Sustainable Computing: Informatics and Systems 48 (2025) 101218

Contents lists available at ScienceDirect

Sustainable Computing: Informatics and Systems

journal homepage: www.elsevier.com/locate/suscom

SDN-Based NFV deployment for multi-objective resource allocation in edge computing: A deep reinforcement learning for iot workload scheduling

Mehdi Hosseinzadeh [a,b,c,1], Amir Haider [d,1], Amir Masoud Rahmani [e], Farhad Soleimanian Gharehchopogh [f], Shakiba Rajabi [f], Parisa Khoshvaght [g,h], Thantrira Porntaveetus [i,*], Sang-Woong Lee [j,*]

[a] Institute of Research and Development, Duy Tan University, Da Nang, Viet Nam
[b] Department of AI, School of Computer Science and Engineering, Galgotias University, Greater Noida, India
[c] Jadara Research Center, Jadara University, Irbid 21110, Jordan
[d] Department of AI and Robotics, Sejong University, Seoul 05006, Republic of Korea
[e] Future Technology Research Center, National Yunlin University of Science and Technology, Yunlin, Taiwan
[f] Department of Computer Engineering, Ur.,C.,Islamic Azad University, Urmia, Iran
[g] Department of Biosciences, Saveetha School of Engineering, Saveetha Institute of Medical and Technical Sciences, Chennai, 602105, India
[h] School of Engineering & Technology, Duy Tan University, Da Nang, Viet Nam
[i] Center of Excellence in Precision Medicine and Digital Health, Department of Physiology, Faculty of Dentistry, Chulalongkorn University, Bangkok, 10330, Thailand
[j] Pattern Recognition and Machine Learning Lab, School of Computing, Gachon University, Seongnam 13120, South Korea

ARTICLE INFO

Keywords:
Edge Computing
Resource Allocation
Human-Computer Interaction
Software-Defined Networking (SDN)
Network Function Virtualization (NFV)
IoT User Experience
Cognitive Load

ABSTRACT

The rapid growth of Internet of Things (IoT) devices presents significant challenges, particularly regarding resource management in real-time data processing environments. Traditional cloud computing struggles with high delay times and limited bandwidth, affecting user interaction and cognitive load. Edge computing mitigates these issues by decentralizing data processing and bringing resources closer to IoT devices, ultimately influencing human-computer interaction. This paper introduces a framework for resource allocation in edge computing environments, leveraging Software-Defined Networking (SDN) and Network Function Virtualization (NFV) alongside Deep Q-Network (DQN) optimization. The framework aims to enhance user experiences by improving CPU, memory, and storage efficiency while reducing network delays, contributing to a smoother and more efficient interaction with IoT systems. Simulated results demonstrate a 40 % improvement in CPU utilization, 30 % in memory, and 20 % in storage efficiency, which can positively impact IoT devices' perceived effectiveness and usability.

1. Introduction

The Internet of Things (IoT) devices have greatly influenced various sectors, and lead to new applications in smart cities, healthcare, and industrial automation [1]. With this rapid growth comes significant challenges, such as data processing difficulties, network delays, and congestion. This happens because traditional systems often cannot keep up with the increasing demands [2]. Traditional cloud computing, that depends on central data processing and storage, need help to meet modern IoT applications' immediate and low-latency demands. Relying on distant cloud data centers leads to longer delays and more bandwidth use. This can negatively impact the performance of applications that need quick responses, such as self-driving cars or real-time health monitoring systems [3,4].

Edge computing addresses the shortcomings of traditional cloud computing by processing data closer to its origin. This method situates computing resources near IoT devices, typically at the network's edge. Such proximity enables faster data processing, reducing the necessity to transmit large data volumes to centralized cloud servers [5–7]. By handling data locally, edge computing minimize the delays usually seen

* Corresponding authors.
E-mail addresses: mehdihosseinzadeh@duytan.edu.vn (M. Hosseinzadeh), amirhaider@sejong.ac.kr (A. Haider), rahmania@yuntech.edu.tw (A.M. Rahmani), Farhad.soleimanian@iau.ac.ir (F.S. Gharehchopogh), shakiba.rajabi@ieee.org (S. Rajabi), parisakhoshvaght@duytan.edu.vn (P. Khoshvaght), thantrira.p@chula.ac.th (T. Porntaveetus), slee@gachon.ac.kr (S.-W. Lee).
[1] These authors contributed equally to this work.

https://doi.org/10.1016/j.suscom.2025.101218

Available online 30 September 2025

with data transfer, making it ideal for real-time applications [8]. It also alleviates network congestion and decreases the demand for bandwidth by shifting tasks away from central cloud facilities [9]. However, edge computing introduces new challenges, especially in resource allocation and management. The distributed nature of edge environments demands sophisticate techniques to effectively distribute computational, storage, and network resources [10]. These challenges grow with the dynamic and fluctuating workloads typical in IoT settings, where resource needs can shift drastically over time. To tackle these problems, edge computing is increasingly adopting advanced technologies like Software-Defined Networking (SDN) and Network Function Virtualization (NFV) [11]. Yet, integrating these technologies introduces added complexity. SDN facilitates centralized network control, allowing for dynamic routing and resource management, while NFV enables the versatile deployment of virtual network functions across edge nodes [12, 13]. Effective integration of these technologies is essential to enhance edge computing setups' flexibility, scalability, and efficiency.

SDN is important in edge computing because it provide a centralized and programmable way to manage networks. SDN separates the control functions from the data-handling functions, making it easier to adjust and manage the network [14]. This central control helps in efficient data routing, better use of resources, and quick adaptation to changes in the network, which is essential in the fast-changing environments of edge computing [15]. NFV works alongside SDN by turning network functions like firewalls, load balancers, and routers, which were traditionally hardware-based, into virtual functions [16]. In edge computing, NFV allows these NFVs to be deployed on different edge nodes as needed, improving scalability and making better use of resources [17]. By virtualizing and spreading out these functions, NFV makes network management more flexible and cost-effective, further enhancing the performance of edge computing systems [18].

Many studies have explored resource allocation in edge computing environments, particularly integrating SDN and NFV. Still, they often fall short in handling the complexity and dynamism of IoT scenarios. Most existing frameworks either rely on static resource allocation strategies or do not fully leverage the potential of machine learning with SDN and VFN for adaptive optimization. These approaches often focus on isolated performance metrics without considering the holistic trade-offs among CPU, memory, storage utilization, energy consumption, and latency across varying network conditions.

Our research addresses these gaps by introducing a comprehensive resource allocation framework integrating SDN-driven NFV deployment with Deep Q-Network (DQN) enhanced scheduling. Unlike previous studies, our framework is designed to dynamically adjust resource allocation in real time, simultaneously considering a broad set of performance metrics. The innovative use of DQN allows the framework to learn from ongoing operations, predicting and adapting to changing network conditions, workload intensities, and resource availability. The

