Results in Engineering 25 (2025) 104342

Contents lists available at ScienceDirect

Results in Engineering

journal homepage: www.sciencedirect.com/journal/results-in-engineering

Research paper

Dynamic priority-based task scheduling and adaptive resource allocation algorithms for efficient edge computing in healthcare systems

J. Anand, B. Karthikeyan *

School of Electronics Engineering, Vellore Institute of Technology, Vellore, 632014, Tamil Nadu, India

ARTICLE INFO

Keywords:
Mobile edge computing
Task scheduling
Dynamic priority
Adaptive resource allocation
Healthcare monitoring systems
IoT

ABSTRACT

The Internet of Things (IoT) has revolutionized healthcare by interconnecting a wide range of devices over the Internet. Cloud computing has traditionally fulfilled the computational demands of these IoT devices, enabling the collection and analysis of data from healthcare environments. However, its suitability for latency-sensitive applications has been limited due to the presence of remote cloud data centers. In response, edge computing has emerged, pushing computational capabilities to the edge of networks to mitigate these limitations. This work presents a novel approach for task scheduling and resource allocation in edge computing environments tailored for healthcare monitoring systems. By implementing dynamic priority-based task scheduling and adaptive resource allocation algorithms, our methodology enhances the handling of resource requests between end devices, edge nodes (ENs), and the cloud. These innovations focus on reducing latency, optimizing resource utilization, and improving overall system performance. Extensive simulations demonstrate the efficacy of our approach, showcasing significant improvements in task execution time, resource utilization, and energy consumption.

1. Introduction

The rapid advancement of Internet of Things (IoT) technologies has transformed the healthcare sector by enabling real-time monitoring and management of patient health through wearable devices and sensors. These innovations improve patient outcomes by facilitating continuous monitoring, early detection of medical conditions, and timely interventions. However, the growing adoption of IoT devices introduces challenges in processing and managing the massive amounts of data generated, particularly in meeting the real-time demands and priority requirements of healthcare tasks [1].

Traditional cloud computing architectures often struggle to deliver the low-latency and high-priority task handling required in healthcare scenarios, especially during critical situations such as emergency alerts. Mobile Edge Computing (MEC) addresses these limitations by processing data closer to the source, reducing latency and bandwidth usage while ensuring faster response times [2]. Fig. 1, below, depicts the conceptual overview of the Mobile Edge Computing (MEC) architecture. This architecture highlights the hierarchical relationship between edge servers, access points, and cloud data centers, showcasing how tasks can be processed closer to data sources, such as wearable devices or sensors. In healthcare applications, this distributed approach significantly reduces dependence on cloud data centers, supporting low-latency data processing and enabling timely interventions such as real-time patient monitoring or emergency alerts. Despite its advantages, MEC poses its own challenges, including limited resource availability, uneven resource distribution across edge nodes (ENs), and the need for adaptive task management in dynamic environments. These challenges become particularly significant in healthcare, where the urgency and priority of tasks can change rapidly based on patient conditions [3].

In edge computing environments for healthcare applications, ensuring real-time data processing with limited computational resources at edge nodes and addressing heterogeneous data formats from multimodal sensors remain significant hurdles. Additionally, maintaining accuracy and efficiency in analyzing dynamic physiological data adds another layer of complexity. To address these issues, systems are being developed that integrate flexible sensors with reservoir computing algorithms, which enable low-latency, cloud-independent monitoring. However, these systems still face challenges related to scalability and adaptability to diverse real-world conditions [4]. In hybrid edge-cloud computing environments, key challenges include high latency from distant cloud centers, scalability issues as IoT devices proliferate, resource bottlenecks at edge nodes, and the dynamic nature of healthcare data, which often involves heterogeneous formats and transmission rates. Ad-

* Corresponding author.
E-mail addresses: anand.j@vit.ac.in (J. Anand), bkarthikeyan@vit.ac.in (B. Karthikeyan).

https://doi.org/10.1016/j.rineng.2025.104342
Received 26 November 2024; Received in revised form 30 January 2025; Accepted 11 February 2025

Available online 13 February 2025

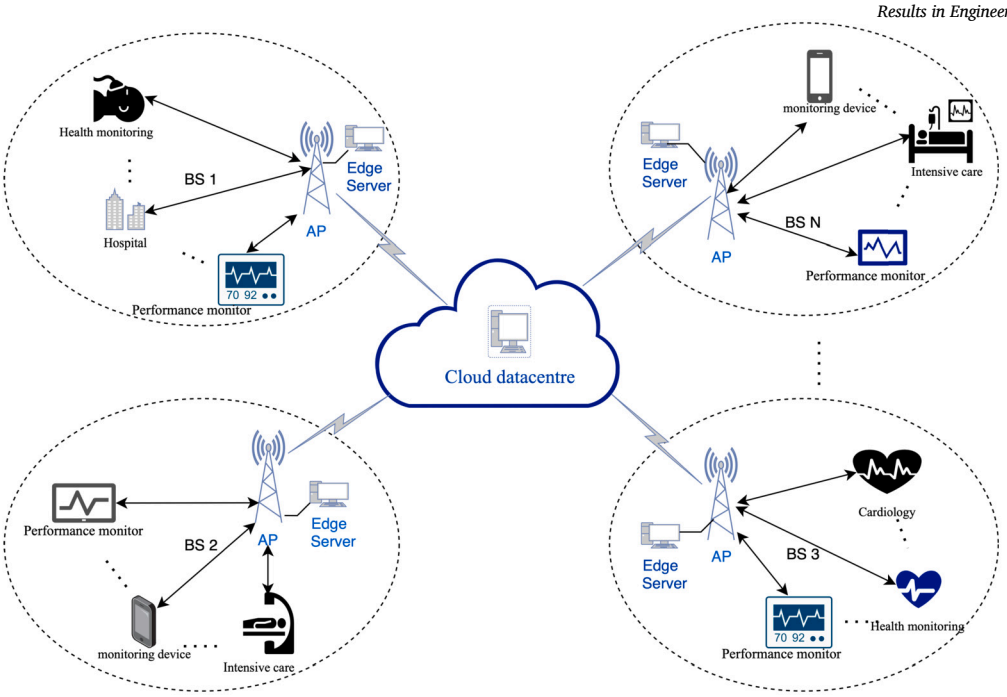

Fig. 1. Conceptual overview of MEC architecture.

dressing these challenges requires innovative strategies for task scheduling and resource allocation that are flexible, adaptive, and responsive to real-time demands.

This study proposes a Dynamic Priority-Based Task Scheduling and Adaptive Resource Allocation (DPTARA) mechanism to address these challenges. The key contributions of this work include:

1. Developing novel algorithms for dynamic priority adjustment and adaptive resource allocation in edge computing environments.
2. Minimizing latency by prioritizing healthcare tasks and allocating resources closer to end devices.
3. Optimizing resource utilization and achieving load balancing across edge nodes to enhance system performance.
4. Evaluating the proposed approach against existing methods using performance metrics such as average latency, task execution time, energy consumption, and resource utilization.

The rest of the paper is structured as follows: Section 2 reviews existing research on task scheduling and resource allocation, highlighting the gaps our work addresses. Section 3 describes the system architecture and proposed methodology, focusing on dynamic priority adjustment and resource allocation. Section 4 presents simulation results, demonstrating the effectiveness of our approach in reducing latency, energy consumption, improving resource utilization, and enhancing system performance. Finally, Section 5 concludes the paper and discusses future research directions.

2. Related work

The integration of Internet of Things (IoT) devices with healthcare systems has unlocked significant opportunities, particularly with the advent of Mobile Edge Computing (MEC). MEC enhances system performance by enabling real-time patient monitoring and improving resource efficiency. While wearable sensors and wireless devices have transformed healthcare, they also introduce challenges such as limited computing power, storage, and energy in edge devices. Addressing these issues has become a key focus of research, with numerous resource allocation and task scheduling techniques proposed to mitigate these constraints.

Task Scheduling in Edge Computing: Task offloading and scheduling mechanisms in IoT edge computing networks have been widely studied to reduce latency, conserve energy, and improve task execution. Z Sharif et al. proposed a Priority-based Task Scheduling and Resource Allocation mechanism for healthcare systems using Mobile Edge Computing (MEC). The mechanism prioritizes tasks based on emergency levels derived from real-time health data, ensuring critical tasks are processed promptly to minimize latency and bandwidth costs. However, the approach may face challenges under extreme workloads or with resource-scarce MEC nodes, potentially leading to delays in processing non-critical tasks [5]. The Adaptive and Priority-Based Resource Allocation (A-PBRA) mechanism has shown effectiveness in improving resource utilization and addressing time-constrained edge computing (EC) applications [6]. However, scalability remains a significant challenge, particularly under high-demand scenarios involving an exponential increase in end devices and resource requests. Additionally, the dynamic management of priorities and real-time resource allocation introduces computational and communication overhead, which can strain system resources. Liu et al. introduced an energy-aware scheduling technique that combines task scheduling with dynamic voltage and frequency scaling (DVFS) to optimize energy consumption while maintaining high performance. Although this approach reduces energy usage effectively, it fails to adequately prioritize tasks. This shortcoming leads to longer execution times for latency-sensitive tasks, limiting its applicability in critical healthcare environments [7]. Similarly, mobility awareness improves MEC performance through efficient resource allocation and task offloading. However, scalability concerns in the proposed algorithms may restrict their applicability to larger and more complex network environments [8].

Energy Efficiency in Resource Management: Energy efficiency is a critical concern in healthcare IoT, where devices often operate on constrained power sources. Li et al. proposed a dynamic request scheduling optimization framework for Mobile Edge Computing (MEC) in IoT applications, which effectively minimizes energy consumption and response delays. However, it may face challenges in highly unpredictable network conditions, potentially reducing system welfare and increasing response times [9]. Bio-inspired algorithms such as the Novel Bio-Inspired Hybrid Algorithm (NBIHA) have shown promise in improving resource utilization and reducing average task execution time. NBIHA efficiently

balances workloads but lacks the dynamic prioritization needed to address high-priority tasks in emergency scenarios. Its reliance on static scheduling strategies also makes it less adaptable to varying task loads and the real-time demands of healthcare IoT, resulting in suboptimal latency performance [10]. Alsurdeh et al. investigated a hybrid scheduling strategy combining deadline-aware and priority-based methods. Their approach improved task execution times and accuracy in processing real-time healthcare data. However, the trade-offs between energy consumption and resource utilization efficiency were not adequately addressed, limiting its effectiveness in dynamic and resource-constrained environments [11]. Tong et al. developed an adaptive resource allocation method that dynamically adjusts resources based on task requirements and system load. This approach improves resource utilization but does not explicitly address energy consumption, making it less suitable for healthcare environments dependent on battery-powered IoT devices. Additionally, the lack of flexibility to manage varying levels of task urgency impacts its performance in handling critical healthcare operations, where rapid response times are essential [12]. Similarly, IoT edge computing platforms enhance energy monitoring with real-time sensor data, offering improved performance and reduced latency over traditional cloud systems. They support diverse protocols, scalability, and open-source accessibility. However, challenges in interoperability, data security, and scalability under high-demand scenarios may limit their suitability for large-scale, resource-constrained healthcare applications [13].

