**Advances in Artificial Intelligence and Machine Learning**; Research (Ahead of Print) Received 25-06-2026; Accepted 08-11-2026; Published online 15-11-2026

# Queue-Aware Deep Reinforcement Learning for Heterogeneous Edge–Fog–Cloud Healthcare Monitoring

**Fawaz Alsolami** falsolami1@kau.edu.sa
*Department of Computer Science,*
*Faculty of Computing and Information Technology*
*King Abdulaziz University, Jeddah, Saudi Arabia*

**Ahmed A. A. Gad-Elrab** aaahmad4@kau.edu.sa
*Department of Computer Science,*
*Faculty of Computing and Information Technology*
*King Abdulaziz University, Jeddah, Saudi Arabia*

**Corresponding Author:** Ahmed A. A. Gad-Elrab.

**Copyright** © 2026 Fawaz Alsolami and Ahmed A. A. Gad-Elrab. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

## Abstract

In order to facilitate real-time healthcare monitoring and emergency response, physiological data generated by wearable devices and Internet of Things (IoT) sensors must be processed as quickly as possible. However, Edge–Fog–Cloud healthcare infrastructures have serious problems with latency, energy consumption, and resource usage due to rising traffic volumes and dynamic workload conditions. The inability of traditional static and threshold-based offloading techniques to adjust to shifting network conditions frequently leads to congestion, delayed alarms, and ineffective resource allocation. This research suggests a Queue-Aware Edge–Fog–Cloud framework that combines two intelligent routing techniques to overcome these issues. The first approach, QueueAware (QA), uses latency-aware heuristics and lightweight queue use to dynamically route healthcare jobs. The second strategy, DDQN-QueueAware (DDQNQA), combines queue-awareness with Double Deep Q-Learning to optimize discrete routing decisions based on queue conditions, latency, energy consumption, packet size, and system congestion. Numerous workload levels and queue-capacity combinations were used in extensive simulations. With maximum observed reductions of up to 40% in mean alert latency, up to 40% in p95/p99 latency, up to 64% in energy usage, and up to 46% in makespan relative to the evaluated baselines, DDQNQA achieved lower mean and tail latency in the majority of tested cases across the evaluated workload levels and queue-capacity configurations. Additionally, DDQNQA maintained fault-tolerance performance and competitive trust. These results show the possibility of integrating queue awareness and Double Deep Q-Learning for latency-sensitive healthcare routing under the simulated Edge-Fog-Cloud situations taken into consideration in this work.

**Keywords:** Distributed system, Fog computing; Edge computing, Remote healthcare monitoring, Deep Reinforcement Learning, and Network simulation.



**Citation:** Fawaz Alsolami and Ahmed A. A. Gad-Elrab. Queue-Aware Deep Reinforcement Learning for Heterogeneous Edge–Fog–Cloud Healthcare Monitoring. **Advances in Artificial Intelligence and Machine Learning**. 2026. [Ahead of Print]. https://dx.doi.org/10.54364/AAIML.2026.66357

# 1 INTRODUCTION

Modern healthcare systems have undergone a substantial transformation thanks to developments in wearable sensors, wireless communication, and Internet of Things (IoT) technology. Patients' physiological signals can now be collected in real time by continuous monitoring equipment such blood pressure sensors, glucose monitors, ECG sensors, and smart watches. Remote health monitoring systems that facilitate early disease identification, ongoing patient supervision, and quick reaction to life-threatening medical situations are made possible by these technologies. As a result, smart healthcare infrastructures and digital health ecosystems now heavily rely on remote healthcare monitoring [1–3]. All gathered sensor data is sent to distant servers for storage, processing, and decision-making in traditional healthcare monitoring architectures, which mostly rely on centralized cloud computing platforms. Large processing power, scalable storage, and sophisticated data analytics are all provided by cloud infrastructures [4, 5]. However, there are a number of difficulties when sending massive amounts of real-time physiological data straight to the cloud. Increased network latency, increased bandwidth usage, network congestion, and greater energy consumption at the sensor and gateway levels are some of these difficulties. Delays in processing or communication can have a detrimental effect on patient safety and reaction time in time-sensitive healthcare applications such as emergency alert detection [1, 6]. Edge computing and fog computing have become popular paradigms for moving processing closer to the data sources in order to overcome these constraints [7–9]. Data processing operations in edge computing are partially carried out at local gateways or edge devices that are situated close to the sensors. Fog computing extends this concept by introducing intermediate computing nodes between the edge layer and the cloud. These fog nodes reduce dependency on centralized cloud infrastructure by increasing processing, storage, and networking capacity. Therefore, in distributed healthcare monitoring systems, edge and fog designs may greatly lower communication latency, increase system scalability, and boost energy efficiency [8, 10–12]. Despite these benefits, improper workload distribution and congestion in intermediate processing steps continue to plague many current healthcare IoT solutions. Fog gateways and proxy nodes may encounter queue congestion and processing delays when several sensors provide physiological data and alarm signals at the same time. Dense IoT environments, such smart homes, senior care facilities, or large-scale healthcare monitoring deployments where hundreds of sensors regularly transmit medical data streams, exacerbate this issue. Thus, to ensure dependable system performance, effective queue management and flexible job allocation across compute tiers are required [13, 14]. Healthcare monitoring systems, in particular, need to make sure that important warning messages are routinely sent and swiftly processed, especially during times of high network traffic. Increased packet loss, delayed alarms, and decreased system reliability can be caused by poor queue management or work distribution.

This paper suggests a queue-aware distributed processing system intended for remote healthcare monitoring scenarios as a solution to these issues. Several computer layers, such as edge gateways, fog nodes, proxy servers, and cloud platforms, are coordinated by the suggested architecture. The solution lowers congestion, boosts processing efficiency, and enhances the dependability of alert transmission in healthcare IoT networks by tracking queue statuses and dynamically allocating workloads across several tiers. By splitting processing between edge and fog layers, the suggested method seeks to minimize congestion while achieving balanced alert delivery. For experimental evaluation, multiple system configurations are used, such as variable numbers of monitored households and queue sizes. Among the metrics used to evaluate performance are Alert Packet Delivery Ratio (AlertPDR), drop rate, processing time, communication overhead, and total energy usage.

The findings demonstrate that splitting processing tasks between edge and fog layers significantly improves system scalability and dependability when compared to conventional centralized cloud-based systems. Intelligent healthcare monitoring systems may now process patient data closer to the data source thanks to recent developments in edge-fog computing, which improves service responsiveness and reduces communication delays [15–17]. However, effective task scheduling and routing is a difficult topic due to the dynamic nature of healthcare workloads, fluctuating network conditions, and heterogeneous computing resources [13, 18, 19]. Without the need for predetermined scheduling rules, Deep Reinforcement Learning (DRL), in particular Double Deep Q-Network (DDPN) [20], has demonstrated considerable promise for learning adaptive decision policies in complicated situations. DDPG can optimize task offloading decisions based on real-time system states and resource availability by engaging with the environment continually. However, current DDPG-based methods frequently concentrate on general performance indicators and fail to sufficiently handle healthcare-specific needs such latency reliability guarantees, emergency-aware routing, queue congestion management, and varied queue capacities. These challenges motivate the development of more intelligent healthcare-aware DDPG frameworks for efficient task scheduling in edge–fog–cloud healthcare systems. The main contributions of this work are summarized as follows:

- A multi-layer Edge-Fog-Cloud healthcare monitoring system is created to enable distributed processing of physiological data produced by wearable and Internet of Things devices
- A lightweight QueueAware routing technique is suggested to lessen congestion by taking into account queue utilization and latency circumstances across gateway, fog, and cloud layers.
- A DDQN-based queue-aware routing algorithm is presented to learn adaptive offloading decisions based on queue utilization, delay, energy consumption, packet size, and healthcare alert priority.
- Real-world healthcare packet sizes and varied queue-capacity configurations are modeled to reflect different deployment dimensions and message severity levels.
- Extensive OMNeT++ simulations are conducted to evaluate latency, p95 and p99 tail latency, energy consumption, makespan, trust, fault tolerance, and communication overhead.

## 2 MULTI-LAYER ARCHITECTURE AND SYSTEM MODEL

The multi-layer edge-fog-cloud architecture of the suggested remote healthcare monitoring framework is intended to enable real-time analysis of physiological data produced by wearable medical devices. As shown in FIGURE 1, the design is composed of five primary layers: mobile sensor nodes, gateway nodes, fog nodes, proxy nodes, and cloud servers. To guarantee effective workload distribution, lower communication latency, and scalable healthcare data analysis, each layer carries out particular processing activities. In addition, the decision process is modeled as a Markov Decision Process (MDP), defined by the tuple $(S, A, R, P)$, where, **State** $S$**:** The state vector captures system conditions, $S = [q_g, q_f, q_c, d_g, d_f, d_c, \ell, e, p, a, s]$, where $q_g, q_f$ and $q_c$ represent queue utilization for gateway, fog, and cloud nodes, respectively, $d_g, d_f$ and $d_c$ denotes delay estimates for gateway, fog, and cloud nodes, respectively, $\ell$ is latency, $e$ is energy consumption, $p$ is packet priority, $a$ indicates alert status, and $s$ denotes the healthcare packet size, which varies according to the message type and severity level. **Action** $A$**:** The agent selects one of three routing decisions,

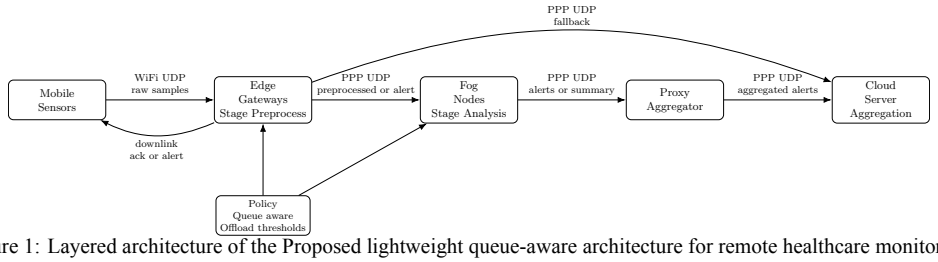



Figure 1: Layered architecture of the Proposed lightweight queue-aware architecture for remote healthcare monitoring.

$A = \{0 : \text{Gateway}, 1 : \text{Fog}, 2 : \text{Cloud}\}$. **Reward $R$:** The reward function balances performance objectives:

$$\begin{aligned} R_t =& \beta_1 \cdot Success - \beta_2 \frac{L}{L_{\max}} - \beta_3 \frac{E}{E_{\max}} - \beta_4 U_t - \beta_5 Drop \\ &+ \beta_6 \mathbb{I}(a = 1 \wedge L < L_{thr}) + C(A_t) - \beta_7 \mathbb{I}(A_t = 0 \wedge U_g > U_{thr}) \end{aligned} \tag{1}$$

where $Success$ indicates successful packet processing, $L$ is latency, $E$ is energy consumption, $U_t$ is the weighted queue utilization, $Drop$ indicates packet loss, $a$ is the alert flag, $L_{thr}$ is the healthcare alert deadline, and $C(A_t)$ is the action-dependent routing cost or reward. Where $\beta_1$ represents the success-reward coefficient, $\beta_2$ is the latency penalty coefficient, $\beta_3$ is the energy penalty coefficient, $\beta_4$ is the queue-utilization penalty coefficient, $\beta_5$ is the packet-drop penalty coefficient, $\beta_6$ is the emergency-priority reward coefficient, and $\beta_7$ is the gateway-congestion penalty coefficient. These coefficients determine the trade-off between latency minimization, energy efficiency, congestion control, reliability, and healthcare-priority awareness. **Transition $P$:** State transitions depend on network dynamics and workload variations.