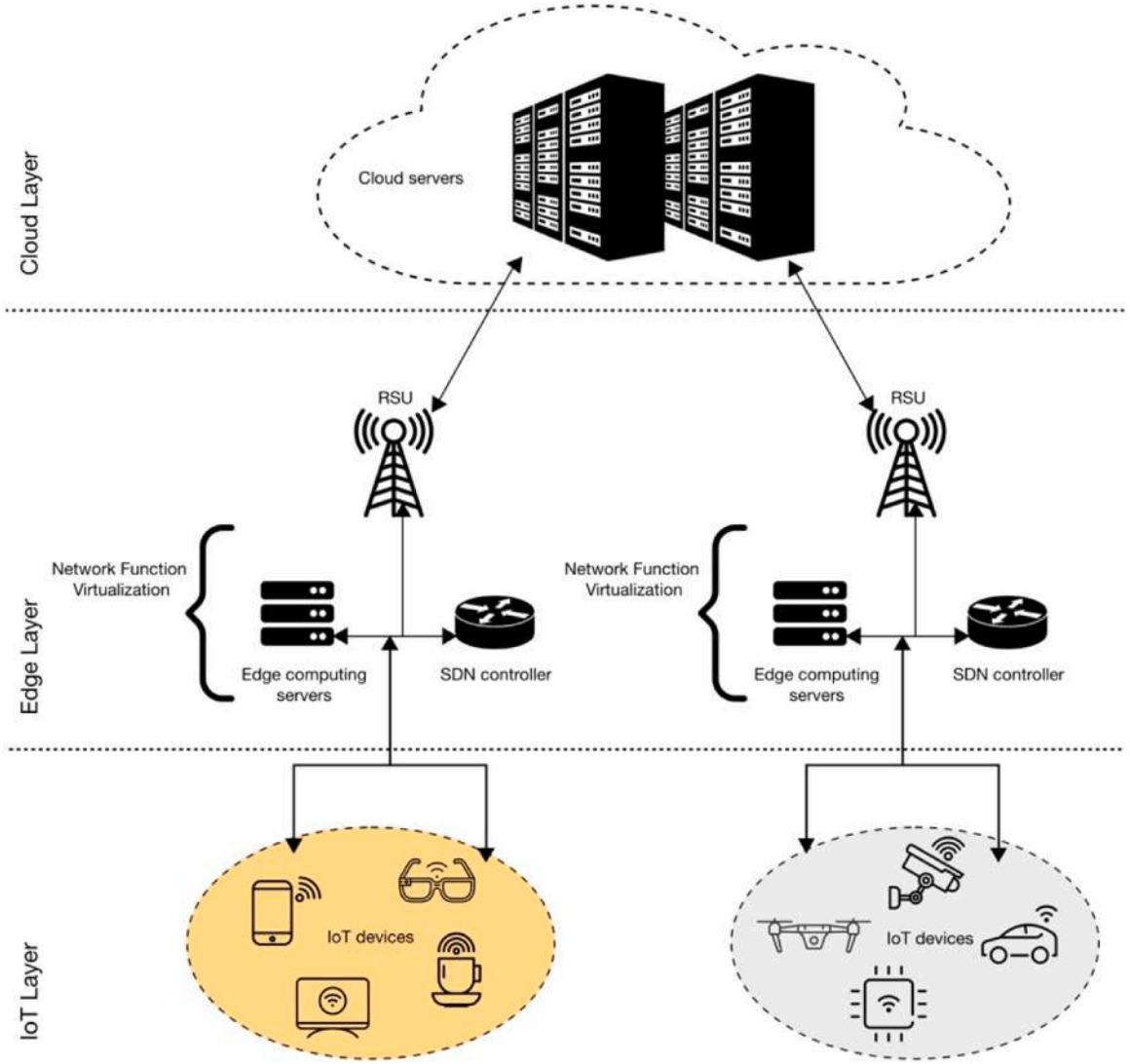

Fig. 1. System Architecture.

critical contributions of our paper are:

- Implement a real-time, multi-objective optimization framework for dynamic resource allocation in edge nodes.
- Integration of SDN-based network control with NFV-driven function deployment for enhanced system flexibility and scalability.
- Utilization of RL via Deep Q-Networks to continuously adapt resource allocation strategies based on network conditions.
- Development of energy-aware resource management algorithms that dynamically balance computational load and power consumption.
- Executing scenario-driven simulations to rigorously evaluate the framework's performance across varying operational conditions and workloads.

Fig. 1 illustrates a three-layer architecture where cloud servers, SDN-controlled edge computing with NFV, and IoT devices interact, enabling efficient communication and smooth data exchange in the network.

This paper is organized as follows: Section 2 review previous studies on resource allocation in edge computing that uses SDN and NFV. Section 3 introduces the system model, which integrates SDN, NFV, and DQN optimization for managing resources dynamically. Section 4 explains the simulation setup and the parameters used to test the framework. In Section 5, we analyze the simulation results and discuss their implications for the performance and efficiency of edge computing. Finally, Section 6 concludes the paper by summarizing the main findings, discussing the study's limitations, and suggesting ideas for future research.

2. Related works

This section reviews the current research on edge computing, especially studies that combine it with Software-Defined Networking (SDN) and Network Function Virtualization (NFV). Different Internet of Things (IoT) methods have been proposed to enhance resource allocation and scheduling. To start with, Alonso et al. [19] proposed an improved Edge-IoT architecture that integrates SDN and NFV with DRL techniques. This architecture, named GECA 2.0, incorporates a DQN model to optimize network resource allocation and virtual data flow management within SDN/NFV-enabled Edge-IoT environments. The proposed architecture enhances scalability, flexibility, and efficiency in managing IoT networks by dynamically reconfiguring network flows and resource distribution through a centralized SDN controller. Rakkiannan et al. [20] presented an automated network slicing framework for edge computing environments using SDN and NFV integrated with federated learning. The framework enables efficient and scalable network slice management by leveraging federated learning to dynamically predict slice templates and optimize resource allocation. The proposed system utilizes SDN controllers as orchestrators to manage slice creation, resource allocation, and lifecycle management across multiple domains, enhancing the overall efficiency and responsiveness of 5 G networks.

Optimizing mobile edge computing migration in next-generation IoT networks presents challenges in cost, time, bandwidth, and energy consumption. To address these issues, Lv et al. [21] proposed a computing migration framework that leverages SDN and NFV. The framework introduces a multi-attribute decision-making model that selects the optimal mobile edge computing center for task migration, balancing network load, and improving user service quality. This approach ensures efficient resource utilization in IoT environments by enhancing the overall performance of mobile edge computing. In cloud-based manufacturing environments, achieving flexible resource scheduling poses significant challenges. To address this, Yang et al. [22] introduced a software-defined cloud manufacturing (SDCM) model integrated with edge computing. The model leverages SDN to separate control logic from hardware, enabling dynamic reconfiguration and efficient resource utilization. Additionally, the authors developed an optimization model for controlling time-sensitive data traffic in manufacturing tasks, employing a combination of genetic algorithms, Dijkstra's shortest path algorithm, and a queuing algorithm to optimize subtask allocation and data routing. Moreover, He et al. [23] proposed an iterative Maximal Independent Set (MIS)–based multiple migration planning and scheduling algorithm for SDN-enabled edge computing environments. This approach models resource competition among live migration requests as a dynamic resource dependency graph, enabling efficient scheduling of large-scale, mobility-induced container migrations. The proposed algorithm optimizes resource allocation and minimizes migration time by leveraging SDN capabilities. It is suitable for real-time, large-scale edge computing scenarios where user mobility significantly impacts network performance. Furthermore, Li et al. [24] proposed a software-defined heterogeneous edge computing network architecture for optimizing network resource scheduling. This architecture separates the control layer from the data layer, enabling centralized control and efficient resource management in heterogeneous edge environments. They developed a Proximal Policy Optimization (PPO) algorithm based on DRL to address the multi-objective optimization of network energy consumption and load balancing, allowing dynamic task scheduling across various edge computing nodes and improving overall network efficiency.

Optimizing data caching and classification in IoT healthcare solutions presents significant challenges. Jazaeri et al. [25] proposed an SDN-based edge computing architecture tailored for healthcare environments to address these. This architecture employs a spectral clustering method to group patients according to their health data and proximity, associating each cluster with an SDN edge node for efficient data caching. Additionally, the MFO-Edge Caching algorithm is introduced to select near-optimal content for caching, considering factors such as data freshness, priority, and energy levels, ultimately aiming to enhance Quality of Service (QoS) in healthcare networks. Hu et al. [26] proposed a FitPath scheme for QoS routing in integrated Edge Computing and SDN environments. The scheme introduces a path selection method based on a fittingness measure, which optimizes the routing paths by considering bandwidth cost, latency, and packet loss rate. FitPath dynamically adjusts network resources to meet varying service requirements while maintaining efficient data delivery, ensuring that QoS guarantees are met across edge-to-edge connections in SDN-managed networks. Du et al. [27] proposed an SDN-based architecture for resource allocation in hybrid edge and cloud computing systems in their publication. They introduced an evolutionary Stackelberg differential game approach to optimize resource pricing and allocation between cloud computing service providers (CCPs) and edge computing service providers (ECPs). The framework dynamically adjusts strategies based on the replicator dynamics of users' service selection, ensuring efficient utilization of computing resources and meeting time-varying computational demands in 5 G heterogeneous networks. Table 1 compares the key contributions, limitations, and complexity levels of related schemes with our approach.