Machine Learning for Resource Allocation: Recent studies have leveraged machine learning (ML) techniques to enhance resource allocation in MEC systems. Choudhury et al. developed a predictive ML-based resource allocation framework that forecasts resource demands using historical data. While this approach improves system performance, its high computational complexity makes it unsuitable for resource-constrained edge devices in real-time scenarios [14]. Similarly, the CORA-GT algorithm [15] leverages game theory for computation offloading and resource allocation in multiuser, multi-MEC edge computing environments. It includes a model to handle large numbers of IoT devices and predicts MEC server waiting times based on task queues at base stations. The algorithm optimizes task processing time, sequence, and frequency to enhance efficiency. While the performance evaluation demonstrates reliable energy consumption and time delay, the work does not address challenges related to user mobility or task offloading strategies for communication switching between MEC servers.

Gaps Addressed by DPTARA: Despite these advancements, existing solutions fail to address the combined challenges of latency, resource heterogeneity, task execution time, energy consumption, and resource utilization efficiency in a unified manner. Traditional methods either optimize a single metric, such as energy consumption, at the expense of others or focus on computational resources without considering dynamic task priorities. Furthermore, most approaches are ill-suited to handle the highly variable and urgent nature of healthcare data, where rapid response times are critical.

To address these gaps, the Dynamic Priority-Based Task Scheduling and Adaptive Resource Allocation (DPTARA) system integrates multiple performance objectives into a unified framework:

1. **Latency Reduction**: DPTARA dynamically adjusts task priorities in real-time, ensuring that high-priority healthcare tasks are processed with minimal delay. This mechanism outperforms priority-based method by effectively handling high-traffic scenarios without compromising performance.
2. **Improved Task Execution Times**: By combining predictive load balancing with adaptive resource allocation, DPTARA minimizes task execution times. Unlike dynamic request scheduling approach, which suffers from delays under high workloads, DPTARA ensures consistent task completion times across dynamic healthcare environments.
3. **Energy Efficiency**: DPTARA employs an adaptive energy management strategy that reduces energy consumption by optimizing the power usage of edge and cloud resources. This approach achieves better energy savings compared to DVFS-based method while maintaining high performance for time-sensitive tasks.
4. **Enhanced Resource Utilization**: By balancing workloads across edge and cloud resources, DPTARA significantly improves resource utilization. It addresses the shortcomings of NBIHA by dynamically allocating resources based on task priorities and predicted system states, avoiding underutilization or overloading of edge nodes.

Key Differentiators

- **Unified Framework**: Integrates latency, task execution time, energy consumption, and resource utilization into a cohesive framework, addressing the fragmented focus of existing methods.
- **Scalability and Adaptability**: Predictive load balancing enables DPTARA to scale efficiently, maintaining performance under fluctuating traffic loads.
- **Healthcare-Specific Optimization**: Tailored for the dynamic and urgent requirements of healthcare IoT systems, ensuring reliable and efficient task handling.

DPTARA offers a comprehensive and scalable solution tailored to the dynamic needs of healthcare systems, efficiently handling time-sensitive tasks while optimizing resource and energy usage. By addressing the limitations of existing methods, DPTARA establishes itself as a robust framework for next-generation healthcare IoT environments, ensuring both high performance and adaptability. Table 1 summarizes notable methods and algorithms, highlighting their advantages and limitations, and contrasts them with DPTARA's unique approach [16].

3. System architecture and proposed methodology

The Dynamic Priority-based Task Scheduling and Adaptive Resource Allocation (DPTARA) system is described in this section. It was designed for vital health monitoring services where responsiveness in real-time is crucial. Depending on how urgent a task is, the system dynamically determines whether it should be processed locally at a hospital workstation (HW) or remotely in the cloud. By prioritizing tasks based on their criticality, DPTARA optimizes processing time and improves resource utilization while minimizing energy consumption, ensuring efficient task management and reducing delays in healthcare services. Additionally, reducing processing and response times is paramount in emergency or accidental scenarios to ensure that essential decisions can be made promptly.

3.1. System architecture

The healthcare application follows the DPTARA scheme architecture, comprising three main components: IoT devices, edge servers, and a central cloud, as shown in Fig. 2. IoT devices generate various tasks such as patient monitoring data, emergency alerts, and routine check-up data. These tasks are initially processed by nearby edge servers to minimize latency. When there are not enough resources locally, tasks are shifted to the central cloud. The dynamic priority adjustment algorithm makes sure that important activities are prioritized correctly by continuously assessing and modifying task priorities in response to urgency and real-time situations. The adaptive resource allocation algorithm strikes an optimal balance between responsiveness and efficiency by allocating resources according to task priority and the state of the system. The system's overall performance and responsiveness must be maintained while guaranteeing that high-priority jobs receive the resources they require on time.

Table 1
A synopsis of some of the main methods and techniques suggested for EC.

Author	Technique	Observations	Advantages	Evaluations	Shortcomings
Zubair Sharif, et al. [5](2023)	Priority-based Task Scheduling and Resource-Allocation (PTS-RA)	To address the demands of emergency conditions under a Healthcare Monitoring System (HMS), there is a need for an effective mechanism that encompasses efficient task scheduling and resource allocation in MEC (Mobile Edge Computing).	The task processing time and bandwidth cost were minimized, respectively	Average response time and Network utilization	The limitations and generalizability of the proposed mechanism in different healthcare contexts. Real-World Implementation Challenges.
Piyush Gupta, et al. [17](2023)	CNN-based prediction model	Smart healthcare monitoring system utilizing edge computing and IoT, including a CNN-based prediction model with a 99.23% accuracy rate.	Enhanced efficiency and decreased latency.	Precision, error rate, and processing time reduction.	Limited resource capacity and Network reliability.
Edris et al., [18] (2024)	DLJSF Algorithm for Data Locality	Reduces latency in fog-cloud environments for IoT task scheduling.	Enhances task scheduling performance and lowers latency.	Demonstrated in fog-cloud IoT systems.	Faces data migration overhead, scalability constraints, centralized dependencies, and energy inefficiency.
Jun Liu, et al. [19] (2022)	Polynomial time approximation scheme multi-task allocation in mobile edge computing	In multi-task MEC scenarios, the aim is to minimize the overall energy consumption by overcoming work allocation problems.	Excellent ratio of quality to quickness.	Time complexity and optimality loss.	Limited scalability consideration.
Yu Zhang, Bing Tang, et al. [20] (2022)	Dynamic time-sensitive scheduling algorithm.	Tasks are scheduled according to deadlines and time sensitivity using a dynamic time-sensitive scheduling method.	Minimized latency in edge computing and effectively met user deadlines.	Cost together with processing time averages.	It is essential to consider fault-tolerant techniques to address anomalies in task scheduling.
Baoshan Lu et al., [21] (2023)	Energy-Efficient Scheduling in MEC	Minimizes energy consumption using parallel and sequential computing.	Reduces energy usage and mitigates VM I/O interference.	Focused on mobile edge computing scenarios.	Scalability challenges, increased complexity, and fixed task arrival assumptions.
Boyin Zhang, et al. [22] (2022)	DNN scheduling algorithms, eQDMP algorithm	With an emphasis on partial job migration, two task scheduling algorithms were created for collaborative edge-cloud inference.	Minimize the total amount of delay.	Queuing and network latency.	A limited number of edge devices.
A. S. Abo-hamama, et al. [23] (2022)	A semi-dynamic real-time task scheduling algorithm	A unique semi-dynamic real-time task scheduling method is presented with a focus on high applications.	Decreases the task rate of failure and processing cost.	The rate of failure and overall execution.	Dynamic scheduling problem
Mohammed Maray, et al. [24] (2022)	Heuristic and fully distributed offloading algorithms	Current research on offloading mechanisms in static and dynamic contexts, as well as optimization technique.	Reducing energy usage and execution latency.	Energy usage and delays in execution	Additionally, the heuristic and fully distributed offloading algorithms have not been incorporated.
Ali Raza et al., [25] (2024)	Load Forecasting for Peer-to-Peer (P2P) Energy Trading	Efficiency improvement and cost-effectiveness in smart grids using scheduling and forecasting.	Supports energy trading and enhances resource utilization.	Evaluated in smart grid energy management.	Scalability issues, dependency on accurate predictions, market complexities, and cybersecurity concerns.
Wen He, et al. [26] (2022)	Concave-Convex Procedure (CCCP)-based algorithm	A multi-user FD-MEC system was designed, comprising the phases of data uploading, processing, and retrieval.	Cut down on processing latency and conserve energy.	Energy usage and latency.	Mechanism only for FD communication for uploads and downloads data.
K. Kumaran E. Sasikala [27] (2022)	Dynamic Weighed Quantum Arithmetic Optimization Algorithm (DWQAOA)	To address the NP-hard offloading problem, a Dynamic Weighed Quantum Arithmetic Optimization Algorithm (DWQAOA) was proposed.	To reduce network costs.	Task latency	Failed to incorporate real-time constraints.
Nan et al., [28] (2022)	Genetic algorithm	Fitness function: Energy consumption. Analyzed 10000 tasks	Real-time cloud services with low time delay	The computational time of transmission and energy consumption	Real-world implementation challenges

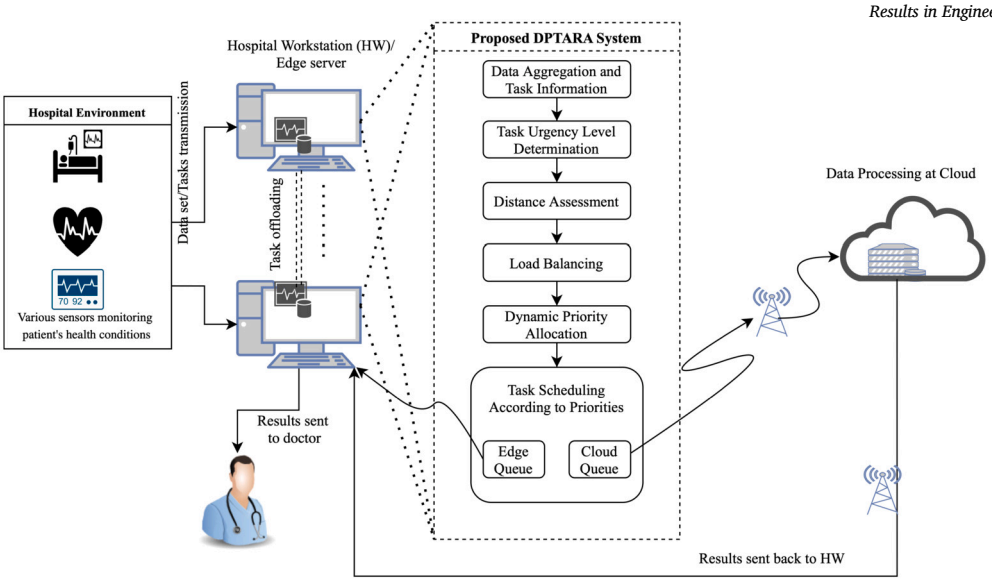

Fig. 2. The DPTARA scheme architecture for the smart hospital workstation.