## 3 RELATED WORK

The fast development of wearable medical devices and Internet of Things (IoT) technology has drawn a lot of research attention to remote healthcare monitoring systems. The goal of these systems is to enable early identification of possible health issues by continuously monitoring physiological signals [1–3]. Several computing paradigms, including cloud computing, fog computing, and edge computing, have been proposed to support healthcare data processing and analysis [6–8].

**Cloud-Based Healthcare Monitoring:** Centralized cloud infrastructures were the main source of data processing and storage for early remote healthcare monitoring systems. For healthcare analytics, cloud computing offers vast storage capacity and scalable processing resources. Cloud computing is an on-demand paradigm for network access to configurable computing resources including servers, storage, and applications, according to the National Institute of Standards and Technology (NIST) [4]. Cloud-based healthcare monitoring architectures enable centralized data analysis and large-scale medical record management. Baker et al. (2017) [1], discussed IoT-based smart healthcare systems that connect medical sensing technologies with communication and computing infrastructures, including cloud-based services. Similarly, big data analytics techniques have been widely used to process large volumes of data, including data generated by IoT applications [5]. Although cloud infrastructures provide powerful computing capabilities, several limitations exist in healthcare monitoring applications. High communication latency, increased bandwidth

consumption, and potential privacy risks may arise when large volumes of real-time physiological data are transmitted to distant cloud servers [1, 6].

**Fog-Based Healthcare Monitoring:** To overcome the limitations of centralized cloud architectures, fog computing has been introduced as an intermediate computing layer between edge devices and cloud servers. Low-latency data processing is made possible by fog computing, which distributes computational resources closer to the network edge. Bonomi et al. (2012) [8], first introduced the concept of fog computing to support Internet of Things applications. Later studies emphasized the importance of fog computing in enabling efficient IoT data processing [9, 12, 13]. Several fog-based healthcare monitoring frameworks have been proposed in the literature. Gia et al. [10], presented a fog computing architecture for healthcare IoT systems that processes physiological data and extracts local ECG features. A smart e-health gateway design that reduces communication overhead by performing edge-level data analytics was proposed by Rahmani et al. (2018) [11]. Fog computing for healthcare applications has been the subject of more recent research. Tuli et al. (2020) [21], proposed the HealthFog architecture, which integrates fog computing and ensemble deep learning for heart disease diagnosis in smart healthcare systems. Ahmadi et al. (2021) [22], and Kashyap et al. (2022) [14], provided systematic reviews of fog computing applications and challenges in healthcare monitoring. Dash et al. (2023) [23], also examined the application of edge and fog computing technologies in healthcare systems. When compared to cloud-only systems, fog-based architectures drastically lower communication latency. However, fog nodes still have limited computational resources, which may lead to processing difficulties under large workloads [13, 14].

**Edge and Smart-Edge Healthcare Systems:** By enabling data analysis right at the network edge, edge computing expands on the idea of distributed processing. Healthcare data can be processed locally by edge devices like gateways or smart sensors before being sent to higher layers. The architectures and difficulties of edge computing have been examined in a number of seminal publications. The significance of edge computing for latency-sensitive applications was emphasized by Shi et al. (2016) [7], and Satyanarayanan (2017) [24]. Mobile edge computing architectures and compute offloading strategies were examined by Mach and Becvar (2017) [25]. Varghese and Buyya (2016) [26], discussed opportunities and challenges in edge computing systems. Mao et al. (2017) [18], and Abbas et al. (2017) [19], further investigated mobile edge computing, communication requirements, resource management, and computation offloading strategies. Edge computing allows for quick anomaly identification and real-time physiological monitoring in healthcare applications. Amin and Hossain (2021) [6], reviewed edge intelligence and IoT technologies for healthcare applications, including the integration of artificial intelligence with distributed computing resources. Alshehri and Muhammad (2021) [2], presented a survey of IoT and AI-based smart healthcare systems. These studies investigated the use of edge intelligence techniques for smart healthcare monitoring. Despite the advantages of edge computing, edge devices often have limited computational capabilities and may struggle to handle complex analytics tasks [6, 7, 26].

**Hybrid Edge-Fog-Cloud Healthcare Architectures:** To overcome the limitations of individual computing paradigms, several studies have proposed hybrid architectures that combine edge, fog, and cloud resources. These architectures aim to distribute computation across multiple layers in order to improve system scalability and reduce communication latency. Daraghmi et al. (2022) [15], proposed an edge–fog–cloud computing hierarchy for remote health monitoring applications, considering performance and security requirements. Khanh et al. (2023) [16], presented an integrated edge–fog–cloud computing framework for Internet of Healthcare Things applications. AlQahtani

(2023) [17], investigated e-health service performance through the integration of 5G IoT, fog, and cloud computing. Recent studies have also investigated collaborative edge–cloud computing frameworks for healthcare analytics [2, 6]. These approaches distribute healthcare data processing across different computing layers to improve system responsiveness and resource utilization. Although hybrid architectures improve processing efficiency, most existing systems do not explicitly address queue management and workload distribution across multiple computing layers. Queue congestion may significantly degrade system performance, particularly in large-scale healthcare monitoring environments where multiple devices generate continuous data streams [13, 14].

Deep reinforcement learning has been widely used for task scheduling, offloading, and resource allocation in edge and fog computing environments. DQN-based methods are particularly suitable for discrete routing and scheduling problems because they estimate the long-term value of each possible action and select the action with the highest expected reward. However, standard DQN may suffer from overestimation bias, which can degrade routing stability under dynamic workloads. Double DQN addresses this issue by decoupling action selection from target-value estimation. Existing DRL-based healthcare scheduling methods, including the DDPG-based approach of Choppara and Lokesh (2025) [20], consider performance objectives such as makespan, energy consumption, trust, and fault tolerance, while giving limited attention to explicit queue congestion, emergency task priority, heterogeneous queue capacities, and tail-latency behavior. Therefore, this work proposes a queue-aware DDQN-based routing mechanism that explicitly incorporates queue utilization, latency, energy consumption, task size, and alert criticality into the routing decision.

Existing studies primarily focus on architectural design and distributed computing frameworks for healthcare monitoring. However, limited research has investigated queue-aware workload management in edge–fog–cloud healthcare systems. Most existing solutions assume unlimited processing capacity or do not consider dynamic queue behavior under varying workloads. To address these limitations, this paper proposes a queue-aware distributed processing framework that dynamically distributes healthcare data processing across edge gateways, fog nodes, proxy nodes, and cloud servers. The proposed approach explicitly models queue delays, workload distribution, and congestion control mechanisms in order to improve system scalability and reduce end-to-end processing latency. Although several DRL-based fog scheduling approaches have been proposed for healthcare systems, most existing methods primarily focus on makespan reduction, energy efficiency, and resource utilization while neglecting explicit queue congestion dynamics and tail-latency optimization. Furthermore, many existing studies assume homogeneous fog resources and do not consider heterogeneous queue capacities or healthcare-aware emergency routing under dynamic workload conditions. These limitations motivate the development of the proposed queue-aware DDQNQA routing framework. The introduction of queue-aware offloading as a novel idea is not what makes the current work novel. Instead, it resides in combining Double Deep Q-Learning (DDQN) for discrete healthcare routing across heterogeneous Edge—Fog—Cloud layers with specific queue-congestion information. When choosing between Gateway, Fog, and Cloud processing, the suggested DDQNQueueAware (DDQNQA) formulation considers queue utilization, latency, energy consumption, packet size, packet priority, and emergency-alert status. Additionally, the evaluation takes into account the average and tail-latency behavior under various workload levels and varied queue-capacity configurations.

# 4 PROPOSED QUEUE-AWARE PROCESSING FRAMEWORK AND METHODOLOGY

The proposed framework introduces a queue-aware distributed processing architecture for remote healthcare monitoring systems. The framework coordinates data processing across multiple computing layers including edge gateways, fog nodes, proxy nodes, and cloud servers. The objective of the proposed design is to minimize communication latency, reduce network traffic, and efficiently manage system workloads generated by continuous physiological monitoring.

## 4.1 Multi-Layer Data Processing Pipeline

The following three successive processing layers are used in the suggested structure to process healthcare data: (1) Edge Preprocessing Layer: At the network edge, gateway nodes carry out the initial processing stage. These nodes obtain physiological data from mobile sensors and carry out preliminary preprocessing tasks like validation, noise reduction, and data filtering. The processing time at the gateway node is defined as

$$T_{preprocess} = \frac{MI_{preprocess}}{MIPS_{gateway}} \tag{2}$$

where $MI_{preprocess}$ represents the number of million instructions required for preprocessing and $MIPS_{gateway}$ represents the processing capability of the gateway device. In addition to preparing the data for additional analysis, this preprocessing step lowers the volume of data sent to higher processing layers. (2) **Fog Analysis Layer:** The second processing stage occurs at fog nodes that provide intermediate computational resources between gateways and the cloud. Fog nodes perform deeper analysis on preprocessed healthcare data in order to detect abnormal physiological patterns or potential medical emergencies. The fog processing time is defined as

$$T_{fog} = \frac{MI_{analysis}}{MIPS_{fog}} \tag{3}$$

where $MI_{analysis}$ represents the computational workload required for data analysis and $MIPS_{fog}$ represents the processing capability of fog nodes. (3) **Cloud Processing Layer:** When more complex analysis or large-scale data aggregation is required, the processed healthcare data is forwarded to cloud servers. The cloud layer performs long-term storage, historical analysis, and large-scale medical data analytics. The cloud processing time is defined as

$$T_{cloud} = \frac{MI_{cloud}}{MIPS_{cloud}} \tag{4}$$

where $MI_{cloud}$ represents the computational workload executed in the cloud infrastructure.

## 4.2 Queue-Aware Processing Mechanism

Each processing layer maintains a queue that temporarily stores incoming healthcare data packets before they are processed. This queue-based mechanism allows the system to manage dynamic workloads generated by multiple monitoring devices. For packet $i$, the queue delay is defined as

$$Q_i = t_{dequeue,i} - t_{enqueue,i} \tag{5}$$

where $t_{enqueue,i}$ represents the time when the packet enters the queue and $t_{dequeue,i}$ represents the time when processing begins. The overall queue delay across the distributed system is computed as

$$Q_{total} = Q_{gateway} + Q_{fog} + Q_{cloud} \tag{6}$$

The queue capacity at each processing node is limited to a predefined size $Q_{max}$. When the queue becomes full, newly arriving packets are dropped to prevent system congestion. The queue drop rate is calculated as

$$DropRate = \frac{DroppedPackets}{TotalArrivals} \tag{7}$$

This metric indicates the severity of congestion within the system.

### 4.3 Queue-Aware Workload Offloading Strategy

The suggested system uses a queue-aware workload distribution method to avoid processing overload at edge devices. Every gateway node keeps an eye on its processing load and queue utilization. The gateway load is defined as

$$Load_{gateway} = \frac{QueueLength}{QueueCapacity} \tag{8}$$

Incoming healthcare data packets are transferred to fog nodes for additional processing if the gateway load surpasses a predetermined threshold value. In a similar vein, when their processing queues surpass a predetermined level, fog nodes may transfer processing work to cloud servers. A balanced workload distribution across several computing tiers is ensured by this hierarchical offloading technique.