Little is known about effectively combining multi-objective optimization with SDN and NFV for dynamic resource management in edge computing. While existing studies, like Alonso et al. [19] and Rakkiannan et al. [20], explore DRL and federated learning for resource optimization, they often lack a comprehensive multi-objective approach or face scalability issues. Our study addresses these gaps by developing a dynamic multi-objective framework using DRL integrated with SDN and NFV. This framework continuously adapts to real-time conditions, optimizing CPU, memory, storage, latency, and energy efficiency. Unlike static or single-objective models in prior work, our approach offers robust performance across diverse, dynamic scenarios.

3. System model and proposed method

This section describes the system model and method we created to improve resource allocation in edge computing systems that use Software-Defined Networking (SDN) and Network Functions

Table 1
Properties of the related schemes and this paper.

Ref	Decision-Making Model	Contributions	Possible limitations	Tools	Methods			Performance Metrics					
					NFV	SDN	Edge Computing	CPU Utilization	Memory Utilization	Storage Utilization	Network Latency	Energy Efficiency	Task Handling
[19]	Hierarchical Model, SDN/NFV, DRL	Introduced SDN and NFV management to optimize data flows using Deep Q-Networks in GECA	Requires extensive training data computational resources and faces challenges integrating with existing IoT environments	OpenFlow, Python, TensorFlow, Mininet, POX	✓	✓	✓	×	×	×	✓	×	×
[20]	FL, SDN/NFV	Implements federated learning for dynamic, automated network slicing using SDN and NFV technologies.	Complexity in coordination among distributed models and dependency on high-quality local data updates.	NA	✓	✓	✓	×	×	×	✓	×	×
[21]	SDN/NFV	They developed MADM models using SDN and NFV for efficient IoT edge computing migration strategies.	Computational complexity and accuracy issues due to varied index scales in the decision-making model.	MATLAB	✓	✓	✓	×	×	×	✓	✓	×
[22]	Genetic Algorithm, Dijkstra's shortest path algorithm, SDN/NFV	Introduces flexible resource allocation in software-defined cloud manufacturing using edge computing.	Requires significant computational resources for real-time data processing and optimization.	NA	✓	✓	✓	×	×	×	×	✓	×
[23]	Maximal Independent Set algorithm, SDN	Enhances large-scale live container migration scheduling using MIS-based algorithm in SDN-enabled environments.	Faces scalability challenges and high computational demands in dynamic and large-scale network environments.	OpenDaylight	×	✓	✓	×	×	×	×	✓	×
[24]	Proximal Policy Optimization, SDN	We developed a PPO-based resource scheduling algorithm for software-defined heterogeneous edge computing networks.	The model may struggle with dynamic real-world conditions and large-scale network complexities.	NA	×	✓	✓	×	×	×	×	✓	×
[25]	Moth-Flame Optimization	Utilizes spectral clustering and MFO-Edge Caching to improve e-health IoT data retrieval and QoS.	Challenges with dynamic conditions, high computational demands, and scaling in real-world applications.	NA	×	✓	✓	×	×	×	✓	×	×
[26]	FitPath	Developed FitPath for efficient QoS-based path selection in integrated edge and SDN networks.	It is limited by computational complexity and scalability in dynamic, large-scale network environments.	NA	×	✓	✓	×	×	×	✓	×	×
[27]	Stackelberg differential game-based	Introduced an SDN-based resource sharing using evolutionary and Stackelberg differential game models.	A complex model requires accurate, real-time data but is limited by scalability and implementation challenges.	NA	×	✓	✓	×	×	×	×	×	×
Ours	SDN/NFV, Deep Q-Networks	Develops a dynamic multi-objective framework for efficient resource allocation in SDN-NFV Edge computing.	Does not integrate advanced security measures into the resource allocation framework.	Python, Tensorflow, Mininet	✓	✓	✓	✓	✓	✓	✓	✓	✓

Virtualization (NFV). This setup is made to handle the demanding needs of the Internet of Things (IoT) applications. The main goals are to boost computational efficiency, reduce latency, and better use resources across various edge nodes. Table 2 lists the abbreviations used in the system model section and their descriptions.

3.1. Overview of the System Architecture

Edge computing is a system with computing resources closer to IoT devices. This helps reduce delays, use less bandwidth, and make real-time applications faster. In this system, edge nodes are placed in different locations. Each node has its own computing power, storage, and network access. It is designed to operate effectively in heterogeneous environments where nodes may differ in resource availability. The model dynamically adjusts resource allocation based on the specific capabilities of each node, ensuring optimal performance across a diverse range of edge computing setups. In this architecture, consider an edge network consisting of N edge nodes, denoted as $E_1, E_2, \ldots, E_N$. Each edge node $E_i (i = 1, 2, \ldots, N)$ is characterized by its computational capacity C_i, storage capacity S_i, and available bandwidth B_i. The computational capacity C_i is typically measured in CPU cycles per second, the storage capacity S_i in gigabytes (GB), and the bandwidth B_i in megabits per second (Mbps). The IoT devices connected to each edge node E_i are represented as a set $\mathscr{D}_i = \{D_{i1}, D_{i2}, \ldots, D_{im_i}\}$, where m_i is the number of devices connected to E_i.

The interaction between these IoT devices and edge nodes Involves the dynamic allocation of resources to handle tasks $T_{i1}, T_{i2}, \ldots, T_{in_i}$ where n_i is the number of tasks generated by the devices in $\mathscr{D}_i$. The decision process for whether to process a task locally at an edge node E_i or offload it to another node or the cloud is modeled by an optimization problem that balances resource availability, latency, and energy consumption.

Table 2
Applied Abbreviations.

Abbreviation	Description	Abbreviation	Description
SDN	Software-Defined Networking	NFV	Network Function Virtualization
IoT	Internet of Things	CPU	Central Processing Unit
GB	Gigabytes	Mbps	Megabits per second
E_i	Edge Node	C_i	Computational Capacity
S_i	Storage Capacity	B_i	Bandwidth
D_i	Set of IoT devices	T_{ij}	Task generated by IoT device
L_{ij}	Latency for processing a task	S_{ij}	Size of task
W_{ij}	Workload factor for a task	E_{ij}	Energy consumption for processing a task
P_i	Power consumption per CPU cycle	$C_{op}(E_i)$	Operating cost for an Edge Node
$\alpha_i, \beta_i, \gamma_i$	Weighting factors for cost, energy, and overhead	C_{task}	Task handling capacity
V	Set of NFVs	R_j^C	Computational resources required by NFV
R_j^S	Storage resources required by NFV	R_j^B	Bandwidth resources required by NFV
SDN Controller	Central control mechanism for network	$L(E_i, V_j)$	Latency for deploying NFV on Edge Node
$E(E_i, V_j)$	Energy consumed by Edge Node when NFV is active	$C_{op}(E_i, V_j)$	Operational cost for deploying NFV
λ, μ, ν	Weighting factors for latency, utilization, coverage	DQN	Deep Q-Network
$R_{ij(t)}$	Resources allocated to a task at time ttt	$U_{opt,CPU}$, $U_{opt,MEM}$, $U_{opt,STOR}$	Optimal utilization levels for CPU, memory, storage
RTT	Round-Trip Time	$E_{task,ij}$	Energy for processing a task on Edge Node

The latency L_ij for processing a task T_{ij}at edge node E_i modeled as shown in Eq. (1).

$$L_{ij} = \frac{S_{ij}}{B_i} + \frac{C_{ij} \cdot W_{ij}}{C_i} \tag{1}$$

where S_{ij} is the size of the task $T_{ij}B_i$ is the bandwidth available at the edge node E_i, C_i represents the computational complexity of the task T_{ij} (in CPU cycles), and W_{ij} is the workload factor associated with the task.

The energy consumption E_{ij} for processing task T_{ij} on edge node E_i is given by Eq. (2).

$$E_{ij} = P_i \cdot \frac{C_{ij} \cdot W_{ij}}{C_i} \tag{2}$$

where P_i represents the power consumption per CPU cycle at edge node E_i. The total energy consumption for all tasks processed by edge node E_i expressed as Eq. (3).

$$E_i^{\text{total}} = \sum_{j=1}^{n_i} E_{ij} = \sum_{j=1}^{n_i} P_i \cdot \frac{C_{ij} \cdot W_{ij}}{C_i} \tag{3}$$

The cost function $C_{\text{op}}(E_i)$ associated with operating an edge node E_i for a set of tasks that formulated as Eq. (4).

$$C_{\text{op}}(E_i) = \alpha_i \cdot \left(\sum_{j=1}^{n_i} \frac{C_{ij} \cdot W_{ij}}{C_i} \right) + \beta_i \cdot E_i^{\text{total}} + \gamma_i \cdot O_i \tag{4}$$

where α_i, β_i, and γ_i are weighting factors that balance the computational cost, energy consumption, and operational overhead O_i.