3.1.1. Task assignment plan in healthcare workstations

In healthcare systems, tasks generated by IoT devices need to be processed efficiently by various hospital workstations (HWs), which act as edge servers. These tasks can include patient monitoring data, medical imaging processing, real-time alerts, and other critical healthcare operations. The task assignment plan aims to distribute these tasks across multiple HWs to ensure timely processing, load balancing, and optimal resource utilization. The proposed DPTARA system addresses these needs through a heuristic approach that optimizes task scheduling and resource allocation in healthcare applications. The task assignment framework involves mapping a set of tasks to different hospital workstations, functioning as edge servers in the healthcare network. The plan is formulated to ensure that the most critical tasks are assigned to the most suitable resources based on their priority and the current load on the resources.

The task assignment algorithm considers the following factors:

- Task priority
- Resource availability
- Predicted load
- Energy consumption

By considering these factors, the system ensures that tasks are processed efficiently and effectively, leading to improved patient care and resource utilization.

3.2. Problem formulation

In computing at the edge contexts, resource allocation is a crucial issue, especially when dealing with tasks of varying priority levels. The objective is to ensure that high-priority tasks (e.g., real-time applications) are allocated sufficient resources while optimizing the overall system performance, including normal-priority tasks. The challenge arises when the available resources are limited, necessitating decisions on whether to allocate resources locally or offload tasks to the cloud.

3.2.1. Problem statement

Given a set of tasks generated by IoT devices, each classified as either high-priority (urgent) or normal-priority, the goal is to efficiently allocate the limited resources available at the edge nodes. In this context, A_r represents the total available resources at the edge nodes. The resource demand for high-priority tasks is denoted as P_r, which is derived from the set of priority-based requests R_p, while the resource demand for normal-priority tasks is denoted as N_r, corresponding to the set of normal requests R_n.

The system must handle the following scenarios:

1. **Resource Allocation for Priority-Based Requests (R_p):**

   - When resources are available and sufficient to meet all R_p ($A_r \geq P_r$).
   - When resources are available but insufficient ($A_r < P_r$).
   - When no resources are available locally ($A_r = 0$).

2. **Resource Allocation for Normal Requests (R_n):**

   - When resources are available and sufficient to meet all R_n ($A_r \geq N_r$).
   - When resources are available but insufficient ($A_r < N_r$).
   - When no resources are available locally ($A_r = 0$).

The problem aims to maximize the utilization of local resources while minimizing latency, particularly for high-priority tasks, and efficiently offloading tasks to the cloud when necessary.

3.2.2. Mathematical formulation

Let $R = \{R_p, R_n\}$ be the set of all requests, where:

$$R_p = \{pr_1, pr_2, \ldots, pr_k\} \tag{1}$$

Equation (1), represents priority-based requests that are latency-sensitive, such as emergency alerts or real-time processing tasks, and the corresponding resource demand P_r is a function of these requests.

$$R_n = \{nr_1, nr_2, \ldots, nr_m\} \tag{2}$$

Equation (2), represents normal requests that are delay tolerant, such as offline data backups or non-urgent computational tasks, with N_r being their corresponding resource demand.

Let $RC_i = \{rc_1, rc_2, \ldots, rc_u\}$ represent the set of computing and memory resources available at the edge nodes. Each rc_i represents a combined metric of CPU cycles ψ_i (in MIPS) and RAM capacity (in GB), as presented in Table 4. For offloaded tasks, cloud resources RC_{cloud} provide additional capacity, including higher CPU and RAM availability.

The total available resources in all edge nodes, denoted Ar, are calculated as the sum of the resource metrics ar_i for each edge node. Mathematically, this is expressed as follows:

$$Ar = \sum_{i=1}^{l} ar_i, \tag{3}$$

where ar_i represents the aggregated metric combining the available CPU and RAM resources of an edge node. To determine ar_i, the relative importance of CPU and RAM is considered using assigned weights w_c and w_m, respectively, such that $w_c + w_m = 1$. These weights ensure a balanced representation of resource capacities based on their significance to task processing.

Each resource type is normalized by dividing its value by the maximum capacity available in the system. This normalization ensures that the aggregated metric ar_i is consistent and comparable across edge nodes. After normalization, each resource is weighted based on its importance to the tasks being processed. By aggregating these metrics across all edge nodes, Ar provides a unified representation of the total available resources in the system, enabling efficient task allocation and resource utilization. Tasks within this framework include diverse applications such as Virtual Reality (VR), Augmented Reality (AR), image and video processing, healthcare data analysis and general-purpose tasks from end users. Each task, represented as task $= \{\delta_i, \psi_i\}$, where δ_i denotes the input data size and ψ_i specifies the required CPU cycles, is categorized into high-priority or normal-priority tasks.

3.2.3. Optimization objectives

The optimization objectives based on the available resources are as follows:

3.2.3.1. Maximize resource utilization Maximize the resource utilization U_r at the edge nodes using Equation (4):

$$U_r = \frac{\left(\sum_{i=1}^{k} pr_i + \sum_{j=1}^{m} nr_j\right)}{\sum_{i=1}^{u} rc_i} \tag{4}$$

where:

- pr_i: Resource requirements for priority tasks.
- nr_j: Resource requirements for non-priority tasks.
- rc_i: Total resource capacity available.

This formulation optimizes resource utilization by balancing the allocation between priority and non-priority tasks, ensuring no resources are left idle while addressing workload demands effectively.

3.2.3.2. Minimize task latency for high-priority tasks Efficient task execution in edge-cloud systems requires minimizing latency, particularly for high-priority tasks, to ensure timely processing and responsiveness. In the proposed framework, latency is modeled as the total time required to process a task, encompassing both transmission delays and computing times. Tasks can be executed either locally at edge servers or offloaded to cloud servers based on resource availability, task priority, and latency constraints.

To achieve latency minimization, the system dynamically evaluates and compares two latency scenarios:

1. **Local Latency (L_{local})**: Tasks processed at edge servers.
2. **Cloud Latency (L_{cloud})**: Tasks offloaded to cloud servers.

To gain insights into these scenarios, the following sections describe the latency models and their respective components in detail.

Latency calculation and components: The total latency for task execution is divided into two primary components, each contributing to the overall processing time:

1. **Transmission Time**: The time required to transfer task data from IoT devices to the processing server (either edge or cloud), which depends on network bandwidth, distance, and intermediate nodes.
2. **Computing Time**: The time required to execute the task based on the processing capacity of the selected server, considering factors such as task size and hardware performance.

Local Processing Latency (L_{local}): Tasks executed locally at edge servers experience latency given by:

$$L_{\text{local}} = T_{\text{tr, edge}} + T_{\text{comp, edge}} \tag{5}$$

where:

- $T_{\text{tr, edge}}$: Transmission time from IoT devices to edge servers.
- $T_{\text{comp, edge}}$: Processing time for tasks at edge servers.

The transmission time depends on network conditions, modeled in the range 5–10 ms (as shown in Table 3). The computing time is calculated as:

$$T_{\text{computing, edge}} = \frac{\delta}{\psi_{\text{edge}}} \tag{6}$$

where:

- δ: Task size (in MB or similar units).
- ψ_{edge}: Processing capacity of the edge server (in MIPS or tasks/sec).

Cloud Processing Latency (L_{cloud}): For tasks offloaded to cloud servers, the total latency comprises the transmission delay from IoT devices to the cloud and the processing time at the cloud server. This is expressed as:

$$L_{\text{cloud}} = T_{\text{tr, cloud}} + T_{\text{comp, cloud}} \tag{7}$$

where:

- $T_{\text{tr, cloud}}$: Total transmission time, which includes delays from IoT devices to edge servers and further to the cloud, as defined in Equation (8).
- $T_{\text{comp, cloud}}$: Processing time at the cloud server.

The total transmission time ($T_{\text{tr, cloud}}$) is calculated as:

$$T_{\text{tr, cloud}} = T_{\text{tr, edge}} + T_{\text{edge-cloud}} \tag{8}$$

where:

- $T_{\text{edge-cloud}}$: Network delay between edge servers and the cloud.

The transmission delay is influenced by network conditions and is typically within the range of 50–100 ms (as shown in Table 3). The processing time at the cloud is determined by the size of the task and the cloud server's processing capacity, calculated as:

$$T_{\text{computing, cloud}} = \frac{\delta}{\psi_{\text{cloud}}} \tag{9}$$

where:

- δ: Task size (in MB or similar units).
- ψ_{cloud}: Processing capacity of the cloud server (in MIPS or tasks/sec).

Decision Framework for Minimizing Latency: To ensure minimum latency for task execution, the proposed framework dynamically evaluates and compares the local and cloud latency scenarios, selecting the processing mode with the lowest latency. The overall latency (L_r) is minimized by choosing the mode that guarantees minimal execution time, as expressed in Equation (10):

$$L_r = \min\left(L_{\text{local}}, L_{\text{cloud}}\right) \tag{10}$$

Tasks with high-priority requirements are assigned to the processing mode that ensures timely processing. Specifically, if resources are sufficient at the edge, tasks are processed locally to exploit low transmission latency. However, if local resources are insufficient, tasks are offloaded to the cloud, taking into account the higher network delays associated with cloud processing. The optimization strategy prioritizes high-priority tasks to maintain low latency while simultaneously offloading non-priority tasks to balance resource utilization across the system effectively.

3.2.3.3. Ensure fairness in resource allocation Fairness in resource allocation is achieved by balancing the needs of high-priority tasks (Rp) and normal-priority tasks (Rn) while maintaining system efficiency. Upon receiving resource requests, the edge node (EN) identifies the task type (Rp or Rn) and checks the availability of resources. If resources are sufficient, they are allocated directly to the request. However, in cases of insufficient or unavailable resources, the system employs adaptive strategies to ensure fairness:

1. **For High-Priority Tasks (Rp):** When resources are insufficient, the EN reallocates resources by deallocating them from normal-priority tasks (Rn). Incomplete Rn tasks are offloaded to the cloud, ensuring that critical high-priority tasks are processed without delay. This ensures Rp tasks meet their deadlines, minimizing execution time and latency.
2. **For Normal-Priority Tasks (Rn):** If resources are partially available, the system allocates the available resources to Rn and offloads the remaining demands to the cloud. Additionally, Rn tasks may be divided into subtasks and processed in series or parallel to maximize resource utilization. This approach ensures progress for Rn tasks while maintaining system efficiency and balancing energy consumption.
3. **Dynamic Adjustments:** Task priorities are updated in real time based on urgency, resource availability, and system load. For example, a routine Rn task may be elevated to Rp if an abnormality is detected. This adaptive mechanism allows the system to respond effectively to changing conditions, ensuring fairness and preventing resource bottlenecks.

These fairness mechanisms, particularly the dynamic reallocation of resources and adaptive adjustments, directly contribute to the improved task execution time, resource utilization, and energy efficiency observed in DPTARA's performance. By balancing the needs of Rp and Rn tasks, the system achieves high efficiency while addressing critical challenges in resource-constrained edge computing environments.