### 4.4 Queue-Aware Congestion Modeling

In the enhanced framework, each fog node maintains a finite queue for incoming healthcare tasks. The queue temporarily stores tasks before execution when the node is occupied. Unlike traditional scheduling methods that only consider CPU, RAM, storage, and VM availability, the proposed framework additionally models queue congestion dynamics. This queue-aware formulation is used by the suggested DDQNQA scheduler to prevent emergency healthcare jobs from being assigned to crowded Fog nodes, even when computational resources are still available. The queue usage is defined as follows for each fog node $i$:

$$U_i = \frac{Q_i}{Q_i^{max}} \tag{9}$$

where, the current queue length of fog node $i$ is represented by $Q_i$, the maximum queue capacity is denoted by $Q_i^{max}$, and the normalized queue utilization is indicated by $U_i$. A fog node deemed congested if $U_i \geq \theta_q$ is satisfied, where the congestion threshold is denoted by $\theta_q$. At fog node $i$, the queue waiting latency for task $j$ is computed as follows:

$$D_q(i,j) = t_{start}(i,j) - t_{arrival}(i,j) \tag{10}$$

where, the arrival time of task $j$ is indicated by $t_{arrival}(i, j)$ and the task execution start time is indicated by $t_{start}(i, j)$.

The computation of the overall response time is calculated as follows:

$$T_{response} = T_{transmission} + T_{queue} + T_{execution} \tag{11}$$

The DDPG [20], scheduler may avoid allocating emergency healthcare jobs to congested fog nodes, even if those nodes still have computing resources available, thanks to this queue-aware formulation.

**Heterogeneous Queue Capacity Modeling:** Heterogeneous queue capacities are allocated to fog nodes based on their computational capabilities in order to simulate genuine fog computing environments. The fog node *i*'s maximum queue capacity is calculated as follows:

$$Q_i^{max} = Q_{base}\left(\alpha\frac{CPU_i}{CPU_{max}} + \beta\frac{RAM_i}{RAM_{max}} + \gamma\frac{VM_i}{VM_{max}}\right) \tag{12}$$

where, the baseline queue capacity is denoted by $Q_{base}$, the CPU capability of fog node *i* is denoted by $CPU_i$, the RAM capacity of fog node *i* is denoted by $RAM_i$, the number of available virtual machines is denoted by $VM_i$, and the weighting coefficients are $\alpha$, $\beta$, and $\gamma$, subject to $\alpha+\beta+\gamma = 1$. In this work, the weighting factors are selected as $\alpha = 0.5, \quad \beta = 0.3, \quad \gamma = 0.2$. When determining queue capacities, this configuration gives priority to CPU capabilities while taking memory and virtualization resources into account. The processing-time formulation, in which Gateway, Fog, and Cloud execution times rely on their respective MIPS capabilities, is used to model computational heterogeneity independently of queue capacity. Furthermore, CPU capabilities, RAM capacity, and virtual machine availability are all included in the heterogeneous queue-capacity model. Queue capacity is changed while the other computation parameters remain constant in the described sensitivity studies. Therefore, rather than being a comprehensive CPU-capacity scaling analysis, the provided results should be largely viewed as a queue-capacity sensitivity research.

**Algorithm 1** DDQNQA Decision Process

**Require:** Current queue states $q_g$, $q_f$, $q_c$; queue capacities $Q_g$, $Q_f$, $Q_c$; delay estimates $d_g$, $d_f$, $d_c$; latency $L$; energy consumption $E$; packet priority $p$; alert flag $a$; packet size $s$

**Require:** Online Q-network $Q_\theta$, target Q-network $Q_{\theta^-}$, replay buffer $\mathcal{D}$, exploration rate $\epsilon$

**Ensure:** Routing action $A_t \in \{0, 1, 2\}$, where 0 = Gateway, 1 = Fog, and 2 = Cloud

Normalize queue utilization:

$$U_g = \frac{q_g}{Q_g}, \quad U_f = \frac{q_f}{Q_f}, \quad U_c = \frac{q_c}{Q_c}$$

Construct current state:

$$S_t = [U_g, U_f, U_c, d_g, d_f, d_c, L, E, p, a, s]$$

Select action using $\epsilon$-greedy policy:

$$A_t = \begin{cases} \text{random action}, & \text{if } rand < \epsilon \\ \arg\max\limits_{x \in \{0,1,2\}} Q_\theta(S_t, x), & \text{otherwise} \end{cases}$$

**if** $A_t = 0$ **then**
    Process packet at the Gateway
**else if** $A_t = 1$ **then**
    Offload packet to the Fog node
**else if** $A_t = 2$ **then**
    Escalate packet to the Cloud
**end if**
Observe next state $S_{t+1}$ after executing action $A_t$
Compute reward:

$$\begin{aligned} R_t = {} & \beta_1 \cdot Success - \beta_2 \cdot \frac{L}{L_{\max}} - \beta_3 \cdot \frac{E}{E_{\max}} - \beta_4 \cdot U_t \\ & - \beta_5 \cdot Drop + \beta_6 \cdot \mathbb{I}(a = 1 \wedge L < L_{thr}) \\ & + C(A_t) - \beta_7 \cdot \mathbb{I}(A_t = A_{edge} \wedge U_g > U_{thr}) \end{aligned}$$

Store transition in replay buffer:

$$\mathcal{D} \leftarrow \mathcal{D} \cup \{(S_t, A_t, R_t, S_{t+1})\}$$

**if** training step is triggered **then**
    Sample mini-batch $\mathcal{B}$ from replay buffer $\mathcal{D}$
    **for** each transition $(S_i, A_i, R_i, S_{i+1}) \in \mathcal{B}$ **do**
        Select next action using the online network:

$$A_i^{next} = \arg\max_{x \in \{0,1,2\}} Q_\theta(S_{i+1}, x)$$

        Compute DDQN target value:

$$y_i = R_i + \gamma Q_{\theta^-}(S_{i+1}, A_i^{next})$$

        Compute predicted value:

$$\hat{y}_i = Q_\theta(S_i, A_i)$$

    **end for**
    Update online Q-network by minimizing Huber loss:

$$\mathcal{L} = SmoothL1Loss(\hat{y}_i, y_i)$$

    Apply gradient clipping to stabilize training
**end if**
**if** target update interval is reached **then**
    Update target network:

$$\theta^- \leftarrow \theta$$

**end if**
Decay exploration rate:

$$\epsilon \leftarrow \max(\epsilon_{min}, \epsilon \cdot \epsilon_{decay})$$

**return** $A_t$

**Communication Model:** System components can communicate with each other via both wired and wireless communication methods, where mobile-to-gateway communication is done using wireless links, gateway-to-fog communication, wired network links are utilized, and fog-to-cloud communication is made easier via fast backbone networks. The transmission delay between two nodes is defined as

$$T_{transmission} = \frac{PacketSize}{Bandwidth} \tag{13}$$

**End-to-End Processing Model:** The total processing time of a healthcare data packet is defined as

$$T_{total} = T_{transmission} + T_{queue} + T_{processing} \tag{14}$$

where $T_{processing}$ stands for computational delay, $T_{queue}$ for queue waiting time, and $T_{transmission}$ for communication delay. The suggested framework may be quantitatively evaluated under various network demands and system settings thanks to this mathematical model.

### 4.5 QueueAware (QA) Algorithm

The queue-aware healthcare processing mechanism begins when physiological measurements are collected by IoT sensor devices and transmitted to the gateway layer. Upon arrival, the gateway checks the availability of its local queue before accepting incoming packets. If sufficient queue space exists, the packet is enqueued and preliminary preprocessing operations are performed to filter, organize, and prepare the healthcare data for further analysis. However, when the gateway queue reaches its capacity limit, newly arriving packets are discarded to prevent excessive congestion and resource exhaustion. After preprocessing, the gateway evaluates the data to identify abnormal physiological patterns that may indicate potential health risks. FIGURE 1 illustrates the proposed queue-aware algorithm used for healthcare data processing. When an abnormal condition is detected, an alert message is forwarded to the fog layer for real-time analysis. Similar to the gateway, the fog node first verifies queue availability before accepting the alert for processing. The fog layer performs latency-sensitive analytics and generates rapid responses for healthcare monitoring applications. If the analysis requires more computational resources, long-term historical data, or advanced machine learning models, the healthcare data is further transmitted to the cloud layer. The cloud infrastructure then performs comprehensive global analytics, large-scale data processing, and long-term healthcare assessment, enabling informed decision-making while maintaining efficient utilization of Edge–Fog–Cloud resources.

### 4.6 DDQNQA Algorithm

A Double Deep Q-Network agent is used in the suggested DDQNQA decision process to choose the best processing layer for every medical packet. The three options in the action space are gateway processing, fog offloading, and cloud escalation because the routing decision is discrete. In addition to delay estimations, energy consumption indicators, packet priority, and emergency status, the agent monitors the usage levels of the gateway, fog, and cloud queues at each decision step. First, queue utilities are normalized to show each layer's current resource occupancy. The DRL agent's state vector is composed of these normalized queue states, energy, delay, and application-specific characteristics. After then, a $\epsilon$-greedy strategy is used to strike a balance between exploration and exploitation, enabling the agent to either choose the action with the largest anticipated long-term

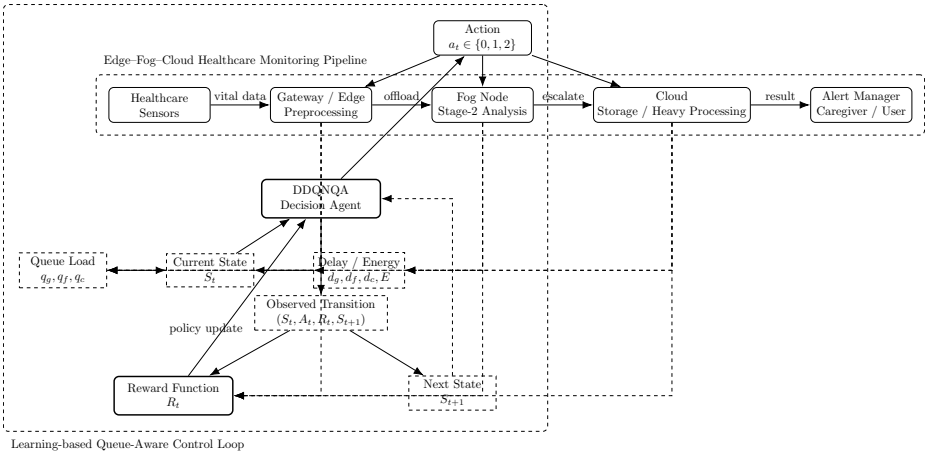



Figure 2: The proposed DDQNQA architecture for remote healthcare monitoring.

return or investigate alternate routing options. The packet is processed locally at the gateway, offloaded to a fog node, or escalated to the cloud for additional processing, depending on the chosen action. Following the routing decision's execution, the agent evaluates the system status and calculates a reward that takes into account certain Quality of Service (QoS) goals. While reducing latency, energy consumption, queue congestion, and packet losses, the incentive function promotes successful packet processing. While packets that are routed to excessively laden edge resources are penalized, additional incentives are offered for prompt handling of emergency healthcare signals. For experience replay, the observed transition—which includes the current state, chosen action, reward, and subsequent state—is kept in a replay buffer. Double Deep Q-Learning is used to update the policy network during training by sampling mini-batches of previous experiences. To minimize overestimation bias, target values are calculated using both the policy and target networks. The optimization process minimizes the Huber loss and employs gradient clipping to improve training stability. Periodically, the target network parameters are synchronized with the policy network, while the exploration rate gradually decays to shift the learning process from exploration toward exploitation. Through continuous interaction with the environment, the DDQNQA agent learns an adaptive routing policy that balances resource utilization, latency, energy efficiency, and healthcare service reliability across the Edge–Fog–Cloud infrastructure. Fig. 2 illustrates the proposed RL architecture used for healthcare data processing. DDQNQA does not alter the basic DDQN learning rule or its convergence features; rather, it maintains the usual Double Deep Q-Learning update method. The contribution is found in congestion modeling, discrete routing decision-making across the Gateway, Fog, and Cloud layers, queue-aware reward formulation, and healthcare-specific state representation. Algorithm 1 replaces fixed threshold-based routing with a learned queue-aware policy that adapts routing decisions according to system congestion, task urgency, latency, and energy conditions.