NFV and SDN are integrated into this architecture to enhance the dynamic allocation of resources. NFV allows for the flexible deployment of NFVs across the edge nodes, while SDN provides a centralized control mechanism for managing the network's data flows and optimizing resource allocation.

Let the set of NFVs deployed across the network be $\mathscr{V} = \{V_1, V_2, \ldots, V_k\}$, where each NFV V_j (for $j = 1, 2, \ldots, k$) requires certain computational resources R_j^C, storage R_j^S, and bandwidth R_j^BAn SDN controller governs the optimization of NFV deployment. The controller seeks to minimize the overall network cost while maintaining performance. The objective function for the SDN controller is expressed as Eq. (5).

$$\min \sum_{i=1}^{N} \sum_{j=1}^{k} \left(\alpha_{ij} \cdot L(E_i, V_j) + \beta_{ij} \cdot E(E_i, V_j) + \gamma_{ij} \cdot C_{\text{op}}(E_i, V_j) \right) \tag{5}$$

subject to the Eq. (6) constraints.

$$\sum_{j=1}^{k} \alpha_{ij} \cdot R_j^C \le C_i, \sum_{j=1}^{k} \alpha_{ij} \cdot R_j^S \le S_i, \sum_{j=1}^{k} \alpha_{ij} \cdot R_j^B \le B_i \tag{6}$$

where α_{ij} is a binary decision variable that indicates whether NFV V_j is deployed on edge node E_i (α_{ij}=1 if deployed, 0 otherwise). The terms $L(E_i, V_j)$, $E(E_i, V_j)$, and $C_{\text{op}}(E_i, V_j)$ represent the latency, energy consumption, and operational cost of deploying V_j on E_i, respectively. Precisely, $L(E_i, V_j)$ is the latency experienced when deploying and operating V_j on E_i, taking into account the network delay and processing time. $E(E_i, V_j)$ denotes the energy consumed by E_i when V_j is active, which is a function of both the computational demand of V_j and the efficiency of E_i. The operational cost $C_{\text{op}}(E_i, V_j)$ includes immediate resource usage and longer-term costs such as wear and tear on the hardware and any penalties for exceeding resource limits.

To make the deployment and management of NFVs better across edge nodes, the SDN controller updates network settings based on real-time data like resource usage, latency, and traffic patterns. This real-time feedback allows the system to react quickly to changes in the network, ensuring good performance and efficient resource use. For example, suppose an edge node has too much traffic and is reaching its resource limits. In that case, the SDN controller can move NFVs to other

nodes or adjust their resource allocation to maintain high service quality.

3.2. Network topology and edge node configuration

The network layout in an edge computing system plays a crucial role in how well resources are allocated, data is processed, and communication happens within the network [28]. The layout includes the arrangement and connections between edge nodes, IoT devices, and the central SDN controller, creating a structure that helps manage the changing demands of edge computing [29].

3.2.1. Network topology

The network setup in our system is built with a distributed structure, where edge nodes are strategically placed to improve the performance of IoT devices and reduce network delays. These edge nodes help share the computing work across the network and keep communication with IoT devices fast. Each edge node connects to several IoT devices, forming a cluster that processes data locally, reducing the need to send data to the cloud. The connections between the edge nodes and IoT devices are made using wireless communication links.

The central SDN controller, which controls the whole network, is placed in the edge layer and has a network view. It links to all edge nodes using fast and low-latency connections. This central connection lets the SDN controller manage network resources, adjust data traffic, and decide the best places for NFVs across the network in real-time.

The placement of edge nodes in the network depends on several important factors. One key factor is the proximity of nodes to IoT devices, as this affects the speed and delay in processing data. Edge nodes are usually placed near the devices they support, especially in applications where low latency is crucial, such as with self-driving cars, intelligent energy systems, or factory automation [30]. Another critical factor is the availability of resources. Edge nodes are placed where there is enough power, cooling, and space for the hardware and where there is room to expand as the network grows [31].

The optimal placement of edge nodes is expressed as an optimization problem that seeks to minimize the total latency across all IoT devices while maximizing resource utilization and ensuring coverage. Let L_i represent the latency between an IoT device $\mathscr{D}_i$ and its corresponding edge node E_i, U_i denote the resource utilization of the edge node E_i, and C_i represent the coverage area of the edge node E_i. The objective function is formulated as Eq.(7).

$$\min \sum_{i=1}^{M} (\lambda \cdot L_i + \mu \cdot (1 - U_i) + \nu \cdot (1 - C_i)) \tag{7}$$

subject to Eq.(8).

$$L_i \leq L_{\max}, U_i \leq U_{\max}, C_i \geq C_{\min} \tag{8}$$

where M is the total number of IoT devices, λ, μ, and ν are weighting factors that balance the importance of latency, resource utilization, and coverage. The constraints ensure that latency L_i does not exceed a maximum threshold $L_{\max}$, resource utilization U_i does not surpass a maximum capacity $U_{\max}$, and coverage C_i meets or exceeds a minimum required area $C_{\min}$.

3.2.2. Edge node configuration

Each edge node in the network is built with hardware and software parts. The hardware in an edge node includes a multi-core CPU that is designed for high-speed processing to handle heavy computing tasks, lots of RAM to manage several processes simultaneously, and enough storage space for storing and processing data locally. The storage includes solid-state drives for quick data access and hard disk drives for more storage space. Also, edge nodes have network interfaces that allow fast and low-latency communication, which is essential for real-time data processing and sending.

Each edge node uses a simple operating system designed for edge computing, usually a Linux-based system that is strong in networking, virtualization, and containerization. The software includes tools like hypervisors or container runtimes that allow NFV. The edge node can run NFV as separate virtual machines or containers. This virtualization layer is essential because it helps the SDN controller assign, move, or adjust the NFVs quickly, depending on network needs.

Resource virtualization is essential to how edge nodes work, allowing for the flexible use and management of physical resources. With virtualization, an edge node's CPU, memory, and storage can be divided into several virtual parts, each assigned to a specific task or NFV. This method helps use resources efficiently and adjust them as the workload changes. For example, CPU power is shared among NFVs based on their current needs, and the SDN controller adjusts this distribution to improve performance and reduce energy use.

The resource allocation process is modeled as a dynamic optimization problem, where the goal is to maximize the overall performance of the edge node while minimizing energy consumption and ensuring fair resource distribution among NFVs. Let R_{ij}^{CPU}, R_{ij}^{MEM}, and R_{ij}^{STOR} denote the CPU, memory, and storage resources allocated to NFV V_j on edge node E_i, respectively. The optimization objective is expressed as Eq. (9).

$$\max \sum_{j=1}^{k} \left(\alpha_j \cdot P_{ij}^{\text{perf}} - \beta_j \cdot E_{ij}^{\text{cons}} \right) \tag{9}$$

subject to Eq. (10).

$$\sum_{j=1}^{k} R_{ij}^{\text{CPU}} \leq C_i, \sum_{j=1}^{k} R_{ij}^{\text{MEM}} \leq M_i, \sum_{j=1}^{k} R_{ij}^{\text{STOR}} \leq S_i \tag{10}$$

where P_{ij}^{perf} is the performance metric (e.g., throughput, latency) of NFV V_j on edge node E_i, E_{ij}^{cons} is the energy consumption of V_j on E_i, and α_j and β_j are weighting factors that balance performance and energy efficiency. The constraints ensure that the total resources allocated to all NFVs on a node do not exceed the node's physical capacity.

This configuration allows the edge nodes to operate efficiently under varying load conditions, ensuring that resources are allocated where they are most needed and can be quickly reallocated in response to changes in demand. The combination of advanced hardware components and sophisticated software virtualization enables edge nodes to support the dynamic, distributed nature of edge computing, providing a flexible and scalable solution for processing the vast amounts of data generated by IoT devices in real-time. Algorithm 1 provides a clear procedural guide on allocating and optimizing resources in an edge computing environment managed by an SDN controller.