3.3. DPTARA system design framework

3.3.1. Task types and priority levels

Tasks are categorized into different types based on their urgency and resource requirements:

- Emergency Tasks: Require immediate processing and have the highest priority.
- Routine Monitoring Tasks: Have lower priority but are crucial for continuous patient monitoring.
- Data Aggregation Tasks: Involve collecting and processing large amounts of data, typically with the lowest priority.

Each task type is assigned a priority level that can change dynamically based on system state and task urgency. For instance, a routine monitoring task may be elevated to a higher priority if it detects an abnormality in the patient's data.

3.3.2. Resource allocation framework

The Resource Allocation Framework ensures optimal task scheduling and resource utilization through two main components:

1. **Dynamic Priority Assignment:** Task priorities are updated in real-time based on the following factors:
   - Task urgency.
   - Current resource availability.
   - Overall system load.
2. **Adaptive Resource Allocation Algorithm:** This algorithm dynamically distributes resources by considering:
   - The updated task priorities.
   - The current state of available system resources.

This framework is designed to adapt to changing system conditions, ensuring that critical tasks are prioritized and resources are utilized efficiently. These mechanisms form the foundation for dynamic task management in resource-constrained environments.

3.3.3. Dynamic task prioritization

Building on the Resource Allocation Framework, dynamic task prioritization plays a pivotal role in aligning task scheduling with system objectives. The DPTARA algorithm employs a real-time task prioritization method to effectively manage tasks of varying significance, urgency, and resource requirements. This process is governed by the Dynamic Task Prioritization (P_i) function, which evaluates key factors such as task deadlines to prioritize time-sensitive tasks, computational requirements to allocate resources efficiently, system conditions—including current load, resource availability, and energy constraints—to optimize scheduling, and external factors like network latency or fluctuating energy availability to adapt to changing environments. Certain edge applications have stringent time constraints that must be met by the completion of tasks. Priorities for deadline-constrained scheduling algorithms can be set depending on deadlines, and resources can be allocated appropriately to fulfill schedule limitations. To guarantee timely task completion, these algorithms take into account variables including task characteristics, execution time estimation, and communication delays [29]. We introduce a mechanism to dynamically alter task priorities based on real-time circumstances like task aging, current load, and system state, as stated in Algorithm 2.

The prioritization is governed by the following enhanced priority equation:

$$P_i = \alpha \cdot P_{\text{orig},i} + \beta \cdot \left(\frac{T_{\text{current}} - T_{\text{arrival},i}}{D_i} \right) - \gamma \cdot L_{\text{current},i} \tag{11}$$

Where:

- P_i: The dynamically calculated priority of task i. This value determines the task's scheduling order in the system.
- α: A scaling coefficient that determines the weight of the original priority ($P_{\text{orig},i}$) in the final calculation.
- $P_{\text{orig},i}$: The initial priority assigned to task i based on its intrinsic importance or application-specific requirements.
- β: A scaling coefficient that determines the weight of the aging factor.
- $\frac{T_{\text{current}} - T_{\text{arrival},i}}{D_i}$: The aging factor, representing the fraction of the task's deadline that has elapsed since its arrival. Here:
  - T_{current}: The current time in the system.
  - $T_{\text{arrival},i}$: The time when task i arrived in the system.
  - D_i: The deadline of task i, representing the maximum allowable time for its completion.

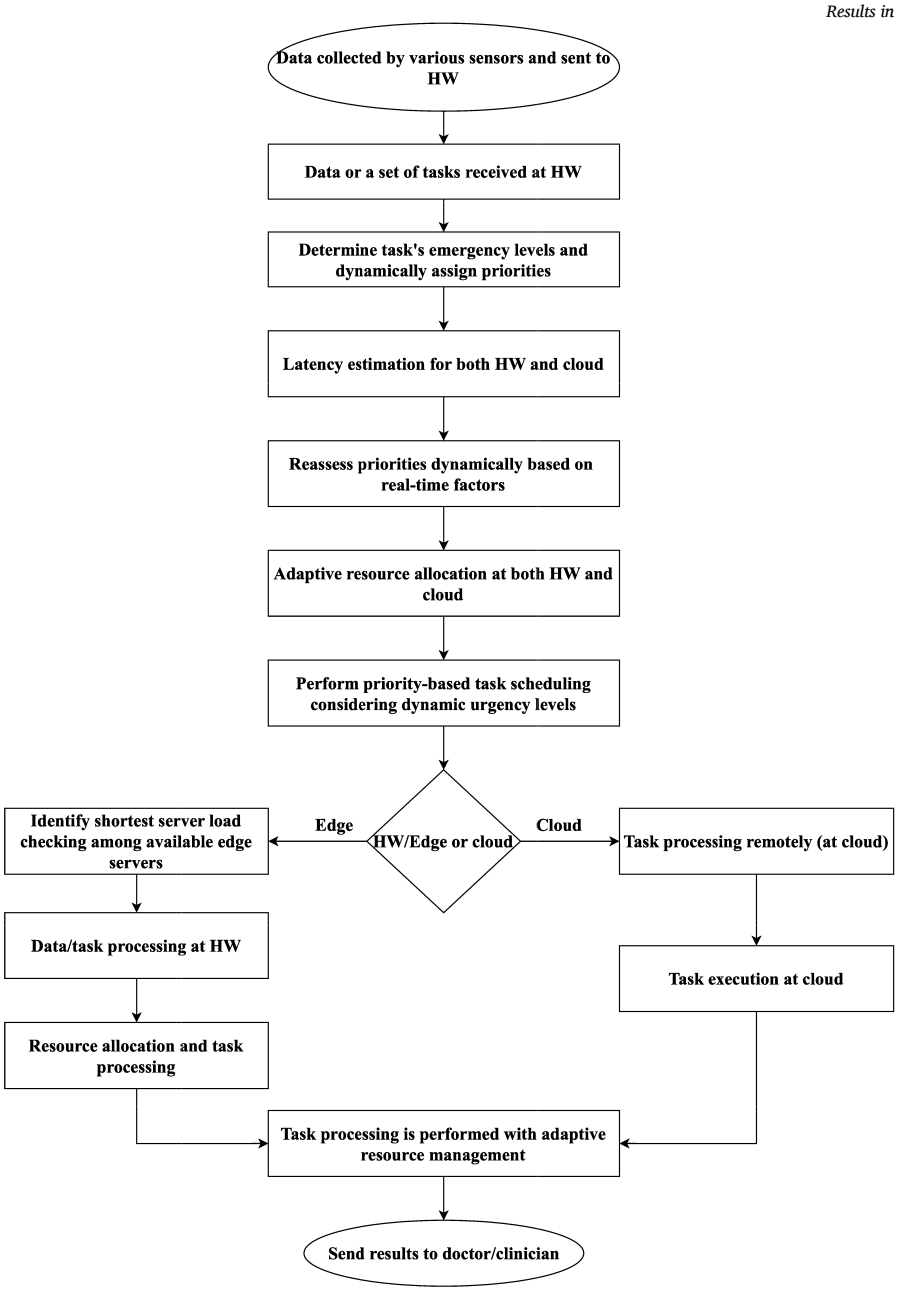

Fig. 3. The flowchart depicts the procedure for allocating tasks to either edge (HW) or the central cloud.

This component ensures that tasks closer to their deadlines are prioritized more urgently.

- γ: A scaling coefficient that determines the impact of the current resource load on the task's priority.
- $L_{\text{current},i}$: The current load on the resource allocated to task i. It reflects the utilization of the resource and helps balance the workload by penalizing tasks assigned to heavily loaded resources.

This Equation (11) ensures that the adjusted priority P_i increases for tasks with:

- Higher original priority ($P_{\text{orig},i}$), reflecting their inherent importance.
- Nearing deadlines ($\frac{T_{\text{current}}-T_{\text{arrival},i}}{D_i}$), signaling urgency.

Conversely, P_i decreases for tasks allocated to heavily loaded resources ($L_{\text{current},i}$), promoting load balancing.

Tasks are divided into two priority categories: high and normal(low). Tasks having a high priority may be vital tasks that need to be completed as soon as possible, whereas jobs with a low priority can have greater latitude in their schedule. Because of the fluid nature of this prioritization, the system may adapt in real-time to changes in task needs or resource availability, guaranteeing that the computational resources needed for the most crucial tasks are allocated to them.

3.3.4. *Resource allocation and load balancing*

To prevent overtaxing some edge devices and underutilizing others, load balancing seeks to divide the computing load among them equally. In order to assign tasks intelligently, load balancing algorithms take into account various aspects, including network conditions, device capabilities, and the present workload. The overall performance of the system can be enhanced by load balancing, which guarantees effective resource usage and minimizes reaction time variances [30].

Algorithm 1 DPTARA Algorithm.

Input: Tasks $T = \{T_1, T_2, \ldots, T_n\}$, Local Resources R_L, Edge Resources R_E, Cloud Server R_C
Output: Task execution times, load balancing rates, energy consumptions
Initialize $task\_metrics = \{\}$, $load\_forecaster \leftarrow$ Linear Regression
Categorize tasks into high-priority (R_p) and normal-priority (R_n)
Compute total resource demand: $P_r = \sum_{T_i \in R_p} T_i.size$, $N_r = \sum_{T_i \in R_n} T_i.size$
for each task $T_i \in T$ **do** ▷ Start of outer loop
  Adjust T_i's priority using DynamicTaskPrioritization(T_i)
  Set $selected\_resource =$ `None`, $min\_load = \infty$
  for each resource $R_j \in R_{total}$ **do** ▷ Start of inner loop
    if $R_j.current\_load + T_i.size \leq R_j.capacity$ **and** $R_j.current\_load < min\_load$ **then**
      Set $min\_load = R_j.current\_load$, $selected\_resource = R_j$
    end if
  end for ▷ End of inner loop
  if no local or edge resource can handle the task **then**
    if $A_r < P_r$ **then** ▷ Prioritize high-priority tasks
      Allocate available resources to R_p
    end if
    if $R_C.current\_load + T_i.size \leq R_C.capacity$ **then**
      Set $selected\_resource = R_C$
    else
      Print "All servers are overloaded!"
    end if
  end if
  $selected\_resource.queue.put(T_i)$, $selected\_resource.current\_load += T_i.size$
  Print "Task `T_i.id` allocated to `selected_resource.resource_type` server `selected_resource.resource_id`."
end for ▷ End of outer loop
for each resource $R_j \in R_{total}$ **do** ▷ Start of resource processing loop
  while $R_j.queue$ is not empty **do** ▷ Start of task processing loop
    $T_i \leftarrow R_j.queue.get()$, $start \leftarrow$ `current_time()`
    Simulate task execution, compute $T_i.completion\_time$ ▷ in ms
    Calculate η_j **using:**

$$\eta_j = \frac{U_{effective}}{C_{total}} \times (1 - R_{downtime})$$

    $T_i.energy\_consumption = \sum_{i=1}^{n} \left(\frac{P_r}{\eta_j}\right) \times (end\_time - start\_time)$ ▷ in joules
    Update $task\_metrics[T_i.id]$
  end while ▷ End of task processing loop
end for ▷ End of resource processing loop
Predict future loads: $future\_loads = load\_forecaster.predict(task\_sizes)$
Print "Predicted loads: " `future_loads`
Plot results: `plot_all_results(task_metrics)`

Algorithm 2 DynamicTaskPrioritization.

function DynamicTaskPrioritization(T_i)
  Input: Task T_i
  Output: Adjusted priority of T_i
  Calculate P_i **using:**

$$P_i = \alpha \cdot P_{orig,i} + \beta \cdot \left(\frac{T_{current} - T_{arrival,i}}{D_i}\right) - \gamma \cdot L_{current,i}$$

  return $T_i.priority$
end function

3.3.4.1. Algorithm 1: dynamic priority-based task scheduling and adaptive resource allocation (DPTARA) The DPTARA Algorithm 1 is designed to optimize the scheduling of incoming tasks T_i and allocate available resources R_j in a hierarchical computing environment that includes local devices (HW), edge servers, and the cloud. The algorithm dynamically prioritizes tasks based on urgency levels, data size, and waiting time, ensuring that higher-priority tasks are handled first while adhering to system constraints such as resource capacity and latency requirements. This approach minimizes latency, reduces energy consumption, and enhances resource utilization across the system.