The modified reward function is defined in equation 1. The healthcare task-scheduling problem examined in [20], where DDPG is used, serves as the inspiration for the suggested DDQNQA framework. However, in the current study, the routing problem is formulated as a discrete-action Double Deep Q-Learning problem and expanded with tail-latency, healthcare-task urgency, and explicit queue-congestion information. In addition to CPU, RAM, storage, and VM availability, the scheduler observes queue length, queue utilization, queue waiting delay, and healthcare task urgency. This enhancement allows the DDQNQA agent to distinguish between resource-available

fog nodes and truly responsive fog nodes. A fog node may have sufficient CPU and memory resources but still experience severe congestion due to overloaded waiting queues. Therefore, the proposed queue-aware DDQNQA scheduler dynamically avoids overloaded nodes and prioritizes emergency healthcare tasks using latency-sensitive and deadline-aware routing policies. Unlike conventional DRL-based fog scheduling approaches that mainly optimize makespan and resource utilization, the enhanced framework explicitly models queue congestion dynamics and tail-latency behavior. This enhancement is particularly important for healthcare applications because average latency alone cannot accurately represent dangerous worst-case delays experienced by emergency medical tasks.

The $\epsilon$-greedy approach facilitates exploration during learning in the simulated environment and should not be confused with the random routing of real patient alarms upon deployment. In a realworld healthcare implementation, the trained policy should operate in exploitation mode ( = 0) for critical traffic, applying deterministic safety constraints or fallback routing before making a decision

## 5 SIMULATION AND RESULTS

The proposed system is implemented and evaluated using the OMNeT++ simulation framework [27]. Each simulation scenario runs for 1200 seconds and evaluates different workload distribution policies. Using network settings with 15, 25, and 50 mobile nodes, 5 gateway nodes, 5 fog nodes, one proxy node, and one cloud server made up the infrastructure. A 100 Mbps bandwidth was set up for each communication link. In addition, TABLE 1 outlines the reinforcement learning configuration used in the DDQNQA algorithm. TABLE 2 outlines the queue-capacity settings that were applied during the trials. To represent various clinical circumstances, the simulation took into account four different healthcare communication types with differing packet sizes. While moderate-condition signals, which contained extra contextual information, used 512-byte packets, normal vital signs were sent as periodic sensor readings of 256 bytes. 1024-byte packets were used to simulate emergency alerts, whereas 4096-byte packets were used to model urgent emergency alerts with an ECG segment attachment. The values were selected to represent increasing communication demands from routine monitoring to critical emergency reporting while maintaining realistic data characteristics for healthcare IoT applications. In this study, the healthcare packet model is not a clinical diagnostic dataset, but rather an abstraction at the communication level. Rather than disease categorization or diagnostic accuracy, the goal is to assess routing, queuing, latency, energy use, and delivery behavior. As a result, the chosen packet sizes reflect growing communication payloads related to normal, moderate, emergency, and ECG-enhanced alert signals. Future research will use trace-driven physiological data for validation. For all reported workload levels and queue-capacity configurations, the DDQNQA hyperparameters shown in TABLE 1 remained constant.

The reward coefficients were fixed according to the implementation as $\beta_1 = 2.0$, $\beta_2 = 1.0$, $\beta_3 = 1.0$, $\beta_4 = 1.0$, $\beta_5 = 5.0$, $\beta_6 = 3.0$, and $\beta_7 = 0.6$. Latency and energy are normalized using $L_{\max} = 10000$ ms and $E_{\max} = 10$, respectively. The weighted queue-utilization term is defined as $U_t = 0.5U_g + 0.3U_f + 0.2U_c$, assigning greater importance to congestion at the Gateway layer. The Gateway congestion threshold is $U_{\text{thr}} = 0.8$. The action-dependent routing term is $C(A_t) = 0.3$ for Gateway processing, $0.1$ for Fog offloading, and $-0.15$ for Cloud offloading. A sensitivity study of congestion under rising resource availability is provided by the four queue-capacity configurations. Workload intensity is adjusted utilizing 15, 25, and 50 monitored nodes, while gateway, fog, and

Table 1: DDQNQA Configuration Parameters

| Parameter | Value | Parameter | Value |
|---|---|---|---|
| Learning Rate ($\alpha$) | 0.001 | Success Reward Coefficient ($\beta_1$) | 2.0 |
| Discount Factor ($\gamma$) | 0.99 | Latency Penalty Coefficient ($\beta_2$) | 1.0 |
| Replay Buffer Size | 10,000 | Energy Penalty Coefficient ($\beta_3$) | 1.0 |
| Mini-Batch Size | 64 | Queue Utilization Penalty Coefficient ($\beta_4$) | 1.0 |
| Target Network Update Interval | 200 steps | Packet Drop Penalty Coefficient ($\beta_5$) | 5.0 |
| Initial Exploration Rate ($\epsilon$) | 1.0 | Emergency Reward Coefficient ($\beta_6$) | 3.0 |
| Minimum Exploration Rate ($\epsilon_{min}$) | 0.05 | Gateway Congestion Penalty Coefficient ($\beta_7$) | 0.6 |
| Exploration Decay ($\epsilon_{decay}$) | 0.995 | | |
| State Dimension | 11 | | |
| Action Dimension | 3 | | |
| Hidden Layers | 128, 128 | | |
| Activation Function | ReLU | | |
| Loss Function | Huber Loss | | |
| Optimizer | Adam | | |

Table 2: Configurations of queue capacity utilized in the tests.

| Scenario | Gateway Queue | Fog Queue | Cloud Queue |
|---|---|---|---|
| Small | 25 | 75 | 200 |
| Medium | 50 | 150 | 400 |
| Large | 75 | 225 | 600 |
| Extra Large | 100 | 300 | 800 |

cloud queue capacity are gradually increased from the Small to Extra-Large configurations. From normal physiological measures to emergency communications with extra contextual or ECG data, the four healthcare packet sizes reflect rising communication payloads. Instead of representing a clinical dataset, these packet sizes are communication-level modeling assumptions that are used to assess the impact of varied message sizes on routing, congestion, latency, and energy usage.

### 5.1 Compared Methods

The proposed QA and DDQNQA algorithms are evaluated against the following baseline approaches: **Cloud-Based Offloading (CB)**: Every piece of data is sent straight to the cloud for processing. Although this method has a lot of processing power, it also adds a lot of network overhead and communication delays, which raises latency and uses more energy, especially in latency-sensitive applications [1, 4–6]. **Fog-based Offloading (FB)**: There is no offloading to higher levels; all incoming data is processed at the fog node. Although this method reduces communication overhead, it has a limited processing capability, which increases latency and may cause congestion when there is a heavy workload [8–11, 13, 22]. **Double Deep Q-Network (DDPG) [20]**: The DDPG method is included as a learning-based healthcare scheduling baseline. It is treated separately from the proposed DDQNQA method, which uses Double Deep Q-Learning for discrete Gateway–Fog–Cloud routing. The DDPG baseline does not include the complete queue-aware state and healthcare-specific routing formulation proposed in DDQNQA.

This baseline techniques are not restricted to antiquated or traditional procedures. Specifically, the DDPG-based approach used from [20], is a recent 2025 study that uses deep reinforcement learning to handle job scheduling and load balancing for healthcare applications. This approach was chosen due to its close relation to the Edge-Fog scheduling problem in healthcare that this paper examines. Cloud-based processing, fog-based processing, and lightweight queue-aware routing are represented

by the remaining baselines. Therefore, rather than offering a comprehensive benchmark of all DRL algorithms, the current study compares DDQNQA with sample conventional, queue-aware, and recent healthcare-oriented DRL techniques. Future research will take into account a more comprehensive comparison with PPO, SAC, and other contemporary DRL techniques. A component-level perspective is also offered by the current comparison. While DDPG is a learning-based healthcare scheduling baseline without the full DDQNQA queue-aware discrete-routing formulation, QA represents explicit queue-aware routing without deep reinforcement learning. Double Deep Q-Learning and queue awareness are combined in DDQNQA. Therefore, comparing QA, DDPG, and DDQNQA helps show how queue information and learning-based adaptive routing contribute differently. Future research will focus on a specific ablation employing different DQN/DDQN variants.

### 5.2 Performance Metrics

The proposed healthcare monitoring framework is evaluated using several performance metrics that measure communication efficiency, processing performance, and system reliability. These metrics allow a comprehensive comparison between the proposed architecture and baseline architectures.

- **End-to-End Alert Latency:** End-to-end alert latency measures the total time required for a medical event generated at the mobile sensor to reach the final processing layer or alert management system. This metric is critical in healthcare applications because delayed alerts may negatively affect patient safety. The latency is calculated as

$$Latency_i = t_{receive,i} - t_{send,i} \tag{15}$$

  where, $t_{send,i}$ represents the time when alert message $i$ is generated by the mobile sensor and $t_{receive,i}$ represents the time when the alert is received by the final processing node. To capture latency variability, additional percentile metrics are also computed which are 95th percentile latency (P95) and 99th percentile latency (P99). These values represent the latency below which 95% and 99% of alerts are delivered.

- **Alert Packet Delivery Ratio (AlertPDR):** Alert Packet Delivery Ratio (AlertPDR) measures the reliability of the healthcare monitoring system in delivering critical medical alert packets generated by sensors or edge nodes to higher processing layers such as fog or cloud servers.

$$AlertPDR = \frac{AlertPackets_{received}}{AlertPackets_{sent}} \tag{16}$$

  where, $AlertPackets_{received}$ represents the number of alert packets successfully received by the destination node and $AlertPackets_{sent}$ represents the total number of alert packets generated by sensor or gateway nodes. A higher AlertPDR indicates that the system reliably delivers critical health alerts with minimal packet loss while maintaining balanced alert delivery with reduced congestion due to distributed processing across edge and fog layers. This behavior is essential in remote healthcare monitoring systems where timely and reliable alert delivery directly supports rapid medical response and improves patient safety.

- **Energy Consumption:** Energy consumption evaluates the computational energy used by the system nodes during healthcare data processing. The total energy consumption is calculated

as

$$Energy = \sum_{i=1}^{N} (P_i \times T_i) \tag{17}$$

where, $P_i$ is the power consumption of node $i$ and $T_i$ is the processing time of node $i$. This metric is particularly important for evaluating the efficiency of edge and fog nodes that may operate under limited energy budgets.