3.3. Resource allocation framework

The proposed resource allocation framework is an intelligent system that combines centralized SDN control with flexible, AI-based resource allocation methods. It uses DQN and NFV to improve how computational, storage and network resources are shared across a distributed edge computing network. This framework is built to handle the challenges of managing resources quickly in a changing and spread-out network, ensuring it remains fast and efficient and can quickly grow as needed.

3.3.1. Centralized SDN control

In this system, the SDN controller is essential because it manages the whole network from one central point. The SDN controller keeps track of what is happening across all edge nodes, like how much CPU and memory are used, how storage is utilized, the network delays, and the amount of data being processed. By watching these things closely, the SDN controller always knows the current state of the network. This is crucial for making smart, quick decisions about how to allocate resources effectively.

The SDN controller's decision-making is guided by an optimization function that balances different goals. These goals include minimizing

network latency *(L)*, reducing energy consumption I, and lowering operational costs *I*, while maximizing resource utilization *(U)*. The controller dynamically adjusts the allocation of resources to meet these objectives, responding to fluctuations in demand or network conditions with minimal delay.

The resource allocation decision at any time t described by the optimization problem as Eq. (11).

$$\min\{\lambda L(t) + \mu E(t) + \gamma C(t)\} - \eta U(t) \tag{11}$$

subject to the constraints as Eq. (12).

$$R_{i,j}(t) \le R_{\max}, L(t) \le L_{\max}, U(t) \le U_{\max} \tag{12}$$

where $R_{i,j}(t)$, represents the resources allocated to task *j* on edge node *I* at time *t*, $R_{\max}$ is the maximum resource capacity, and λ, μ, γ, and η are weights that control the relative importance of each objective.

The SDN controller gathers data from each edge node at frequent intervals or upon the occurrence of specific events, using this data to update its understanding of the network state. This data collection involves transmitting telemetry information from edge nodes, such as the current load, available resources, and network performance metrics. The controller analyzes this data using advanced analytical models, including machine learning techniques, to detect trends, predict future demand, and optimize resource allocation.

The SDN controller's predictive capability, a crucial innovation of the proposed framework, enables it to foresee increases in demand or potential bottlenecks. By adjusting resources ahead of time based on past data and real-time information, the controller proactively prevents performance issues, ensuring high performance even in the most dynamic and unpredictable environments.

Algorithm 1. Resource Allocation Optimization

Input	Set of edge nodes $E = E1, E2, \ldots, EN$
	Set of NFVs $V = V1, V2, \ldots, Vk$
	Set of IoT devices $D = D1, D2, \ldots, DM$
	Resource capacities of each edge node (CPU, Memory, Storage)
	Network latency thresholds L_{max}
	Resource utilization thresholds U_{max}
	Coverage area thresholds C_{min}
Output	Optimized resource allocation across edge nodes
	Begin
1	Initialize SDN Controller with **global network** view
2	**For** each edge node E_i **in** E do
	Calculate current resource utilization U_i
	Calculate current network latency L_i to connected IoT devices
	Calculate coverage area C_i **for** the node
3	End **for**
4	For each NFV V_j **in** V do
	Identify candidate edge nodes **for** deployment based on:
	Resource availability (CPU, Memory, Storage)
	Latency constraints ($L_i \le L_{max}$)
	Resource utilization constraints ($U_i \le U_{max}$)
	Coverage constraints ($C_i \ge C_{min}$)
	If multiple candidate nodes exist then
	Select edge node E_i that minimizes the objective function
	Minimize $(\lambda \cdot L_i + \mu \cdot (1 - U_i) + \nu \cdot (1 - C_i))$
	Allocate resources to NFV V_j on selected edge node E_i
	Update resource capacities of E_i
	Deploy NFV V_j on E_i
5	End **for**
6	Monitor network performance metrics **in** real-time
7	If network conditions change (e.g., increase **in** demand, changes **in** latency) then
	Re-evaluate resource allocation
	Migrate NFVs **as** necessary to maintain optimal performance
8	End If
9	Return optimized resource allocation configuration
	End

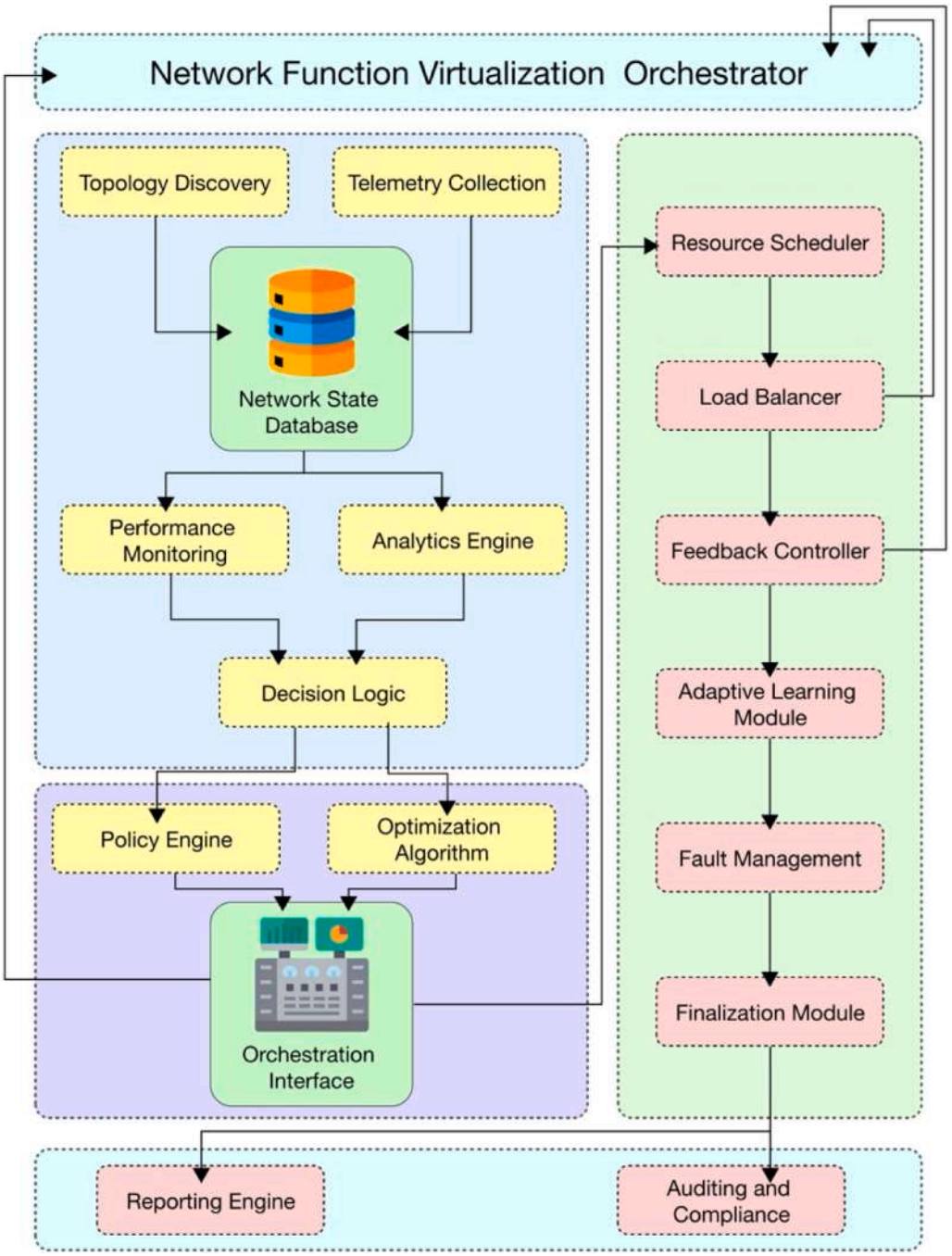

Fig. 2. The architecture of SDN Controller with Integrated NFV for Dynamic Resource Allocation.

Table 3
Network Parameters.