The resource allocation process is governed by an adaptive strategy defined by the following equation (12),

$$f(T_i) = \begin{cases} R_j, & \text{if } \min\left(\sum_{j=1}^{m} \left(R_j.\text{current\_load} + S_i\right) \leq R_j.\text{capacity}\right) \\ & \text{and } R_j \in R_{\text{total}} \text{ and priority allows,} \\ R_C, & \text{if no local or edge resource can handle the task, and} \\ & R_C.\text{current\_load} + S_i \leq R_C.\text{capacity,} \\ \text{None}, & \text{if all servers are overloaded.} \end{cases} \tag{12}$$

Where:

- R_j: The available resource at the local or edge layer.
- R_C: The cloud resource.
- S_i: The size of task T_i.
- R_{total}: The total set of resources in the system.

A task T_i is allocated to a resource R_j if the sum of the resource's current load and the task size S_i does not exceed the resource's capacity and the task's priority aligns with the scheduling policy. If local resources are insufficient, tasks are offloaded to edge servers. If edge servers are also unable to process the task, it is transmitted to the cloud, provided that the cloud's capacity permits. In cases where all resources are overloaded, tasks may be deferred or dropped based on the system's failure-handling policy. This hierarchical offloading mechanism ensures that latency-critical and urgent tasks are processed promptly, while less urgent tasks are managed efficiently.

The algorithm incorporates a queue management system for tasks offloaded to the cloud. Tasks waiting for cloud execution are queued based on weighted priorities determined by urgency levels, data size, and waiting time. This queuing strategy reduces overall task processing delays and improves system throughput. Once a task is processed, the results are transmitted back to the originating hardware, completing the task lifecycle.

Key constraints for the resource allocation process include ensuring that the load on each resource, including local, edge, and cloud servers, does not exceed its capacity. Tasks are also assigned only to resources that align with their priority levels and latency requirements. This structured allocation ensures that system performance is not compromised even under high workloads. The constraints for resource allocation, as outlined in Equation (12), are defined as follows:

$$C_1 : \sum_{j=1}^{m} \left(R_j.\text{current\_load} + S_i\right) \leq R_j.\text{capacity},$$

C_2 : Task priority matches the resource policy.

$C_3 : R_C.\text{current\_load} + S_i \leq R_C.\text{capacity}$, for cloud allocation.

The DPTARA algorithm is characterized by its adaptability, efficiency, and scalability. It dynamically adapts to changing task loads and resource availability, ensuring robust performance in dynamic edge-cloud environments. By prioritizing urgent tasks and optimizing the utilization of local and edge resources, the algorithm minimizes processing delays and reduces energy consumption. Its hierarchical task scheduling and offloading mechanisms enable the system to handle a high volume of tasks, making it highly scalable for various IoT and edge-computing applications.

The performance of the proposed DPTARA algorithm has been thoroughly evaluated through extensive simulations, with detailed results elaborated in the last section Results and Discussion. Key findings highlight the algorithm's significant advantages in optimizing task scheduling and resource allocation across hierarchical computing environments. DPTARA demonstrates a marked improvement in latency reduction, resource utilization, throughput, and energy efficiency compared to baseline methods. Specifically, it effectively balances computational

loads, reduces average task execution times, and minimizes cloud offloading for latency-tolerant tasks, ensuring efficient energy consumption. These results affirm DPTARA's capability to address the challenges of task scheduling in edge-cloud systems, offering a scalable and robust solution tailored for dynamic environments. Fig. 3 illustrates the flowchart for the procedure of allocating tasks to either local hardware or the central cloud.

Algorithm 3 CalculateBalancingRate.

```
function CALCULATEBALANCINGRATE(R_j)
    Input: Resource R_j
    Output: Load balancing rate for R_j
    total_tasks = ∑_{R_k ∈ R_total} R_k.current_load
    num_resources = |R_total|
    average_load = total_tasks / num_resources
    balancing_rate = (1 − |R_j.current_load − average_load| / average_load) × 100
    return balancing_rate
end function
```

Initialization: To monitor task execution times, load balancing rates, and energy consumption, the program initializes many data structures. In addition, it builds up a load forecasting model that uses historical data to forecast future resource demands.

1. **Task Allocation**: The method looks through all of the resources in $R_{\text{total}} = R_L \cup R_E \cup \{R_C\}$ for each task T_i. If the resource R_j can handle the task's size S_i without going over capacity, then that resource is given the assignment. For the purpose of ensuring a fair allocation of jobs throughout the network, the resource with the lowest overall load is chosen.
2. **Cloud Server as Backup**: The method verifies the R_C of the cloud server if no local or edge resources are able to complete the task. The algorithm recognizes that all servers have exceeded their capacity and flags the issue if the cloud server is likewise overwhelmed and cannot complete the task.
3. **Load Balancing and Execution**: The task's load is modified as a result of the selected resource queuing it for execution. Tasks in the queue are subsequently processed by the algorithm, which updates metrics like execution time and energy usage while simulating their execution. The effectiveness and efficiency of the resource allocation plan must be evaluated using these metrics.

Predictive Load Balancing: Predictive load balancing distributes tasks across various healthcare workstations (HWs) and servers by forecasting future loads. This approach ensures that no single resource is overwhelmed, and tasks are completed efficiently. To prevent future resource bottlenecks, the DPTARA algorithm employs load balancing using equation (13),

$$L_{\text{pred},t} = L_{\text{current},t} + \sum_{i=1}^{n} \lambda_i \cdot S_i \cdot \left(1 - p_i\right) \tag{13}$$

This equation forecasts future loads on each resource by considering the current load, task arrival rates, task sizes, and the probability of successful task completion. By anticipating future loads, the algorithm can distribute tasks in a way that balances the overall system load, ensuring that no single resource is overwhelmed. This is particularly important in healthcare environments, where the workload can fluctuate significantly.

3.3.5. Adaptive energy management

Efficient energy management is a critical aspect of edge-cloud systems, particularly in healthcare applications where resource constraints and task priorities necessitate optimized scheduling and execution strategies. To address these challenges, the proposed framework integrates an adaptive energy management strategy, as outlined in Algorithm 3, as part of the DPTARA algorithm to dynamically monitor and manage energy consumption based on task priorities, resource utilization, and real-time workload demands.

To support the adaptive energy management strategy, the framework incorporates a robust efficiency metric (η_i) that evaluates resource utilization effectiveness under varying load conditions. This metric serves as the foundation for dynamic energy optimization by monitoring resource performance and reliability.

Efficiency Metric (η_i): The energy management framework employs an efficiency metric (η_i) to quantify resource utilization effectiveness under varying load conditions. This metric, referred to as Utilization Efficiency, evaluates how efficiently computational resources are utilized, accounting for both operational reliability and resource availability. To determine utilization efficiency, the concept of effective resource utilization is introduced. It represents the portion of resources actively and productively used during task execution. Effective utilization depends on both high-priority (pr_i) and normal-priority (nr_j) tasks. It is derived from the resource utilization (U_r), as defined in Equation (4), and is adjusted by the Resource Utilization Factor (RUF) to reflect the active and productive time of the resources.

Building on this foundation, the efficiency metric is mathematically expressed as:

$$\eta_i = \frac{U_{\text{effective}}}{C_{\text{total}}} \times (1 - R_{\text{downtime}}) \tag{14}$$

Where:

- $U_{\text{effective}}$:Effective resource utilization, incorporating adjustments for priority tasks and productive time.
- C_{total}: Total resource capacity is referenced in Table 3.
- R_{downtime}: Downtime rate based on historical reliability data.

By incorporating these parameters, the efficiency metric enables the framework to evaluate resource performance in real time, facilitating intelligent scheduling decisions. Building on this metric, the next section introduces an enhanced energy consumption model that leverages these insights for dynamic energy optimization.

3.3.5.1. Energy consumption model Building on the efficiency metric (η_i), the proposed energy consumption model captures energy usage dynamics by considering resource-specific efficiency, power consumption, and task execution times. This model not only estimates energy consumption under varying workloads but also guides adaptive scheduling decisions to minimize energy overhead. The energy consumption (E) for each task is calculated using equation (15).

$$E = \sum_{i=1}^{n} \left(\frac{P_i}{\eta_i}\right) \cdot \left(T_{\text{end},i} - T_{\text{start},i}\right) \tag{15}$$

Where:

- P_i: Power consumption of resource i.
- η_i: Efficiency metric for resource i, as defined in Equation (14).
- $T_{\text{start},i}$ and $T_{\text{end},i}$: Task execution start and end times, respectively.

Dynamic Energy Optimization Strategies: Leveraging the insights provided by the efficiency metric and energy model, the framework implements several optimization strategies to dynamically manage resources, minimize energy wastage, and prioritize high-priority tasks. These strategies focus on maintaining system responsiveness while ensuring energy-efficient operation:

1. **Dynamic Resource Allocation:** Prioritizes high-priority tasks for low-latency edge processing, minimizing reliance on energy-intensive cloud resources.
2. **Idle Resource Management:** Transitions idle resources to low-power states during periods of inactivity, reducing energy wastage.

Table 2
Notations and Descriptions.