- **Makespan:** In fog computing, makespan—defined as the entire time required to do all tasks—is a crucial measure, particularly for healthcare applications where prompt processing is crucial. Makespan can be minimized by dynamically allocating tasks to fog nodes based on their real-time resource availability, such as CPU, RAM, and storage capacity.

$$Makespan = \max_{i} (Finish\ Time\ of\ Task_i) \tag{18}$$

- **Trust:** Trust in fog computing refers to the reliability of nodes to execute tasks accurately and securely. By prioritizing task allocation to nodes with higher historical success rates and lower failure occurrences will improve the trust. While fog nodes inherently decentralize processing and increase system reliability, DRL further enhances trust by dynamically evaluating and selecting nodes based on their reliability and performance. a higher trust value indicates better system reliability.

$$Trust = \frac{SuccessfulTasks}{TotalTasks} \tag{19}$$

- **Fault Tolerance:** Fault tolerance is critical in ensuring system reliability, especially in healthcare applications where task interruptions can have severe consequences. Lower fault tolerance values indicate fewer rerouted tasks and better system stability. While fog nodes inherently improve fault tolerance through decentralized processing.

$$FaultTolerance = \frac{ReroutedTasks}{TotalTasks} \tag{20}$$

### 5.3 Results and Discussion

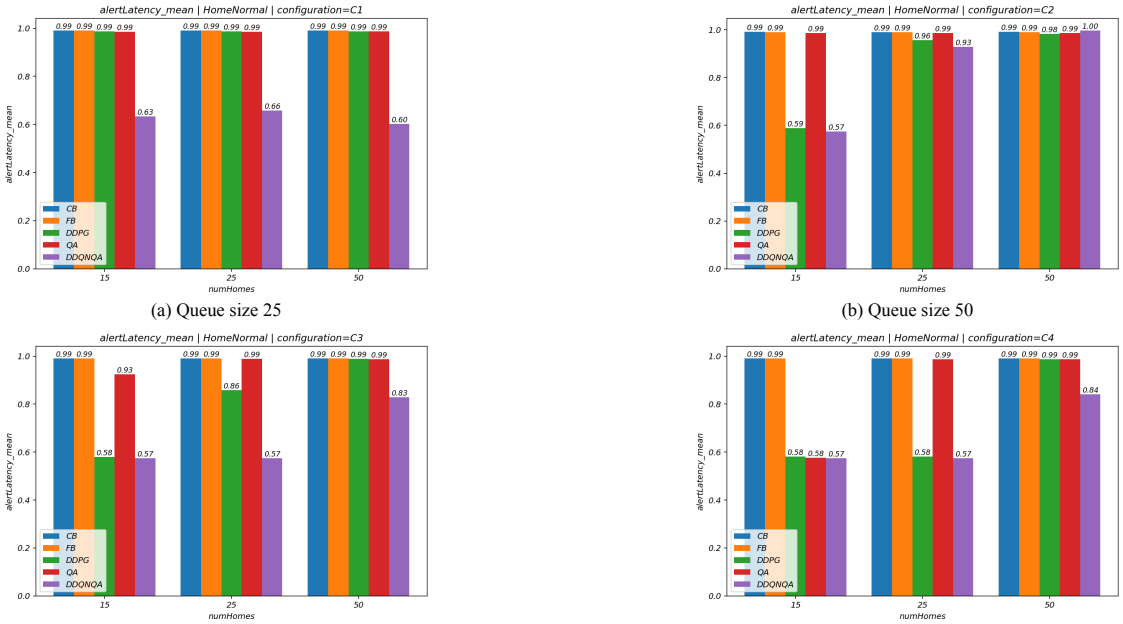
The observed outputs of the configured OMNeT++ simulation scenarios are represented by the reported results. The average and tail-latency behavior is described by mean, p95, and p99 latency. Since hypothesis testing and repeated-seed confidence intervals were not included in the initial experimental design, no statistical significance is claimed for the reported changes. Future study will involve statistical validation and repeated independent experiments.

#### 5.3.1 Alert latency analysis

FIGURES 3 - 5 present the alert latency results under the HomeNormal scenario for different queue sizes and numbers of monitored homes. The performance of the proposed architecture is compared with three baseline systems, namely CB, FB, and DDPG.

- **Mean Alert Latency:** The four figures show that under various queue capacities and workload levels, the suggested QA and DDQNQA approaches outperform the baseline methods in terms

of alert latency. Across all setups, CB and FB continuously show the greatest alert latency, staying near 0.99, suggesting poor flexibility in response to shifting network and workload conditions. Although the typical DDPG technique lowers latency in some configurations, its performance deteriorates when the number of homes and queue parameters change, leading to unstable behavior. The effectiveness of integrating queue-awareness into routing decisions is demonstrated by the suggested QA technique, which, in contrast, maintains low and stable alert latency throughout the majority of circumstances. Additionally, compared to CB and FB, the suggested DDQNQA technique provides the lowest alert latency in the majority of setups, especially in crowded environments with small and medium queue capacities, where latency savings of roughly 35%–45% are noted. These findings demonstrate that integrating deep reinforcement learning with queue-aware decision-making increases scalability under growing system workloads, lowers reaction times, and allows for more effective handling of healthcare alarms. All things considered, DDQNQA is the best option for reducing alert latency, but QA is a reliable and lightweight substitute that regularly beats conventional routing and scheduling techniques.

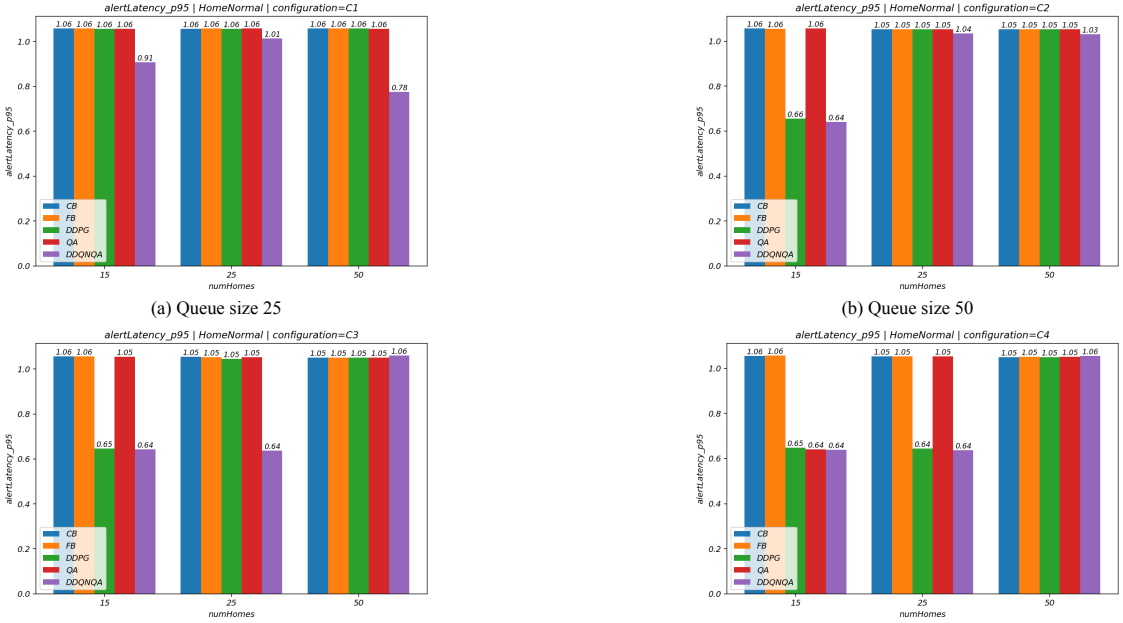



(a) Queue size 25 (b) Queue size 50

(c) Queue size 75 (d) Queue size 100

Figure 3: Mean alert latency under the HomeNormal scenario for different queue sizes and numbers of monitored homes.

- **95th Percentile Alert Latency:** The findings of the p95 alert latency further show how well the suggested DDQNQA approach works to increase the dependability of healthcare alert delivery under different workload and queue configurations. The suggested DDQNQA greatly lowers the tail latency in the majority of cases, whereas CB, FB, and QA consistently show p95 latencies of 1.05–1.06 seconds, suggesting that 95% of alert messages incur very large response delays. When compared to CB, FB, and QA, DDQNQA achieves p95 latencies as low as 0.64–1.04 seconds for queue sizes of 25, 50, 75, and 100. This represents reductions of up to almost 40%. Although the traditional DDPG method similarly reduces p95 latency in a number of settings, as the number of households and queue capacities rise, its performance becomes less consistent. DDQNQA, on the other hand, continuously maintains lower tail latency across various workload levels, indicating its capacity to efficiently prioritize impor-




(a) Queue size 25

(b) Queue size 50

(c) Queue size 75

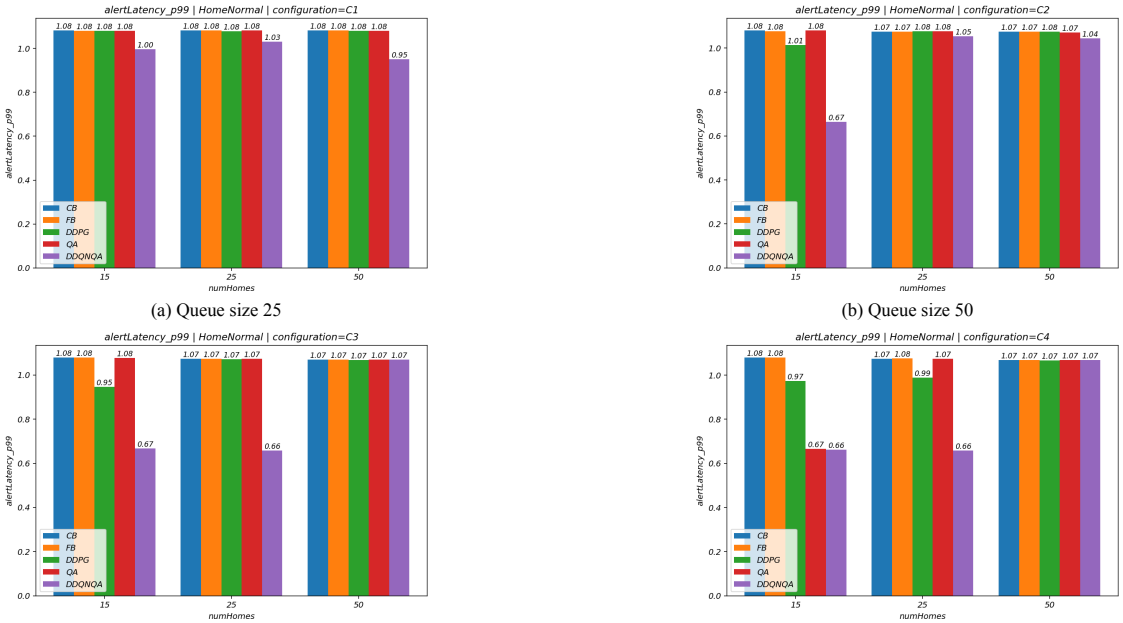
(d) Queue size 100

Figure 4: 95th percentile alert latency comparison for different queue sizes and system loads.

tant healthcare warnings and prevent delays brought on by congestion. Since p95 delay is the worst-case performance experienced by most users, the observed reductions confirm that integrating queue-awareness with deep reinforcement learning significantly improves service dependability and responsiveness for latency-sensitive healthcare applications. DDQNQA is the most dependable approach overall for reducing high-latency events, even though QA by itself does not increase tail-latency reduction over the baseline techniques.