Parameter	Value
Number of Edge Nodes (N)	25
Network Topology	Mesh
Network Bandwidth (B_i)	1 Gbps
Baseline Network Latency (L_i)	10 ms
Data Transmission Rate (R_i)	200 Mbps

3.3.2. Dynamic resource allocation algorithms

Our model includes a dynamic resource allocation mechanism to account for the dynamic nature of resource availability and demand in real-world environments. This mechanism continuously monitors resource utilization across edge nodes and adapts allocations in real-time. For example, during periods of high demand, resources can be dynamically reallocated from underutilized nodes to those experiencing a surge in tasks. This ensures that the system maintains optimal performance despite fluctuating conditions.

Fig. 2 provides an in-depth view of the internal architecture of the SDN controller, integrated with NFV, designed for dynamic resource allocation in edge computing environments. It illustrates the interactions between critical components, such as the Policy Engine, Optimization Algorithm, and Resource Scheduler, showing how they work together to manage and allocate resources efficiently. The figure also emphasizes the role of real-time data monitoring, feedback loops, and adaptive learning in ensuring optimal performance, low latency, and efficient utilization of computational resources across the network.

Algorithm. Design

The algorithm addresses the resource allocation problem by modeling it as a Markov Decision Process (MDP). In this model, each state at time t represents the current network conditions. The action taken at that time is a specific decision on allocating resources, such as assigning CPU, memory, or bandwidth to a task or NFV. The immediate benefit from this action is the reward, which could include reduced latency, improved load distribution, or more efficient resource utilization. The goal of the DQN is to learn an optimal policy π\piπ that maximizes the cumulative reward over time as Eq. (13).

$$\pi^* = \underset{\pi}{\operatorname{argmax}} E\left[\sum_{t=0}^{T} \gamma^t r_t \middle| \pi\right] \tag{13}$$

where γ is a discount factor that balances the trade-off between immediate and future rewards. The DQN achieves this by approximating the Q-function, which represents the expected utility of taking action a_t in state s_t, its calculates by Eq. (14).

$$Q(s_t, a_t) = r_t + \gamma \underset{a'}{\max} Q(s_{t+1}, a') \tag{14}$$

The Q-values are updated iteratively as the DQN learns from the outcomes of previous allocation decisions, gradually improving its ability to predict the most effective resource allocations under varying conditions.

3.3.3. DQN integration

Including the DQN in the resource allocation framework offers several significant benefits. First, it helps the system learn from past experiences, improving resource management as new situations arise. Second, the DQN's ability to predict future needs allows the system to adjust resources, which helps prevent shortages or delays. Finally, the DQN can adapt to the specific needs of different tasks or applications, ensuring that resources are used best to improve performance for each situation. Algorithm 2 provides a comprehensive approach to dynamic resource allocation using DQN integration.

3.3.4. Load balancing and scaling

In addition to optimizing individual resource allocations, the framework includes load balancing and scaling mechanisms across the entire network. Load balancing helps ensure that no single edge node is too busy while others are not fully used. It does this by adjusting the distribution of tasks and network functions in real-time, depending on how busy each edge node is and it is capacity.

The load balancing problem is framed as minimizing the variance in resource utilization across all edge nodes as Eq. (15).

$$\min \frac{1}{N} \sum_{i=1}^{N} (U_i(t) - \overline{U}(t))^2 \tag{15}$$

where $U_i(t)$ is the resource utilization of edge node i at time t, and $\overline{U}(t)$ is the average utilization across the network. By reducing this difference, the framework ensures that the tasks are spread evenly, stopping any one node from slowing down the overall performance.

Resource scaling is another critical component of the framework. It allows the system to dynamically increase or decrease the resources allocated to a task or NFV in response to changes in demand. For example, during periods of high demand, the framework can scale up the CPU cycles, memory, or bandwidth allocated to a particular task, ensuring that performance remains consistent. Conversely, during periods of low demand, resources can be scaled down to conserve energy and reduce operational costs.

Scaling decisions are informed by the network's current state and predictions generated by the DQN. The framework can maintain optimal performance across various operating conditions by continuously monitoring and adjusting resource allocations, ensuring the network remains responsive, efficient, and scalable. Algorithm 3 outlines a detailed approach for load balancing and resource scaling in edge computing environments.

Table 4
Edge Node Parameters.

Parameter	Value
CPU Capacity (C_i)	8–32 cores, 2.0–3.0 GHz
Memory Capacity (M_i)	32–128 GB DDR4
Storage Capacity (S_i)	512 GB to 2 TB SSD
Power Consumption per CPU Cycle (P_i)	0.5 nJ per cycle
Edge Node Placement	Within 10 km radius of IoT devices
Resource Virtualization Overhead	5 % CPU and Memory overhead
Network Interface Bandwidth	1 Gbps to 10 Gbps
Number of NFVs hosted on each Edge Node	Up to 10 NFVs
NFV Initialization Time	2–5 s
NFV Migration Time	500 ms to 1 s
Edge Node Failure Rate	0.01 failures per day
Cooling Efficiency	90 % (10 % heat dissipation loss)
Edge Node Startup Time	30 s

Algorithm 2. Dynamic Resource Allocation with DQN Integration

Input	S: Current states (network conditions, resource utilization, etc.) A: Set of possible actions (resource allocation decisions) Q: Q-value function (initialized randomly **or** based on prior knowledge) γ: Discount factor (**0** < γ < **1**) α: Learning rate (**0** < α ≤ **1**) ε: Exploration probability (**0** < ε ≤ **1**) max_iterations: Maximum number of iterations **for** training
Output	π*: Optimal policy **for** resource allocation Q*: Updated Q-value function
	Initialize:
1	$Q(s, a)$ ← arbitrary values **for all** $s \in S$, $a \in A$
2	Set iteration count ← **0**
2	Loop until convergence **or** max_iterations:
3	For each time step t:
4	Observe current state $s_t \in S$
5	With probability ε, select a random action $a_t \in A$ (exploration)
6	Otherwise, select $a_t \leftarrow argmax_a Q(s_t, a)$ (exploitation)
7	Execute action a_t
8	Observe reward r_t **and next** state s_{t+1}
9	Update Q-value **for** the state-action pair (s_t, a_t): $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_t + \gamma max'_a Q(s_{t+1}, a') - Q(s_t, a_t)]$
10	Update current state: $s_t \leftarrow s_{t+1}$
11	Increment iteration count
12	If s_{t+1} **is** a terminal state **or** max_iterations reached:
13	Break the loop
14	Reduce ε (e.g., ε ← ε * decay_rate) to gradually reduce exploration

3.4. Performance metrics and evaluation criteria

To provide a comprehensive evaluation of the proposed framework, we have selected a range of interconnected performance metrics. For instance, CPU utilization directly influences energy consumption, as higher utilization typically leads to increased power draw. Similarly, memory and storage utilization can impact network latency, as data bottlenecks at storage points may delay processing times. These measures are chosen to understand how well the system works in terms of efficiency, quick response, and overall performance in different areas of edge computing tasks. The measures are grouped into categories that show how healthy resources are used, how the network performs, how energy-efficient the system is, and how tasks are managed.

3.4.1. CPU, memory, and storage utilization

To evaluate resource usage efficiency within the edge nodes, CPU, memory, and storage utilization are critical metrics. CPU utilization U_{CPU} is measured as the percentage of CPU cycles consumed relative to the total available cycles on an edge node. Memory utilization U_{MEM} and storage utilization U_{STOR} are similarly defined as the ratio of used memory and storage to the total available memory and storage, respectively. These measurements are collected continuously across all edge nodes using monitoring tools such as Prometheus or custom telemetry systems integrated into the SDN controller.

The main goal of improving these usage measures is to find a balance between using too little and too much. Using too little, or underutilization, happens when resources are available but not used well, leading to waste and inefficiency. On the other hand, using too much, or over-provisioning, occurs when more resources are used than needed, which can cause extra costs and more energy use. Optimization aims to ensure resources are used just right, avoiding wastage from underutilization and unnecessary expense from over-provisioning. This helps keep the network running efficiently.

The objective is expressed as minimizing the deviation from optimal utilization as Eq. (16).

$$[\min\left\{\sum_{i=1}^{N}\left(\left|U_{\mathrm{CPU},i}-U_{\mathrm{opt,CPU}}\right|+\left|U_{\mathrm{MEM},i}-U_{\mathrm{opt,MEM}}\right|+\left|U_{\mathrm{STOR},i}-U_{\mathrm{opt,STOR}}\right|\right)\right\}] \tag{16}$$

where $U_{\mathrm{opt,CPU}}$, $U_{\mathrm{opt,MEM}}$, and $U_{\mathrm{opt,STOR}}$ are the optimal utilization levels for CPU, memory, and storage, respectively.