Notation	Description
P_i	Dynamic priority of task i
R_p	Resource allocation for Priority requests
R_n	Resource allocation for Normal requests
A_r	Available resources
R_j	Selected resources
L_{local}	Tasks processed at edge servers.
L_{cloud}	Tasks offloaded to cloud servers.
δ	Task size (in MB or similar units).
ψ_{edge}	Processing capacity of the edge server (in MIPS or tasks/sec).
$T_{\text{tr, cloud}}$	Total transmission time from IoT devices to edge servers and then to the cloud.
ψ_{cloud}	Processing capacity of the cloud server (in MIPS or tasks/sec).
α	Weight factor for the original priority of the task
$P_{\text{orig},i}$	Initial priority assigned to task i
β	Weight factor for the urgency of the task
$T_{\text{tr, edge}}$	Transmission time from IoT devices to edge servers.
$T_{\text{comp, edge}}$	Processing time for tasks at edge servers.
T_{current}	The current time in the system
$T_{\text{arrival},i}$	The time when task i arrived in the system
D_i	The deadline by which task i must be completed
γ	Weight factor for the current load on the assigned resource
$L_{\text{current},i}$	The current load on the resource that is handling task i
$L_{\text{pred},t}$	The predicted future load on resource t
$L_{\text{current},t}$	The current load on resource t
$T_{\text{comp, cloud}}$	Processing time for tasks at the cloud server.
$T_{\text{edge-cloud}}$	Network delay between edge servers and the cloud.
n	The number of tasks or task types considered in the prediction
λ_i	The rate at which tasks of type i arrive for resource t (Arrival Rate)
S_i	The average size of tasks of type i for resource t (Average Task Size)
p_i	The probability that a task of type i will be completed successfully
E	Total energy consumption of the system
P_i	The power usage of resource i (Power Usage)
η_i	The efficiency of resource i under current load conditions (Utilization Efficiency)
$T_{\text{end},i}$	The time when resource i finishes its task (End Time)
$T_{\text{start},i}$	The time when resource i starts its task (Start Time)
O	The overall objective function that balances multiple objectives
ω_1	Weight factor for minimizing completion time
$T_{\text{completion}}$	The total time taken to complete all tasks (Completion Time)
ω_2	Weight factor for minimizing energy consumption
ω_3	Weight factor for maximizing load balancing rate
R_{balance}	The rate at which the system balances the load across resources
η_{resource}	Efficiency of resource utilization in the system
$U_{\text{effective}}$	The actual productive usage of the resource
C_{total}	The maximum possible capacity of the resource
R_{downtime}	The proportion of time the resource is unavailable due to maintenance or failures

3. **Intelligent Task Offloading:** Offloads tasks with lower urgency or higher energy requirements to cloud servers, leveraging their processing efficiency.
4. **Energy-Aware Scheduling Decisions:** Predicts future workload patterns and adjusts configurations proactively to optimize energy usage.

These strategies enable the framework to dynamically respond to fluctuating workloads, reducing unnecessary energy consumption without compromising system performance. Building on this foundation, the framework also addresses practical deployment considerations in real-world scenarios, ensuring scalability and sustainability.

The proposed framework effectively addresses real-world challenges in healthcare-oriented edge computing by dynamically optimizing resource utilization based on real-time workloads and energy profiles. The DPTARA algorithm minimizes energy wastage through proactive adjustments, prioritizes high-energy tasks for efficient resources, and transitions idle resources to low-power states. Its energy-aware scheduling ensures scalability and reliability, making it suitable for unpredictable healthcare workloads.

The proposed DPTARA system incorporates several key components, including task generation from IoT devices, dynamic priority adjustment, predictive load balancing, adaptive energy management, and a multi-objective optimization strategy. The notations used in the system model are detailed in Table 2.

4. Results and discussions

Extensive simulations were conducted using Python to evaluate the performance of the proposed Dynamic Priority-Based Task Scheduling and Adaptive Resource Allocation (DPTARA) algorithm within a healthcare-oriented edge computing environment. The system integrated Internet of Things (IoT) devices, edge nodes (ENs), and a centralized cloud server to emulate a realistic healthcare infrastructure. Each hospital workstation (HW), acting as an edge node (EN), was equipped with its own processing capabilities, including CPU power, memory, and other computational resources. The allocation of resources to received tasks was conducted based on task-specific parameters such as priority (urgency level), processing time, input data size, and the maximum time delay the task could tolerate. These criteria ensured that high-priority tasks were executed promptly, minimizing latency, while lower-priority tasks were queued or offloaded to the cloud server as necessary.

The simulation environment parameters and specifications, along with the system properties for the experimental setup, are comprehensively detailed in Table 3 and Table 4, respectively. The system consisted of 50 IoT devices generating tasks, 10 HWs serving as edge servers, and a

Table 3
System Parameters and Their Values.

Parameter	Value
Number of IoT devices	50
Number of HWs (edge servers)	10
Number of cloud servers	1
Task arrival rate	Poisson distribution ($\lambda = 5$ tasks/sec)
Task size δ_i	Uniform distribution (5-50 MB)
Deadline for tasks	Uniform distribution (10-100 sec)
Power usage (HW)	50 W
Power usage (cloud)	200 W
Network & Transmission Latency (edge)	5-10 ms
Network & Transmission Latency (cloud)	50-100 ms

Table 4
Simulation Hardware Platform.

Parameter	Edge Node	Cloud Node
CPU MIPS ψ_i	[500,3000]	[2000,10000]
CPU Utilization	0.5	2.0(virtual machines)
RAM capacity	32GB	64GB

Table 5
Comparative Performance Analysis.

Metric	DPTARA	A-PBRA	NBIHA	CORA-GT
Latency (ms)	320	350	480	560
Resource Utilization (%)	88	57	48	35
Task Execution Time (ms)	335	540	570	1100
Energy Consumption (J)	321	600	700	802

single cloud server for processing overflow tasks. Tasks were generated following a Poisson distribution with an arrival rate of $\lambda = 5$ tasks/sec, and were characterized by size (5–50 MB) and deadline (10–100 seconds), sampled from uniform distributions. Hardware configurations included edge nodes with computational capacities ranging from 500 to 3000 MIPS and 32GB of RAM, while the cloud server provided higher capacities, ranging from 2000 to 10,000 MIPS with 64GB of RAM. Network latency was modeled realistically, with delays of 5–10 ms for edge connections and 50–100 ms for cloud interactions.

The proposed DPTARA algorithm was evaluated using key performance metrics, including latency, average response time of incoming requests for resources, resource utilization, task execution time, and energy consumption at the edge nodes. Moreover, the proposed approach was benchmarked against other algorithms, such as Adaptive and Priority-Based Resource Allocation (A-PBRA) [6], NBIHA [9], and CORA-GT [13]. DPTARA ensures efficient task scheduling through dynamic priority adjustment, adaptive resource allocation, and fairness in resource distribution. By prioritizing high-priority tasks (Rp) while adaptively allocating resources to normal-priority tasks (Rn), DPTARA achieves a balanced allocation of resources, ensuring both categories of tasks are processed effectively, even under varying resource conditions. In contrast, A-PBRA experiences higher latency due to less dynamic prioritization, which impacts its ability to handle resource contention fairly. Similarly, NBIHA and CORA-GT rely on static strategies, making them less adaptable to real-time resource fluctuations, often resulting in resource starvation for Rn tasks.

The evaluation results for various metrics across 100 resource requests, summarized in Table 5, demonstrate that DPTARA consistently outperforms the benchmark algorithms. By ensuring fairness in resource allocation, DPTARA achieves lower latency, higher resource utilization, and balanced task execution times, even under high load. An in-depth analysis of key performance metrics—including latency, task execution time, resource utilization, and energy consumption—highlights the strengths of the DPTARA algorithm in addressing critical challenges in edge computing environments.

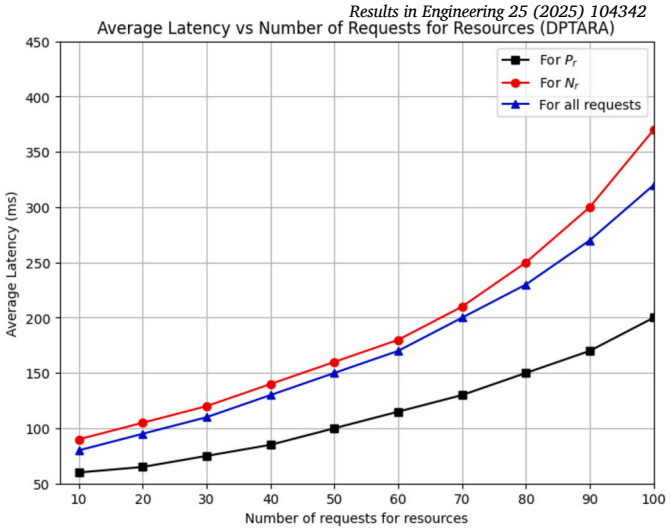

Fig. 4. Average latency in proportion to the quantity of incoming requests.

4.1. Average latency

Latency is a critical metric for healthcare systems where delays can directly impact patient outcomes. DPTARA's dynamic task prioritization ensures that high-priority tasks (Pr) are addressed first, minimizing average latency even as the number of resource requests increases. Fig. 4, illustrates the average latency for both priority-based (Pr)and normal (Nr) types of requests under varying numbers of resource requests. During the simulations, the maximum latency tolerance was set to 250 ms for latency-constrained requests (Pr) and 400 ms for delay-tolerant requests (Nr). The results demonstrate that the proposed DPTARA mechanism adheres to these defined constraints effectively. For priority-based requests (Pr), the average latency consistently remains below the 250 ms threshold, even when handling up to 100 resource requests. Similarly, for normal requests (Nr), the latency remains within the acceptable 400 ms limit. Notably, the average latency for all combined requests grows steadily, reaching approximately 300 ms at 100 requests, which is well below the defined maximum tolerance for normal requests. This improvement is attributed to DPTARA's real-time load balancing and priority adjustment, which dynamically allocates resources to prevent bottlenecks, thereby ensuring efficient processing for critical tasks.

4.2. Average task execution time

DPTARA consistently achieves lower task execution times by employing predictive load balancing and real-time priority adjustments. Unlike CORA-GT, which lacks mechanisms to forecast future resource demands, DPTARA accurately predicts task arrival rates and resource requirements. This predictive capability enables pre-emptive resource allocation, effectively minimizing queueing delays.

Furthermore, while NBIHA attempts to find an ideal match of resources by considering the specific demands of each request, this approach significantly increases waiting time and processing costs due to the static nature of its scheduling mechanism. On the other hand, A-PBRA demonstrates an ability to analyze received request types and adaptively manage resources based on these requirements. Its prior management of resources reduces waiting and processing time, resulting in moderately lower execution times compared to NBIHA and CORA-GT. However, A-PBRA's less dynamic prioritization mechanism limits its efficiency under fluctuating workloads, leaving room for improvement in adaptability and latency reduction.