- **99th Percentile Alert Latency:** By evaluating the latency of the poorest 1% of warning signals, the p99 alert latency data enable a more rigorous assessment of system performance. The results indicate the increased capabilities of the suggested DDQNQA technique to reduce excessive delay occurrences under high-load conditions, as it consistently produces the lowest p99 latency values across the majority of configurations. Conversely, CB, FB, and QA strategies usually maintain p99 latencies between 1.07 and 1.08 seconds, suggesting a limited capacity to manage tail-latency circumstances. The conventional DDPG technique reduces p99 latency in many scenarios, particularly for lesser workloads; nevertheless, its performance becomes less reliable as the number of residences and queue capacities change. When compared to CB, FB, and QA, the suggested DDQNQA technique considerably reduces p99 latency, achieving values as low as 0.66–0.67 seconds in a number of circumstances. This results in latency reductions of about 38% to 40%. These improvements are particularly important for healthcare monitoring applications since p99 latency is the worst-case response time experienced by critical alerts. Even in the face of heavy workloads and traffic bottlenecks, emergency medical communications are dispatched promptly thanks to a reduced p99 latency. Overall, the results demonstrate that queue-awareness and DDPG together enable much greater dependability for latency-sensitive medical services, improved urgent healthcare traffic prioritization, and more effective congestion management. The strong p99 performance

of DDQNQA attests to its effectiveness in reducing tail-latency events and ensuring the timely transmission of critical healthcare alerts in a range of network scenarios.




(a) Queue size 25

(b) Queue size 50

(c) Queue size 75

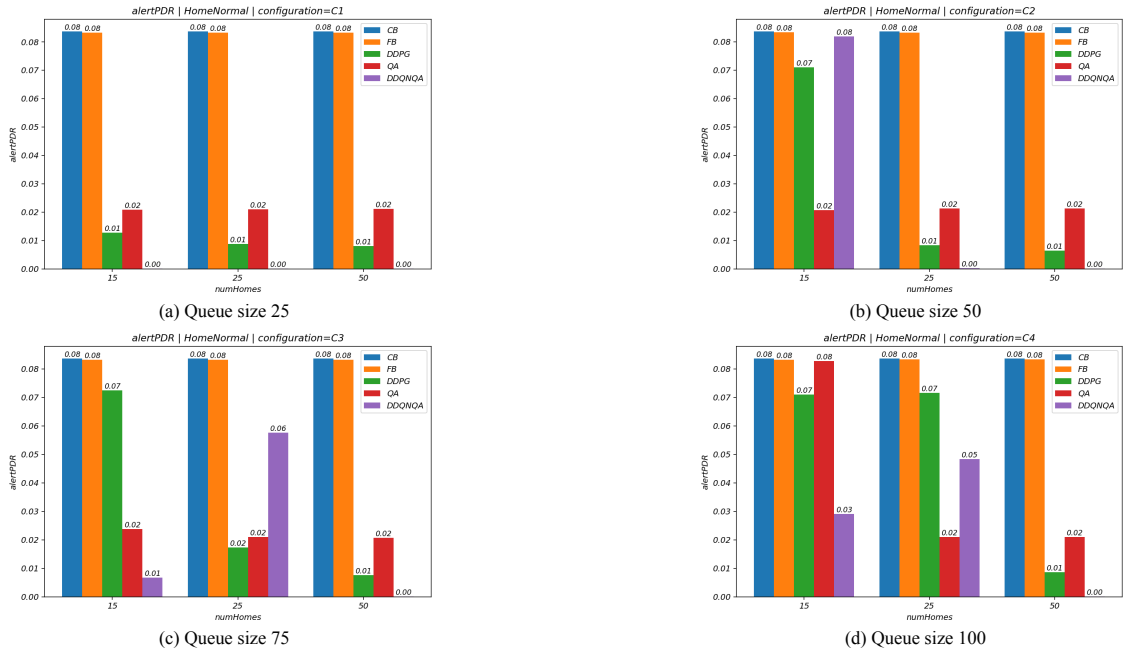
(d) Queue size 100

Figure 5: 99th percentile alert latency showing worst-case delay behavior under different queue capacities.

### 5.3.2 Alert packet delivery ratio (AlertPDR)

FIGURE 6 presents the alert packet delivery ratio for different architectures under increasing system load and queue sizes. The Alert Packet Delivery Ratio (Alert PDR) results examine each routing strategy's capacity to effectively convey emergency medical alerts. Since alert messages are suggestive of important medical events, a higher Alert PDR indicates more dependability and better quality of service for patient monitoring applications. The results show that CB and FB consistently achieve the greatest Alert PDR values, maintaining approximately 0.084 in nearly all cases, across all queue sizes and workload levels. This illustrates that, regardless of network traffic, the majority of emergency notifications are properly sent by using static routing strategies. However, earlier testing have demonstrated that enhanced reliability comes at the cost of considerably higher latency, energy usage, and makespan. Using the DDPG method results in moderate alert PDR performance. For medium, big, and extra-large queue sizes with configurations C2, C3, and C4 with 15 homes, DDPG approaches CB and FB at roughly 0.07. However, under more rigorous workloads, its performance declines as the number of households increases, achieving values between 0.008 and 0.018. This pattern suggests that in order to emphasize latency and energy minimization, DDPG occasionally sacrifices packet delivery dependability during congestion conditions.

When employing the QA technique, alert PDR values are consistent but rather low. In most cases, QA maintains delivery ratios between 0.02 and 0.08. Despite being lower than CB and FB, QA shows more consistent behavior when workload increases than DDPG, indicating the effectiveness of queue-awareness in reducing the effects of congestion. The proposed DDQNQA method demonstrates adaptive behavior. Under mild workloads, such as medium queue size with configuration




(a) Queue size 25

(b) Queue size 50

(c) Queue size 75

(d) Queue size 100

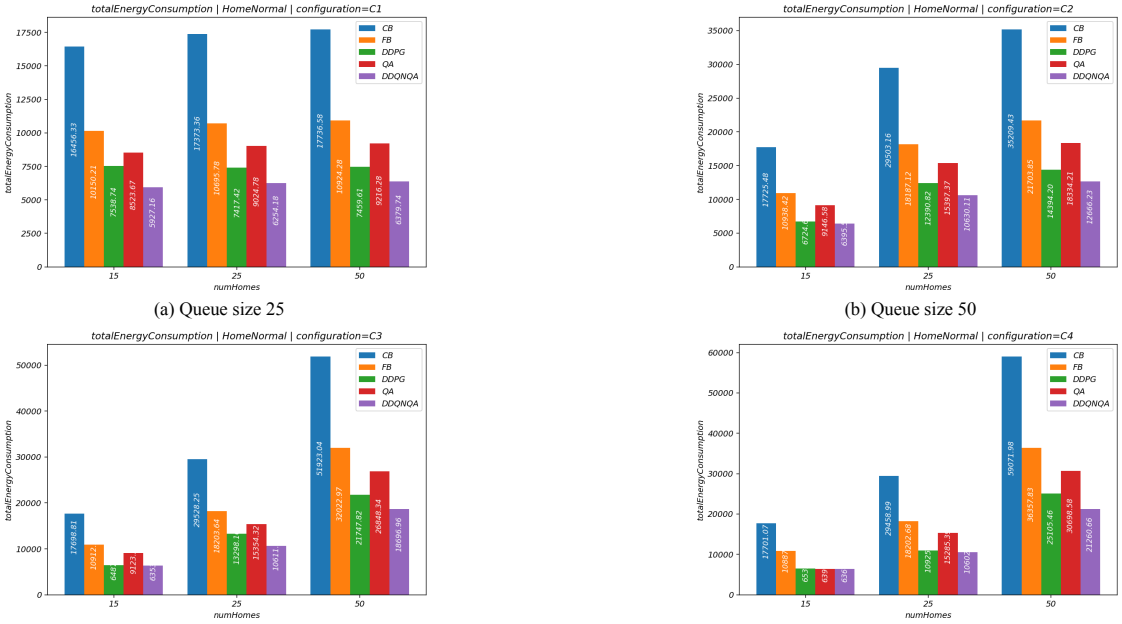
Figure 6: Alert packet delivery ratio comparison

C2 and 15 homes, DDQNQA achieves an Alert PDR of about 0.082, matching the performance of CB and FB while also achieving lower latency and energy consumption. Alert PDR declines in a number of cases as workload increases, demonstrating that the suggested technique places resource efficiency and speedy processing ahead of packet delivery maximization. Nonetheless, DDQNQA maintains competitive delivery performance in a variety of configurations while providing notable gains in latency, energy economy, and makespan. When all evaluation indicators are included, alert PDR results reveal a considerable trade-off. CB and FB maximize delivery dependability despite having the highest latency, energy consumption, and execution time. By dynamically assigning healthcare traffic depending on queue situations and reinforcement learning decisions, DDQNQA, on the other hand, strikes a compromise between system efficiency and dependability. Although it does not always achieve the highest Alert PDR, DDQNQA offers a more practical and balanced approach for real-time healthcare monitoring systems, where low latency, energy efficiency, and fast response times are just as critical as packet delivery reliability.

### 5.3.3 Energy consumption analysis

FIGURE 7 presents the Energy Consumption Analysis for different queue size and system loads. The energy consumption statistics unequivocally show how well the suggested DDQNQA strategy reduces resource usage throughout the Edge–Fog–Cloud healthcare architecture. The energy findings consistently favor DDQNQA under all tested settings, in contrast to the latency and packet drop metrics, where benefits were occasionally workload-dependent. Because the majority of healthcare tasks are sent to cloud resources, which increases transmission, processing, and communication overhead, CB has the greatest energy consumption for all queue sizes and home counts. By processing a greater percentage of tasks closer to consumers, FB uses less energy than CB, although it still has significant overhead when workload grows. By taking queue conditions into account before forwarding jobs, QA technique further increases energy efficiency. Nevertheless, out of all

the approaches that are compared, DDQNQA consistently produces the lowest energy consumption. DDQNQA lowers energy usage to about 5,927 units for small queue sizes with configuration C1 and 15 residences, as opposed to 16,456 units for CB, 10,150 units for FB, and 8,524 units for QA. Similar pattern is shown when the workload grows to 25 and 50 houses, while DDQNQA continues to use the least amount of energy even though the amount of healthcare traffic created has significantly increased. Under higher queue capacities, the benefit is even more noticeable. DDQNQA continues to use significantly less energy than other competing methods for medium, large, and extra-large queues with configurations C2, C3, and C3. DDQNQA uses about 21,261 units in the most demanding scenario of 50 homes and queue size with configuration C4, as opposed to 59,072 for CB, 36,358 for FB, and 30,699 for QA. This translates into an energy savings of roughly: 64.0%, 41.5%, and 30.7% in contrast to CB, FB, QA, respectively.