Algorithm 3. Load Balancing and Resource Scaling.

Input	N: Number of edge nodes T: Set of tasks to be allocated across edge nodes U: Current resource utilization of each edge node R: Available resources on each edge node (CPU, memory, bandwidth) U_{max}: Maximum allowable resource utilization per node U_{min}: Minimum resource utilization threshold to trigger scaling down D: Demand (resource requirements) **for** each task **in** T
output	Task allocation plan (which task goes to which edge node) Updated resource allocations per node after scaling
	Initialize
1	Allocate tasks to edge nodes based on current resource utilization.
2	Compute the initial utilization cap U sub i for each edge node I (I = 1 to N)U_i
3	Loop until **all** tasks are allocated
4	For each task t **in** T:
5	Find the edge node I **with** the minimum U_i
6	If $U_i + D_t \leq U_{max}$:
7	Allocate task t to edge node i
8	Update U_i: $U_i \leftarrow U_i + D_t$
9	Else:
10	Find an alternative node j **with** $U_j + D_t \leq U_{max}$
11	If found, allocate task t to edge node j **and** update U_j
12	If no suitable node **is** found:
13	Trigger resource scaling up on node i:
14	Increase resources R_i to accommodate D_t
15	Allocate task t to node i
16	Update U_i: $U_i \leftarrow U_i + D_t$
17	End **for**
18	Check **for** Load Imbalance **and** Scale Down
19	For each edge node I *(I = **1** to N)*:
20	If $U_i < U_{min}$:
21	Evaluate tasks on node I **for** possible migration
22	For each task t on node i:
23	Find a node j where $U_j + D_t \leq U_{max}$
24	Migrate task t to node j
25	Update U_i **and** U_j
26	If **all** tasks can be migrated:
27	Scale down resources on node I to conserve energy.

3.4.2. Network performance metrics

Network performance is a critical factor in the edge computing framework's overall efficiency, particularly in latency and throughput. Latency is measured as the end-to-end delay experienced by data packets as they travel from IoT devices to the edge nodes and onward to the SDN controller. Latency *(L)* is a crucial metric for time-sensitive applications such as real-time analytics and autonomous systems. The latency measurement involves capturing the packets' round-trip time (RTT) and averaging these measurements over multiple intervals to ensure accuracy. This latency is further broken down into transmission delay, processing delay at the edge node, and any possible queuing delays.

The optimization goal for latency is to minimize it, ensuring that data packets are processed and transmitted as quickly as possible. The latency optimization is expressed as Eq. (17).

$$\min\left\{\sum_{i=1}^{N} L_i\right\} \tag{17}$$

3.4.3. Energy efficiency

Energy per Task is calculated as the energy consumed to process a single task on an edge node. This metric is essential for understanding the efficiency of the resource allocation framework, particularly in environments where energy consumption must be minimized. The energy consumed by a task E_{task} calculated as Eq. (18).

Table 5
Workload Parameters.

Parameter	Value
Number of IoT Devices	100
Task Arrival Rate	10 tasks per second
Task Size	1 MB
Task Computational Complexity	500 Million Instructions (500 MIPS)
Task Priority Levels	3 Levels (Low, Medium, High)
Task Deadlines	100–500 ms
Task Inter-arrival Time	100 ms
Task Execution Time	50 ms
Task Resource Requirements (CPU, Memory, Storage)	2 CPU cores, 512 MB memory, 100 MB storage
Task Mobility (Frequency of Task Migration between Nodes)	5 % of tasks per minute
Task Dependency Levels	Medium (Tasks have moderate dependencies)
Task Execution Order	Sequential with occasional parallel tasks

Table 6
DQN Algorithm Parameters.

Parameter	Value
State Space Dimensions	20 dimensions
Action Space Dimensions	10 possible actions
Reward Function Parameters	Weighted sum of latency, energy, and utilization metrics
Learning Rate α	0.001
Discount Factor γ	0.99
Exploration Rate ε	1.0 (initial)
Exploration Decay Rate	0.995 per episode
Batch Size	64 samples per update
Replay Buffer Size	10,000 experiences
Number of Training Episodes	10,000 episodes
Maximum Steps per Episode	200 steps
Target Network Update Frequency	Every 100 steps
Initial Q-values	Initialized to 0
Experience Replay Strategy	Prioritized Experience Replay
Reward Clipping	[−1, 1]
Neural Network Architecture (Layers, Neurons)	3 layers: 128, 64, 32 neurons
Activation Functions	ReLU (Rectified Linear Unit)
Optimizer Type	Adam optimizer
Loss Function	Mean Squared Error (MSE)

$$E_{\text{task},ij} = P_i \cdot \frac{C_{ij} \cdot W_{ij}}{C_i} \tag{18}$$

where P_i is the power consumption per CPU cycle at the edge node E_i, and C_{ij} and W_{ij} are the computational complexity and workload factors of the task T_{ij}, respectively.

The framework aims to minimize the energy per task while meeting performance requirements. This is part of a broader strategy of System-Wide Energy Optimization, which seeks to reduce the total energy consumption across all edge nodes. This objective is modeled as Eq. (19).

$$\min\left\{\sum_{i=1}^{N}\sum_{j=1}^{m_i} E_{\text{task},ij}\right\} \tag{19}$$

where m_i is the number of tasks processed by an edge node E_i.

3.4.4. Task management metrics

Task Handling Capacity refers to the maximum number of tasks the system can handle simultaneously without significant performance degradation. This capacity C_{task} is measured by gradually increasing the number of concurrent tasks in the system and observing the point at which performance metrics such as latency and throughput begin to deteriorate. The goal is to maximize the task-handling capacity while maintaining acceptable levels of other performance metrics.

These metrics comprehensively evaluate the proposed resource allocation framework's performance and efficiency. By carefully monitoring and optimizing these metrics, the system can balance performance, resource utilization, energy efficiency, and reliability, making it well-suited for the demands of modern edge computing environments.

4. Simulation setup and parameters

This section describes the setup and parameters used for the simulation. The simulation was created using Python. We used Mininet to imitate the network and TensorFlow to implement DQN. The simulation setup includes 25 edge nodes arranged in a Mesh topology, which helps in flexible data movement. Each node has a bandwidth of 1 Gbps, and the standard network delay is set to 10 ms. This low delay is perfect for real-time applications. The data is sent at a rate of 200 Mbps. Table 3 presents the network parameters we used for testing the new framework.

In the simulation described in Table 4, edge nodes are equipped with varying hardware capabilities, accurately reflecting the heterogeneous nature of real-world environments. For example, some nodes may be configured with an 8-core CPU running at 2.0 GHz, while others might have up to 32 cores operating at 3.0 GHz. Memory capacities range from 32 GB to 128 GB of DDR4, and storage capacities vary between 512 GB and 2 TB of SSD, ensuring that the nodes can efficiently handle data processing tasks. Each CPU cycle consumes approximately 0.5 nJ of energy, indicating high energy efficiency across the nodes. The edge nodes are strategically placed within a 10 km radius of IoT devices, providing robust network coverage and minimizing response times. Virtualization overheads are kept low, around 5 %, slightly impacting CPU and memory usage. Network interface bandwidths range from 1 Gbps to 10 Gbps, facilitating fast data transfers. Each node can support up to 10 NFVs, with initialization times ranging from 2 to 5 s and migration times between 500 ms and 1 s. Designed for reliability, these nodes exhibit a low failure rate of 0.01 failures per day, achieve a cooling efficiency of 90 %, and have a startup time of 30 s.

Table 5 presents the workload parameters for testing the proposed resource management system. This test was done in a setup that imitates actual conditions, using 100 IoT devices. Each device creates tasks at a constant rate of 10 tasks every second. The tasks are not too simple; they handle 1 MB of data and need to perform 500 million instructions, also known as 500 MIPS. These tasks are sorted into three groups based on their urgency—Low, Medium, or High priority. The tasks have deadlines that vary from 100 ms to 500 ms. The time between when tasks start is set at 100 ms, and each task takes 50 ms to complete. Each task uses specific resources, specifically 2 CPU cores, 512 MB of memory, and 100 MB of storage. The setup also includes changes to make it more like real-life conditions, such as moving about 5 % of tasks between different network locations every minute. This introduces movement in the workload. Generally, the tasks are moderately dependent on each other and are done in order, although some need to be processed simultaneously by different resources.