The findings are presented in Figs. 5–7 and summarized in Table 6. In Fig. 5, the execution time of DPTARA is always better when compared to other benchmarked algorithms (A-PBRA, NBIHA, and CORA-GT). At 50 resource requests, DPTARA achieves an average task execution time

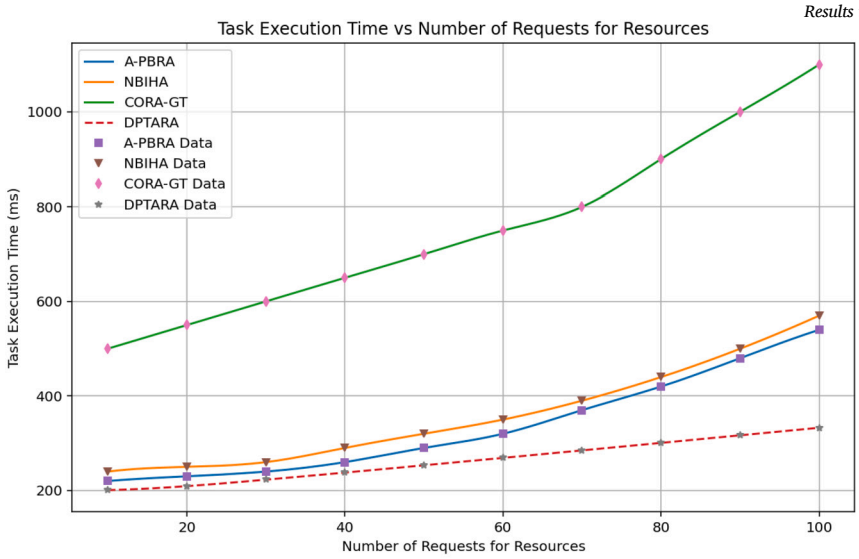

Fig. 5. Analysis of Task execution time with respect to the number of incoming requests.

Table 6
DPTARA's Improvement in Task Execution Time Over Benchmark Algorithms.

Resource Load	A-PBRA (%)	NBIHA (%)	CORA-GT (%)
10 Requests	9.09%	16.67%	60.00%
50 Requests	12.68%	20.94%	63.33%
100 Requests	38.52%	41.75%	69.5%

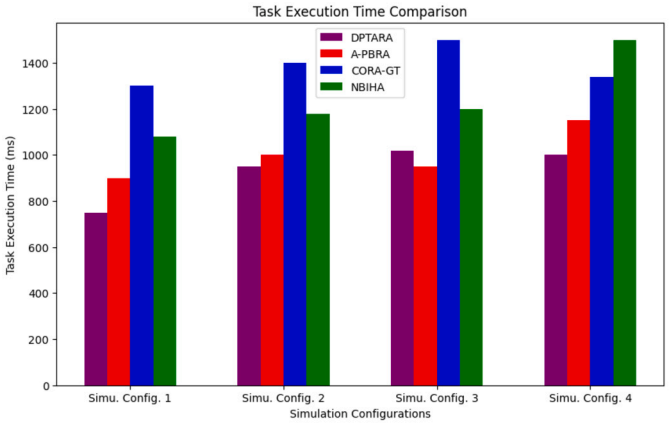

Fig. 6. Task completion time under four different configuration settings.

of approximately 255 ms, representing a 12.68% reduction compared to A-PBRA (around 292 ms) and a significant 63.3% reduction compared to CORA-GT (around 695 ms). Under heavy loads (100 resource requests), DPTARA maintains its efficiency with an execution time of 335 ms, outperforming A-PBRA (around 540 ms), NBIHA (570 ms), and CORA-GT (over 1000 ms). These results emphasize DPTARA's superior ability to manage resource-intensive scenarios effectively, ensuring timely task execution even under high-traffic conditions.

Task execution times of four algorithms—DPTARA, A-PBRA, NBIHA, and CORA-GT—are compared in different configuration as shown in Fig. 6. To test the robustness of the suggested DPTARA, a few crucial parameters are changed for each configuration, including task size, task priority, latency, and the number of tasks (maintained between 100 and 200). The results show that DPTARA performs better than the other three algorithms on all configurations and has the lowest task execution times. This highlights the efficacy of DPTARA, presumably as a result of its dynamic and prioritized job scheduling. A-PBRA also works very

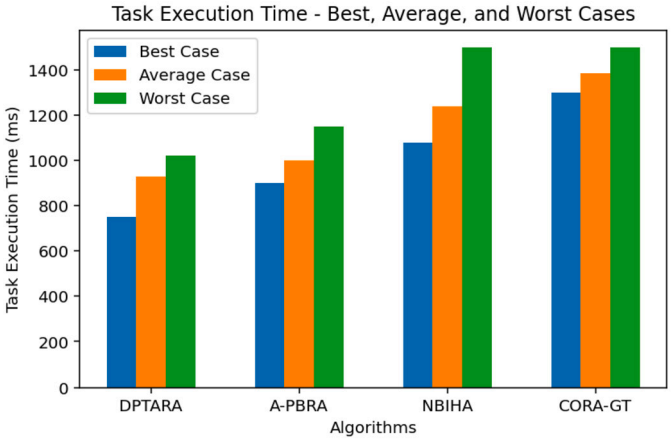

Fig. 7. Task execution time under the best, average, and worst cases.

well, however it is not as optimized as DPTARA. While NBIHA offers a well-rounded approach, CORA-GT, despite its intricate graph-based resource allocation, has the longest execution times and is hence less suitable for time-sensitive applications. DPTARA is the greatest overall when it comes to cutting down on execution times in dynamic environments.

Fig. 7 shows the task execution times for four distinct algorithms: DPTARA, A-PBRA, NBIHA, and CORA-GT, in the best, average and worst-case scenarios. The efficiency of the task is demonstrated by the DPTARA algorithm, which performs best consistently in all scenarios and has the lowest task execution times. The performance of A-PBRA and NBIHA is very close; in the best case scenario, NBIHA marginally outperforms A-PBRA, but in the average and worst instances, it lags. In contrast, CORA-GT exhibits the longest job execution durations, especially under extreme conditions, underscoring its inefficiency in comparison to the other algorithms. When it comes to task execution time, DPTARA is the most efficient algorithm overall, whereas CORA-GT is the least effective, particularly when there is a high load.

4.3. Energy consumption

Energy efficiency is a pivotal concern in healthcare systems, particularly for battery-operated IoT devices that require optimal resource utilization. DPTARA's adaptive energy management strategy dynamically scales power consumption based on task priority and system load,

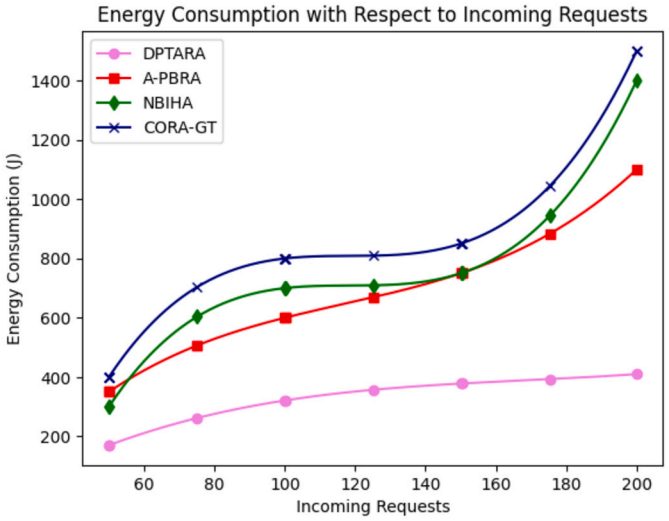

Fig. 8. Energy consumption relative to incoming requests.

ensuring minimal energy wastage. As illustrated in Fig. 8, the energy consumption percentage of DPTARA is always better when compared to other benchmarked algorithms and techniques. DPTARA consistently exhibits the lowest energy consumption compared to other methods, maintaining approximately 215 J at 60 incoming requests, which is 61.6% lower than CORA-GT (560 J), 53.76% lower than NBIHA (465 J), and 50% lower than A-PBRA (430 J). Under high traffic conditions with 200 incoming requests, DPTARA's energy consumption rises modestly to around 410 J, significantly outperforming CORA-GT (1500 J), NBIHA (1400 J), and A-PBRA (1100 J), achieving reductions of 72.66%, 70.7%, and 62.7%, respectively.

According to the aforementioned analysis, CORA-GT's performance significantly declines as additional requests are made due to its inability to ensure fair energy utilization during resource allocation. While NBIHA demonstrates slightly better performance than CORA-GT through improved work scheduling and resource management, its gains remain limited due to a lack of adaptability. Similarly, the A-PBRA technique achieves relatively lower energy consumption by allocating resources based on availability without unnecessary switching. This is facilitated by its effective resource monitoring capabilities and efficient resource management. However, DPTARA outperforms all the aforementioned approaches in terms of energy efficiency. By dynamically adjusting resource allocation instantaneously based on task priorities, DPTARA minimizes reliance on energy-intensive cloud resources for high-priority tasks. Additionally, it intelligently transitions idle resources to low-power states, enhancing overall energy conservation. This superior performance enables DPTARA to consistently achieve lower energy consumption while maintaining task execution efficiency. These results establish DPTARA as a scalable and sustainable solution for energy-constrained healthcare IoT systems, effectively balancing energy utilization and operational efficiency.

4.4. Resource utilization efficiency

Resource utilization efficiency measures how effectively a system uses its available resources. High efficiency indicates productive resource usage with minimal downtime. The DPTARA system achieves this by dynamically adjusting resource allocation based on current and predicted conditions, maximizing productive use of resources, reducing waste, and improving overall performance.

Among the algorithms analyzed, CORA-GT struggles with inefficient resource management, often leading to resource overuse or underuse, which hampers overall system performance. NBIHA, while better at matching resources to task demands, can still suffer from delays in finding the perfect match, leading to periods of resource idleness. On the other hand, the DPTARA system works better than these methods since it modifies resource allocation dynamically in real time. As illustrated in Fig. 9, the resource utilization percentage achieved by DPTARA consistently outperforms other benchmark algorithms and approaches. However, other benchmarked approaches, such as NBIHA and CORA-GT, exhibit uneven variations and fluctuations in resource utilization DPTARA routinely outperforms all other algorithms—A-PBRA, NBIHA, and CORA-GT—by maintaining stable and efficient resource utilization, demonstrating its robustness and adaptability under varying traffic loads.

It obtains approximately 94% utilization at 200 tasks, whereas A-PBRA has a rate of 77%. While CORA-GT does much worse, never using more than 54%, NBIHA performs differently, peaking at around 62%, showing instability under various loads. These results show approximately twice the utilization efficiency of CORA-GT and definitely outperform A-PBRA by roughly 40% in high-demand conditions. DPTARA effectively balances task requirements with resource availability to guarantee that every resource is used to its maximum capacity. This results in minimized waste and maximized performance, making DPTARA the superior choice for environments like healthcare, where efficient resource utilization directly impacts service quality and outcomes. These results highlight DPTARA's greater efficiency in managing resources as demand increases.

4.5. Discussion

The results unequivocally demonstrate that DPTARA excels in managing resource-intensive and time-sensitive healthcare tasks, consistently outperforming traditional approaches and benchmark algorithms such as A-PBRA, CORA-GT, and NBIHA across key metrics of latency, task execution time, energy consumption, and resource utilization. In emergency scenarios with 60 requests, DPTARA achieves a 20-30% reduction in latency by prioritizing high-urgency tasks through dynamic priority adjustment, outperforming A-PBRA by 22%. Predictive load balancing prevents bottlenecks at edge nodes, ensuring steady performance even under high-traffic conditions. Compared to CORA-GT, DPTARA achieves a 62% reduction in energy consumption through adaptive energy management, dynamically scaling resource usage to optimize efficiency. Additionally, DPTARA outperforms NBIHA by demonstrating more stable resource utilization and significantly lower latency during high-demand scenarios, where NBIHA shows variability in performance. Furthermore, the framework's ability to maintain balanced resource utilization under varying traffic loads showcases its scalability and robustness. These findings affirm DPTARA as a sustainable and effective solution, addressing critical challenges of latency, energy efficiency, and resource optimization in real-world healthcare IoT systems.