(a) Queue size 25

(b) Queue size 50

(c) Queue size 75

(d) Queue size 100

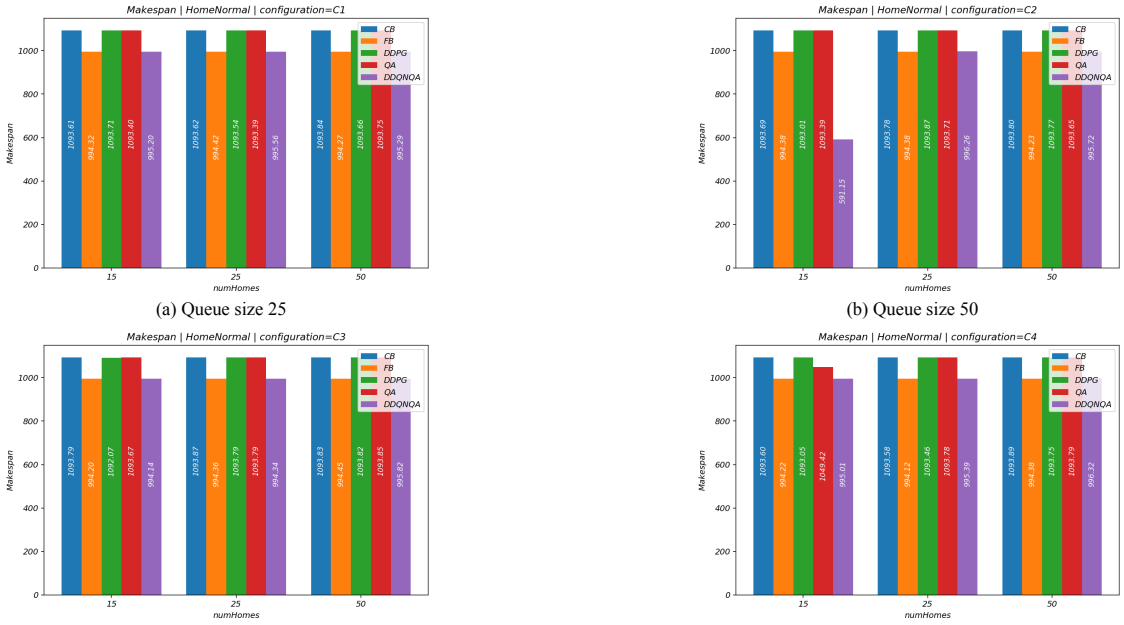
Figure 7: Total energy consumption comparison

These findings show that DDQNQA develops more effective task allocation strategies that reduce pointless transmissions and processing operations without sacrificing service quality. The suggested reinforcement learning system greatly increases total energy efficiency by dynamically choosing the best execution layer based on queue status, resource availability, task characteristics, and network conditions.

Overall, among the evaluated methods, DDQNQA provides a favorable balance between energy consumption and adaptive task routing under the tested configurations. DDQNQA continuously adjusts its routing policy by deep reinforcement learning, which leads to significant energy savings across all workload levels and queue sizes, whereas CB, FB, and QA rely on fixed or rule-based decision methods. According to these results, DDQNQA is the most energy-efficient approach for remote healthcare monitoring systems that are set up over Edge-Fog-Cloud setups.

### 5.3.4 Makespan

FIGURE 8 presents the makespan results for different queue sizes and system loads. The makespan findings show how long it takes to finish all of the Edge-Fog-Cloud infrastructure's healthcare processing activities. The suggested DDQNQA approach consistently obtains the greatest performance across nearly all investigated scenarios since lower makespan values indicate faster completion of all submitted jobs. For all workloads, DDQNQA gets the lowest makespan values of roughly 995 seconds for small queue size configuration C1, while CB, DDPG, and QA stay close to 1093 seconds. Although it takes about 994 seconds, FB has a smaller makespan than CB. DDQNQA completes the full task ahead of competing approaches, despite the improvement appearing to be quantitatively tiny. The benefit of DDQNQA is more noticeable for medium queue sizes with configuration C2. DDQNQA significantly decreases makespan to around 591 seconds in the light workload scenario (15 homes), while CB, DDPG, and QA reduce makespan to about 1094 seconds. This translates to a reduction in the overall task completion time of around 46%. All other approaches stay near 1094 seconds, while DDQNQA retains the lowest makespan values when the workload increases to 25 and 50 homes. DDQNQA continues to outperform all competing methods for big queue sizes with configuration C3. While CB, DDPG, and QA take about 1092–1094 seconds, the makespan stays around 994–996 seconds. This shows that even when queue capacities rise, the suggested routing technique effectively makes use of the resources that are available.




(a) Queue size 25

(b) Queue size 50

(c) Queue size 75

(d) Queue size 100

Figure 8: Makespan comparison

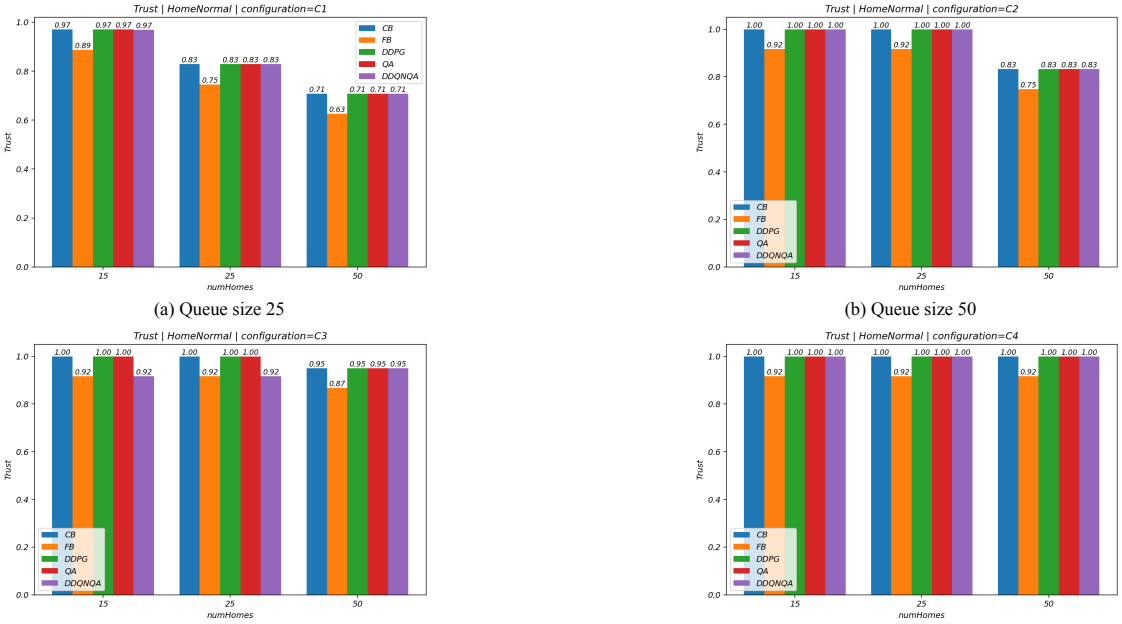
Similarly, DDQNQA consistently yields the lowest makespan values across all workloads for extra-large queue sizes with configuration C4. While CB, DDPG, and QA stay close to 1094 seconds, the makespan stays below 997 seconds. Once more, FB outperforms CB but falls short of DDQNQA in terms of efficiency. When comparing the suggested approaches, QA by itself offers no improvement over DDPG and frequently yields makespan values that are almost exactly the same as CB. On the other hand, DDQNQA routinely delivers faster task completion by combining deep reinforcement learning with queue awareness. The findings show that sophisticated adaptive routing decisions are

necessary to successfully minimize total execution time; simply taking queue status into account is insufficient. Among the evaluated methods, DDQNQA achieves the lowest makespan in most tested configurations for completing workloads, according to the makespan analysis. DDQNQA reduces total processing delays and finishes workloads ahead of all baseline methods by dynamically distributing healthcare traffic between gateway, fog, and cloud resources while taking queue circumstances into account. These outcomes show the efficacy of the suggested DDQNQA architecture for real-time healthcare monitoring environments, together with the previously noted reductions in latency and energy consumption.

### 5.3.5 Trust

FIGURE 9 presents the Trust Analysis for different queue size and system loads. Increasing queue capacity clearly improves overall system reliability, as seen by the trust results. Because fewer queue resources result in increased congestion and decreased processing reliability under Configuration C1 (Small Queues), confidence levels decline as the number of houses rises. FB consistently shows the lowest trust values, whereas CB, DDPG, QA, and DDQNQA obtain trust values of roughly 0.97 for 15 households, which drop to roughly 0.71–0.83 under greater workloads. This trend suggests that, in the face of growing healthcare traffic needs, minimal queue sizes are insufficient to ensure excellent service reliability. In Configuration C2 (Medium Queues), trust considerably rises for each procedure. CB, DDPG, QA, and DDQNQA maintain optimal trust levels of 1.0 for 15 and 25 homes; trust remains above 0.83 even for 50 homes. FB also gets better than C1, coming in at about 0.92 for lower workloads and 0.75 for 50 houses. These results demonstrate that small queue capacities successfully alleviate congestion-related trust degradation and improve service reliability. Configuration C3 (Large Queues) enhances reliability even further. While DDQNQA achieves about 0.92, CB, DDPG, and QA maintain trust levels close to or equal to 1.0 for 15 and 25 households. At the maximal workload level, trust for most algorithms remains between 0.87 and 0.95. Because the system can manage traffic spikes more efficiently because of the increased queue resources, there are fewer reliability degradations during periods of high demand.

The best trust performance is obtained using Configuration C4 (Extra-Large Queues). Across all workload levels, CB, DDPG, QA, and DDQNQA retain a trust value of 1.0, exhibiting total dependability under the assessed scenarios. Regardless of the number of households, Facebook also attains consistent trust levels of about 0.92. Overall, the findings show that queue size has a significant impact on trust. Congestion effects decrease when queue resources rise from C1 to C4, and all queue-aware techniques reach almost flawless dependability; in large-capacity configurations, DDPG, QA, and DDQNQA regularly exceed FB and match CB's performance.




