C. S. Hong, "Energy-Efficient Resource Management in UAV-Assisted Mobile Edge Computing," IEEE Communications Letters, vol. 25, no. 1, pp. 249–253, Jan. 2021, doi: 10.1109/LCOMM.2020.3026033.

[12] Y. Ye, R. Q. Hu, G. Lu, and L. Shi, "Enhance Latency-Constrained Computation in MEC Networks Using Uplink NOMA," IEEE Transactions on Communications, vol. 68, no. 4, pp. 2409–2425, Apr. 2020, doi: 10.1109/TCOMM.2020.2969666.

[13] X. Huang, S. Zeng, D. Li, P. Zhang, S. Yan, and X. Wang, "Fair Computation Efficiency Scheduling in NOMA-Aided Mobile Edge Computing," IEEE Wireless Communications Letters, vol. 9, no. 11, pp. 1812–1816, Nov. 2020, doi: 10.1109/LWC.2020.3001994.

[14] K. Guo, M. Yang, Y. Zhang, and J. Cao, "Joint Computation Offloading and Bandwidth Assignment in Cloud-Assisted Edge Computing," IEEE Transactions on Cloud Computing, vol. 10, no. 1, pp. 451–460, Jan. 2022, doi: 10.1109/TCC.2019.2950395.

[15] R. Xiong, C. Zhang, X. Yi, L. Li, and H. Zeng, "Joint Connection Modes, Uplink Paths and Computational Tasks Assignment for Unmanned Mining Vehicles' Energy Saving in Mobile Edge Computing Networks," IEEE Access, vol. 8, pp. 142076–142085, 2020, doi: 10.1109/ACCESS.2020.3013714.

[16] G. Interdonato and S. Buzzi, "Joint Optimization of Uplink Power and Computational Resources in Mobile Edge Computing-Enabled Cell-Free Massive MIMO," IEEE Transactions on Communications, vol. 72, no. 3, pp. 1804–1820, Mar. 2024, doi: 10.1109/TCOMM.2023.3336355.

[17] A. Al-Shuwaili and A. Lawey, "Latency Reduction for Mobile Edge Computing in HetNets by Uplink and Downlink Decoupled Access," IEEE Wireless Communications Letters, vol. 10, no. 10, pp. 2205–2209, Oct. 2021, doi: 10.1109/LWC.2021.3096897.

[18] L. Zhang and N. Ansari, "Latency-Aware IoT Service Provisioning in UAV-Aided Mobile-Edge Computing Networks," IEEE Internet of Things Journal, vol. 7, no. 10, pp. 10573–10580, Oct. 2020, doi: 10.1109/JIOT.2020.3005117.

[19] M. Ke, Z. Gao, Y. Wu, X. Gao, and K.-K. Wong, "Massive Access in Cell-Free Massive MIMO-Based Internet of Things: Cloud Computing and Edge Computing Paradigms," IEEE Journal on Selected Areas in Communications, vol. 39, no. 3, pp. 756–772, Mar. 2021, doi: 10.1109/JSAC.2020.3018807.

[20] L. Huang, L. Zhang, S. Yang, L. P. Qian, and Y. Wu, "Meta-Learning Based Dynamic Computation Task Offloading for Mobile Edge Computing Networks," IEEE Communications Letters, vol. 25, no. 5, pp. 1568–1572, May 2021, doi: 10.1109/LCOMM.2020.3048075.

[21] T. Zhou, Y. Yue, D. Qin, X. Nie, X. Li, and C. Li, "Mobile Device Association and Resource Allocation in HCNs With Mobile Edge Computing and Caching," IEEE Systems Journal, vol. 17, no. 1, pp. 976–987, Mar. 2023, doi: 10.1109/JSYST.2022.3157590.

[22] C. Park and J. Lee, "Mobile Edge Computing-Enabled Heterogeneous Networks," IEEE Transactions on Wireless Communications, vol. 20, no. 2, pp. 1038–1051, Feb. 2021, doi: 10.1109/TWC.2020.3030178.

[23] M. A. Hossain and N. Ansari, "Network Slicing for NOMA-Enabled Edge Computing," IEEE Transactions on Cloud Computing, vol. 11, no. 1, pp. 811–821, Jan. 2023, doi: 10.1109/TCC.2021.3117754.

[24] S. Huang, B. Lv, R. Wang, and K. Huang, "Scheduling for Mobile Edge Computing With Random User Arrivals—An Approximate MDP and Reinforcement Learning Approach," IEEE Transactions on Vehicular Technology, vol. 69, no. 7, pp. 7735–7750, Jul. 2020, doi: 10.1109/TVT.2020.2990482.

[25] B. Jedari, G. Premsankar, G. Illahi, M. D. Francesco, A. Mehrabi, and A. Ylä-Jääski, "Video Caching, Analytics, and Delivery at the Wireless Edge: A Survey and Future Directions," IEEE Communications Surveys & Tutorials, vol. 23, no. 1, pp. 431–471, 2021, doi: 10.1109/COMST.2020.3035427.