5. Conclusion and future work

This research presents an advanced Dynamic Priority-Based Task Scheduling and Adaptive Resource Allocation system tailored for healthcare applications. By dynamically adjusting task priorities, employing predictive load balancing, and integrating adaptive energy management, our system ensures optimal performance of hospital workstations (HWs) acting as edge servers. The multi-objective optimization framework balances latency, throughput, and energy consumption, ensuring efficient and responsive task processing. The robust task offloading mechanism further enhances system efficiency by considering both latency and load on edge and cloud resources. The effectiveness of our approach in distributing workloads across healthcare facilities is demonstrated, offering significant improvements in task management, resource utilization, and energy efficiency. In the future, research will focus on creating strong security policies to protect patient data, integrating AI and machine learning for improved predictive skills, and putting real-time data analytics into practice for detailed system insights.

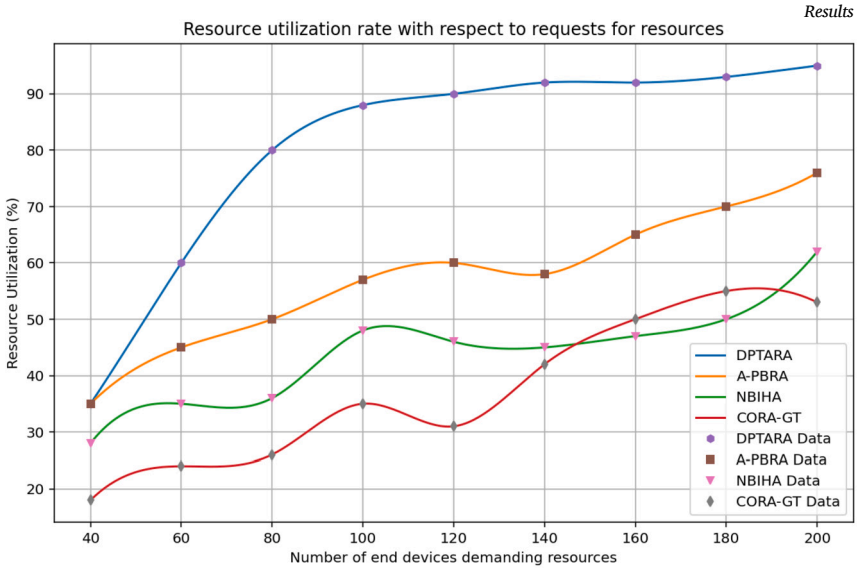

Fig. 9. Rate of resource utilization relative to resource requests.

CRediT authorship contribution statement

J. Anand: Writing – original draft, Visualization, Software, Methodology, Investigation, Formal analysis, Data curation, Conceptualization. **B. Karthikeyan:** Writing – review & editing, Validation, Supervision, Project administration, Investigation.

Supplementary material

Not applicable.

Funding statement

This research did not receive any specific grant from funding agencies in the public, commercial, or not-for-profit sectors.

Declaration of competing interest

The authors declare that they have no known competing financial interest or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

No data was used for the research described in the article.

References

[1] N. Hassan, S. Gillani, E. Ahmed, I. Yaqoob, M. Imran, The role of edge computing in internet of things, IEEE Commun. Mag. 56 (11) (2018) 110–115, https://doi.org/10.1109/MCOM.2018.1700906.
[2] M. Raeisi-Varzaneh, O. Dakkak, A. Habbal, B.-S. Kim, Resource scheduling in edge computing: architecture, taxonomy, open issues and future research directions, IEEE Access 11 (2023) 25329–25350, https://doi.org/10.1109/ACCESS.2023.3256522.
[3] P. Dong, et al., Edge computing based healthcare systems: enabling decentralized health monitoring in internet of medical things, IEEE Netw. 34 (5) (2020) 254–261, https://doi.org/10.1109/MNET.011.1900636.
[4] Guren Matsumura, et al., Real-time personal healthcare data analysis using edge computing for multimodal wearable sensors, Device (2024), https://doi.org/10.1016/j.device.2024.100597.
[5] Z. Sharif, L.T. Jung, M. Ayaz, M. Yahya, S. Pitafi, Priority-based task scheduling and resource allocation in edge computing for health monitoring system, J. King Saud Univ, Comput. Inf. Sci. 35 (2) (2023) 544–559, https://doi.org/10.1016/j.jksuci.2023.01.001.
[6] Z. Sharif, L.T. Jung, I. Razzak, M. Alazab, Adaptive and priority-based resource allocation for efficient resources utilization in mobile-edge computing, IEEE Internet Things J. 10 (4) (2023) 3079–3093, https://doi.org/10.1109/JIOT.2021.3111838.
[7] C. Jiang, et al., Energy Aware Edge Computing: A Survey, Elsevier Communications, 2020.
[8] Soule Issa Loutfi, et al., An overview of mobility awareness with mobile edge computing over 6g network: challenges and future research directions, Results Eng. 23 (2024) 102601, https://doi.org/10.1016/j.rineng.2024.102601.
[9] S. Hu, G. Li, Dynamic request scheduling optimization in mobile edge computing for iot applications, IEEE Internet Things J. 7 (2) (2020) 1426–1437, https://doi.org/10.1109/JIOT.2019.2955311.
[10] H. Rafique, M.A. Shah, S.U. Islam, T. Maqsood, S. Khan, C. Maple, A novel bio-inspired hybrid algorithm (nbiha) for efficient resource management in fog computing, IEEE Access 7 (2019) 115760–115773, https://doi.org/10.1109/ACCESS.2019.2924958.
[11] R. Alsurdeh, R.N. Calheiros, K.M. Matawie, B. Javadi, Hybrid workflow scheduling on edge cloud computing systems, IEEE Access 9 (2021) 134783–134799, https://doi.org/10.1109/ACCESS.2021.3116716.
[12] Z. Tong, X. Deng, F. Ye, S. Basodi, X. Xiao, Y. Pan, Adaptive computation offloading and resource allocation strategy in a mobile edge computing environment, Inf. Sci. 537 (2020) 116–131, https://doi.org/10.1016/j.ins.2020.05.057.
[13] Daniel Alfonso Verde Romero, et al., An open source iot edge-computing system for monitoring energy consumption in buildings, Results Eng. 21 (2024) 101875, https://doi.org/10.1016/j.rineng.2024.101875.
[14] A. Choudhury, M. Ghose, A. Islam, Yogita, Machine learning-based computation offloading in multi-access edge computing: a survey, Elsevier Syst. Archit. (2024), https://doi.org/10.1016/j.sysarc.2024.103090.
[15] J. Xu, Z. Hu, J. Zou, Computing offloading and resource allocation algorithm based on game theory for iot devices in mobile edge computing, Int. J. Innov. Comput. Inf. Control 16 (6) (2020) 1895–1914, https://doi.org/10.24507/ijicic.16.06.1895.
[16] J. Anand, B. Karthikeyan, Exploration of multi-task scheduling in multi-access edge computing, Eng. Proc. 62 (1) (2024), https://doi.org/10.3390/engproc2024062004.
[17] P. Gupta, A.V. Chouhan, M.A. Wajeed, S. Tiwari, A.S. Bist, S.C. Puri, Prediction of health monitoring with deep learning using edge computing, Meas. Sens. 25 (2023), https://doi.org/10.1016/j.measen.2022.100604.
[18] Edris Khezri, et al., Dljsf: data-locality aware job scheduling iot tasks in fog-cloud computing environments, Results Eng. 21 (2024) 101780, https://doi.org/10.1016/j.rineng.2024.101780.
[19] J. Liu, X. Liu, Energy-efficient allocation for multiple tasks in mobile edge computing, J. Cloud Comput. 11 (1) (2022) 1–14, https://doi.org/10.1186/s13677-022-00342-1.
[20] Y. Zhang, B. Tang, J. Luo, J. Zhang, Deadline-aware dynamic task scheduling in edge–cloud collaborative computing, Electronics (Switzerland) 11 (15) (2022), https://doi.org/10.3390/electronics11152464.
[21] Baoshan Lu, X.H.J.S. Junli Fang, Energy-efficient task scheduling for mobile edge computing with virtual machine i/o interference, Future Gener. Comput. Syst. 148 (2023) 538–549, https://doi.org/10.1016/j.future.2023.06.020.
[22] B. Zhang, Y. Li, S. Zhang, Y. Zhang, B. Zhu, An adaptive task migration scheduling approach for edge-cloud collaborative inference, Wirel. Commun. Mob. Comput. 2022 (2022), https://doi.org/10.1155/2022/8804530.
[23] A.S. Abohamama, A. El-Ghamry, E. Hamouda, Real-time task scheduling algorithm for iot-based applications in the cloud–fog environment, J. Netw. Syst. Manag. 30 (4) (2022), https://doi.org/10.1007/s10922-022-09664-6.
[24] M. Maray, J. Shuja, Computation offloading in mobile cloud computing and mobile edge computing: survey, taxonomy, and open issues, Mob. Inf. Syst. 2022 (2022), https://doi.org/10.1155/2022/1121822.

[25] Ali Raza, M.A.I.A. Li Jingzhao, Optimal load forecasting and scheduling strategies for smart homes peer-to-peer energy networks: a comprehensive survey with critical simulation analysis, Results Eng. 22 (2024) 102188, https://doi.org/10.1016/j.rineng.2024.102188.

[26] W. He, et al., Integrated resource allocation and task scheduling for full-duplex mobile edge computing, IEEE Trans. Veh. Technol. 71 (6) (2022) 6488–6502, https://doi.org/10.1109/TVT.2022.3163627.

[27] K. Kumaran, E. Sasikala, Learning based latency minimization techniques in mobile edge computing (mec) systems: a comprehensive survey, in: 2021 International Conference on System, Computation, Automation and Networking (ICSCAN), 2021.

[28] Z. Nan, L. Wenjing, L. Zhu, L. Zhi, L. Yumin, N. Nahar, A new task scheduling scheme based on genetic algorithm for edge computing, Comput. Mater. Continua 71 (1) (2022) 843–854, https://doi.org/10.32604/cmc.2022.017504.

[29] S. Lee, S.K. Lee, S.S. Lee, Deadline-aware task scheduling for iot applications in collaborative edge computing, IEEE Wirel. Commun. Lett. 10 (10) (2021) 2175–2179, https://doi.org/10.1109/LWC.2021.3095496.

[30] G. Li, Y. Yao, J. Wu, X. Liu, X. Sheng, Q. Lin, A new load balancing strategy by task allocation in edge computing based on intermediary nodes, EURASIP J. Wirel. Commun. Netw. 2020 (1) (2020), https://doi.org/10.1186/s13638-019-1624-9.