(a) Queue size 25

(b) Queue size 50

(c) Queue size 75

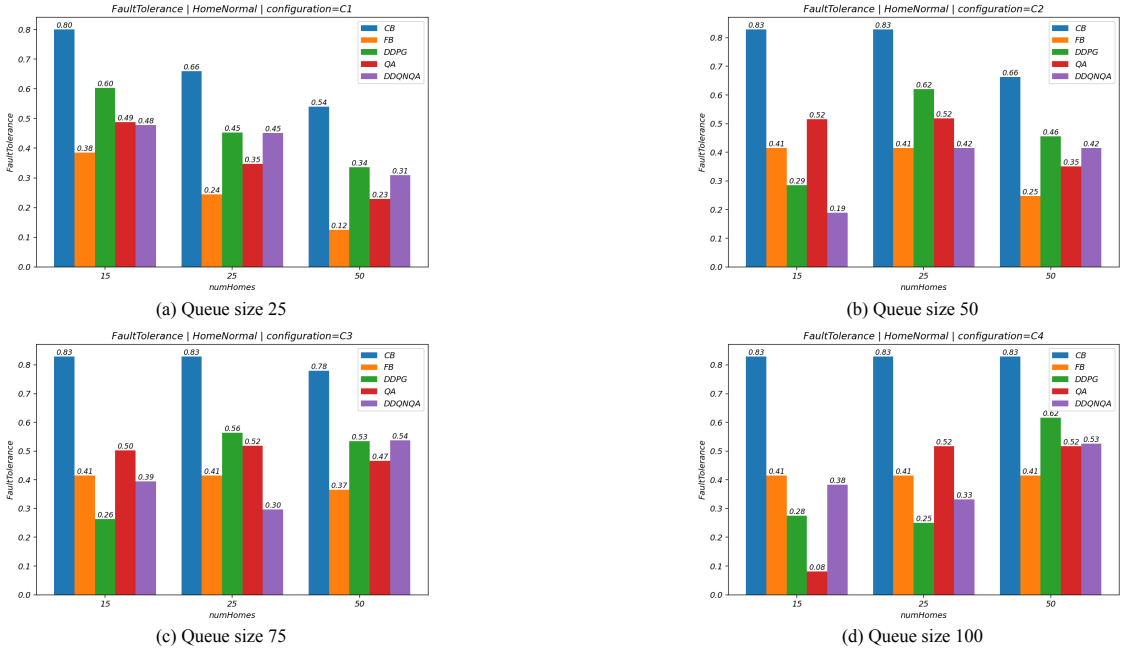
(d) Queue size 100

Figure 9: Trust comparison

### 5.3.6 Fault tolerance

FIGURE 10 presents the Fault Tolerance Analysis for different queue size and system loads. The latency and energy indicators show a distinct trend than the fault tolerance results. In contrast to energy consumption, where DDQNQA routinely beat all rivals, fault tolerance performance varied according on queue capacity and workload intensity. However, under various operating situations, the suggested methodologies typically show greater robustness than conventional routing techniques. In most settings, CB gets the highest fault tolerance values; in multiple scenarios, it reaches almost 0.83. In spite of network congestion or processing delays, a greater percentage of jobs can be successfully performed because to cloud resources' extensive computational capacity, redundancy, and reliability. This benefit, however, comes at the tradeoff of significantly larger energy use and communication overhead. FB frequently generates the lowest fault tolerance values, which range from approximately 0.12 to 0.41. The minimal computing capability of the fog nodes makes the system more vulnerable to overload scenarios, especially as the number of households increases. In terms of failure tolerance, the DDPG approach usually performs better than FB via learning adaptive routing decisions. In contrast to FB (0.41), DDPG achieves a fault tolerance rating of around 0.62 at extra-large queue size with configuration C4 and 50 households. This implies that reinforcement learning can effectively distribute healthcare responsibilities among available resources and mitigate the impacts of overloaded nodes. The QA approach exhibits competitive fault tolerance performance in certain cases. With configuration C2 and 25 homes, QA reaches about 0.52 at medium queue size, whilst FB and DDQNQA reach 0.41 and 0.42, respectively. This illustrates the advantage of specifically taking queue conditions into account when choosing a route. Nevertheless, QA exhibits instability in certain setups, especially at extra-large queue sizes with configuration C4 and 15 houses, when the fault tolerance drastically decreases to about 0.08. In principle, the suggested DDQNQA approach offers balanced fault tolerance performance across various workloads by combining queue awareness with reinforcement learning. DDQNQA

regularly beats FB and maintains its competitiveness with DDPG and QA, although not always achieving the best absolute value. Interestingly, DDQNQA achieves about 0.54 at high queue size with configuration C3 and 50 houses, marginally outperforming both DDPG (0.53) and QA (0.47). Similarly, DDQNQA reaches about 0.53 at extra-large queue size with configuration C4 and 50 houses, surpassing FB (0.41) and excelling QA (0.52).




(a) Queue size 25

(b) Queue size 50

(c) Queue size 75

(d) Queue size 100

Figure 10: FaultTolerance comparison

Overall, the findings imply that the availability of computational resources inside the infrastructure has an impact on fault tolerance in addition to routing intelligence. Due to the abundance of cloud resources, CB has the maximum fault tolerance; yet, it has a significant energy overhead. By preserving competitive fault tolerance while concurrently attaining the lowest energy usage and excellent latency performance, DDQNQA, on the other hand, offers a more balanced approach. This trade-off shows that DDQNQA provides a useful and effective routing approach for large-scale remote healthcare monitoring systems where resource efficiency and dependability are important goals.

In this study, the scalability assessment is restricted to 15, 25, and 50 mobile nodes. The neural-network input and output dimensions do not grow as the number of monitored devices increases because DDQNQA employs an aggregated 11-dimensional state and three routing operations. Larger deployments, however, would result in more traffic and wait congestion. Therefore, without direct evaluation, no numerical performance claim is offered for deployments with 500–1000 nodes; such extensive testing is left for future work.

**DDQNQA Decision Overhead:** The current simulation does not assess hardware-specific inference delay or wall-clock DDQN training time; instead, it concentrates on routing and network performance. While an online routing choice simply necessitates a forward pass over the trained network and the selection of one of three routing actions, model training can be carried out offline in a realistic implementation. With two hidden layers of 128 neurons each, three output actions, and

an 11-dimensional input, the DDQNQA network has 18,435 trainable parameters and needs about 18,176 multiply-accumulate operations for each forward decision. Hardware-specific inference latency will be assessed on actual devices in the future.

**DDQNQA Computational Overhead:** The OMNeT++ Edge-Fog-Cloud model's processing and communication operations are taken into consideration by the energy metric offered in this study. There is no distinct model for the hardware-level energy used by DDQNQA neural-network inference. For a single forward routing decision, the DDQNQA network needs about 18,176 multiply-accumulate operations and has 18,435 trainable characteristics. Future real-device implementation will include measurements on hardware-specific inference time and energy consumption.

## 6 CONCLUSION

For real-time healthcare monitoring in Edge-Fog-Cloud systems, two queue-aware routing strategies were examined: QueueAware (QA) and DDQNQueueAware (DDQNQA). DDQNQA employs Double Deep Q-Learning to make adaptive discrete routing decisions, whereas QA uses lightweight queue- and latency-aware routing. In comparison to the assessed baselines, DDQNQA typically decreased latency, energy consumption, processing time, makespan, and communication overhead within the evaluated OMNeT++ scenarios while preserving competitive trust and fault-tolerance performance. These results suggest that deep reinforcement learning and queue awareness could be combined for latency-sensitive healthcare routing under the studied circumstances. Future research will concentrate on verifying the suggested DDQNQA framework with genuine wearable technology, medical sensors, and Edge-Fog-Cloud computing platforms in real-world healthcare settings. Larger-scale deployments, repeated statistical validation, multi-agent reinforcement learning, federated learning, trust-aware routing, and comparisons with other DRL techniques will all be taken into account in future research.

## 7 ACKNOWLEDGMENTS

The project was funded by KAU Endowment (WAQF) at king Abdulaziz University, Jeddah, Saudi Arabia. The authors, therefore, acknowledge with thanks WAQF and the Deanship of Scientific Research (DSR) for technical and financial support.

## References

[1] Baker SB, Xiang W, Atkinson I. Internet of Things for Smart Healthcare: Technologies, Challenges, and opportunities. IEEE Access. 2017;5:26521-44.

[2] Alshehri F, Muhammad G. A Comprehensive Survey of the Internet of Things (IoT) and AI-Based Smart Healthcare. IEEE Access. 2021;9:3660-3678.

[3] Kashani MH, Madanipour M, Nikravan M, Asghari P, Mahdipour E. A Systematic Review of IoT in Healthcare: Applications, Techniques, and Trends. J Netw Comput Appl. 2021;192:103164.

[4] https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf

[5] Chen M, Mao S, Liu Y. Big Data: A Survey. Mob Netw Applx. 2014;19:171-209.

[6] Amin SU, Hossain MS. Edge Intelligence and Internet of Things in Healthcare: A Survey. IEEE Access. 2021;9:45-59.

[7] Shi W, Cao J, Zhang Q, Li Y, Xu L. Edge Computing: Vision and Challenges. IEEE Internet Things J. 2016;3:637-646.

[8] Bonomi F, Milito R, Zhu J, Addepalli S. Fog computing and its role in the internet of things. In Proceedings of the first edition of the MCC workshop on Mobile cloud computing. ACM. 2012:13-16.

[9] Dastjerdi AV, Buyya R. Fog Computing: Helping the Internet of Things Realize Its Potential. Computer. 2016;49:112-116.

[10] Gia TN, Jiang M, Rahmani AM, Westerlund T, Liljeberg P, et al. Fog Computing in Healthcare Internet of Things: A Case Study on Ecg Feature Extraction. In 2015 IEEE international conference on computer and information technology; ubiquitous computing and communications; dependable, autonomic and secure computing; pervasive intelligence and computing. IEEE. 2015:356-363.

[11] Rahmani AM, Gia TN, Negash B, Anzanpour A, Azimi I, et al. Exploiting Smart E-Health Gateways at the Edge of Healthcare Internet-Of-Things: A Fog Computing Approach. Future Gener Comput Syst. 2018;78:641-658.

[12] Chiang M, Zhang T. Fog and IoT: An Overview of Research Opportunities. IEEE Internet Things J. 2016;3:854-864.

[13] Mouradian C, Naboulsi D, Yangui S, Glitho RH, Morrow MJ, et al. A Comprehensive Survey on Fog Computing: State-Of-The-Art and Research Challenges. IEEE Commun Surv Tutor.. 2017;20:416-464.

[14] Kashyap V, Kumar A, Kumar A, Hu YC. A Systematic Survey on Fog and IoT-Driven Healthcare: Open Challenges and Research Issues. Electronics. 2022;11:2668.

[15] Daraghmi YA, Daraghmi EY, Daraghma R, Fouchal H, Ayaida M. Edge–Fog–Cloud Computing Hierarchy for Improving Performance and Security of Nb-Iot-Based Health Monitoring Systems. Sensors. 2022;22:8646.

[16] Khanh QV, Hoai NV, Van AD, Minh QN. An Integrating Computing Framework Based on Edge-Fog-Cloud for Internet of Healthcare Things Applications. Internet of Things. 2023;23:100907.

[17] AlQahtani SA. An Evaluation of E-Health Service Performance Through the Integration of 5G IoT, Fog, and Cloud Computing. Sensors. 2023;23;23:5006.

[18] Mao Y, You C, Zhang J, Huang K, Letaief KB. A Survey on Mobile Edge Computing: The Communication Perspective. IEEE Commun Surv Tutor. 2017;19:2322-2358.

[19] Abbas N, Zhang Y, Taherkordi A, Skeie T.Mobile Edge Computing: A Survey. IEEE Internet Things J. 2017;5:450-465.

[20] Choppara P, Lokesh B. Efficient Task Scheduling and Load Balancing in Fog Computing for Crucial Healthcare Through Deep Reinforcement Learning. IEEE Access. 2025;13:26542-26563.

[21] Tuli S, Basumatary N, Gill SS, Kahani M, Arya RC, et al. Healthfog: An Ensemble Deep Learning Based Smart Healthcare System for Automatic Diagnosis of Heart Diseases in Integrated IoT and Fog Computing Environments. Future Gener Comput Syst. 2020;104:187-200.

[22] Ahmadi Z, Haghi Kashani M, Nikravan M, Mahdipour E. Fog-Based Healthcare Systems: A Systematic Review. Multim Tools Appl. 2021;80:36361-36400.

[23] Dash S, Biswas S, Banerjee D. Edge and Fog Computing in Healthcare: A Review. Scalable Comput Pract Exp. 2019;20:191-206.

[24] Satyanarayanan M. The Emergence of Edge Computing. Computer. 2017;50:30-39.

[25] Mach P, Becvar Z. Mobile edge computing: A survey on architecture and computation offloading. IEEE Commun Surv Tutor. 2017;19:1628-1656.

[26] Varghese B, Wang N, Barbhuiya S, Kilpatrick P, Nikolopoulos DS. Challenges and opportunities in edge computing. In 2016 IEEE international conference on smart cloud (SmartCloud). IEEE. 2016:20-26.

[27] `http://www.omnetpp.org`