Table 6 details the settings of the DQN algorithm. The algorithm

Table 7
Evaluation Metrics.

Parameter	Value
CPU Utilization (U_{CPU})	70–90 %
Memory Utilization (U_{MEM})	60–80 %
Storage Utilization (U_{STOR})	50–75 %
Network Latency L_i	10–30 ms
Throughput *(T)*	200 tasks per second
Energysub-taskk (E_{task})	0.5 Joules per task
Task Handling Capacity C_{task}	1000 concurrent tasks
Task Success Rate	99 %
Task Drop Rate	1 %
Network Congestion Level	20 % utilization threshold
Task Queue Length	Maximum 50 tasks per queue
Edge Node Load Balancing Efficiency	90 % (variance in utilization)
NFV Resource Allocation Efficiency	85 % resource utilization efficiency

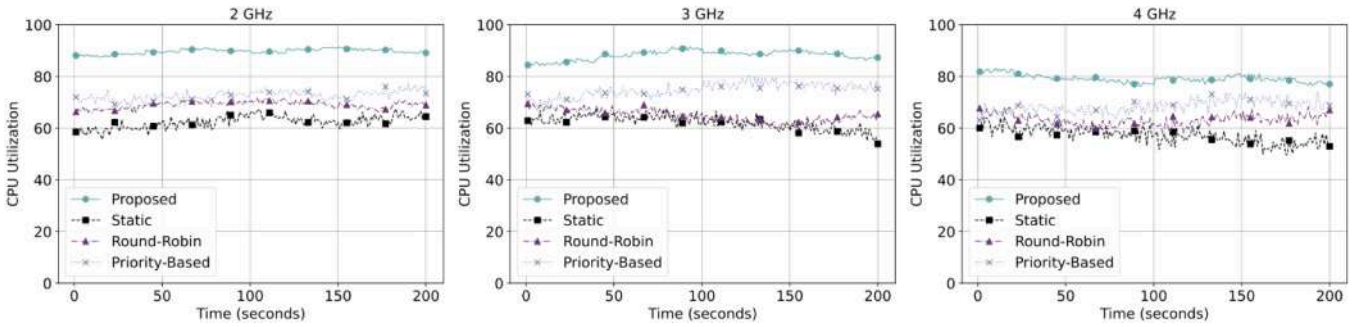

Fig. 3. CPU Utilization Comparison across Scheduling Algorithms at Different Frequencies.

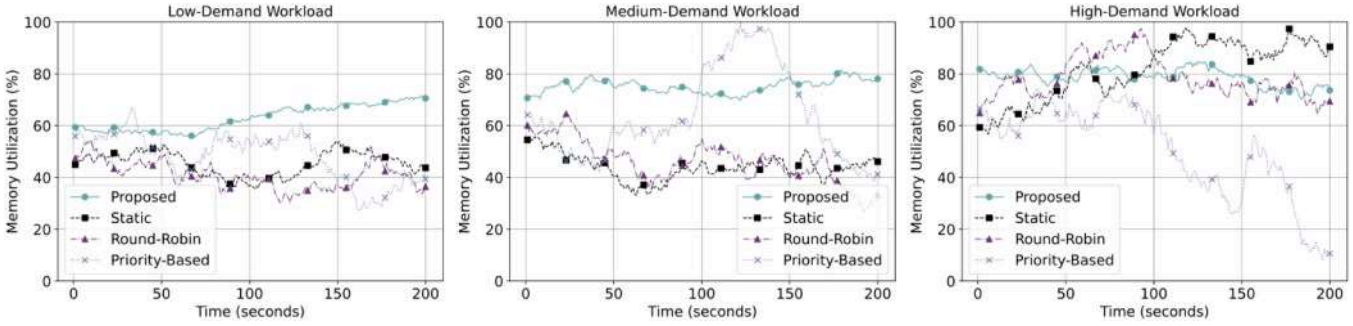

Fig. 4. Memory Utilization Analysis for Different Scheduling Algorithms Under Various Loads.

examines a state space of 20 dimensions and can choose from 10 different actions, giving it many options for decision-making. It uses a reward function that combines delay, energy use, and utilization measures to guide its actions toward achieving a balanced performance. The learning rate is set at 0.001, with a discount factor of 0.99, which helps it focus on long-term benefits. The initial rate at which the algorithm explores different actions is 100 %, decreasing by 0.5 % after each training episode, encouraging the algorithm to try new things early on. It learns by reviewing batches of 64 samples and keeps 10,000 past actions to learn from. The algorithm trains through 10,000 episodes, each limited to 200 steps and updates its target network every 100 steps. It starts with Q-values set at zero and uses a method that prioritizes learning from more important past actions. It limits the rewards to values between −1 and 1. For learning, the algorithm uses a network of three layers with 128, 64, and 32 neurons respectively, employs ReLU functions to activate neurons, uses the Adam method for optimization, and calculates

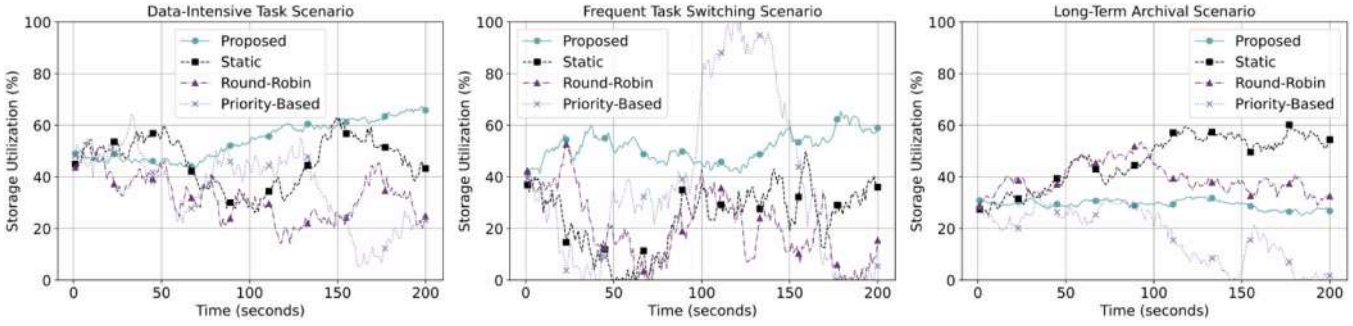

Fig. 5. Storage Utilization Comparison for Different Scheduling Algorithms and Tasks.

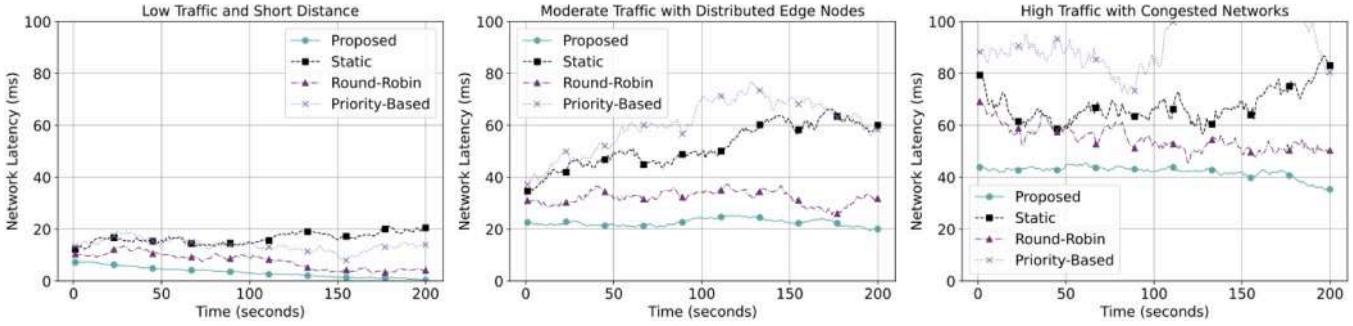

Fig. 6. Comparison of Network Latency Across Traffic Levels.

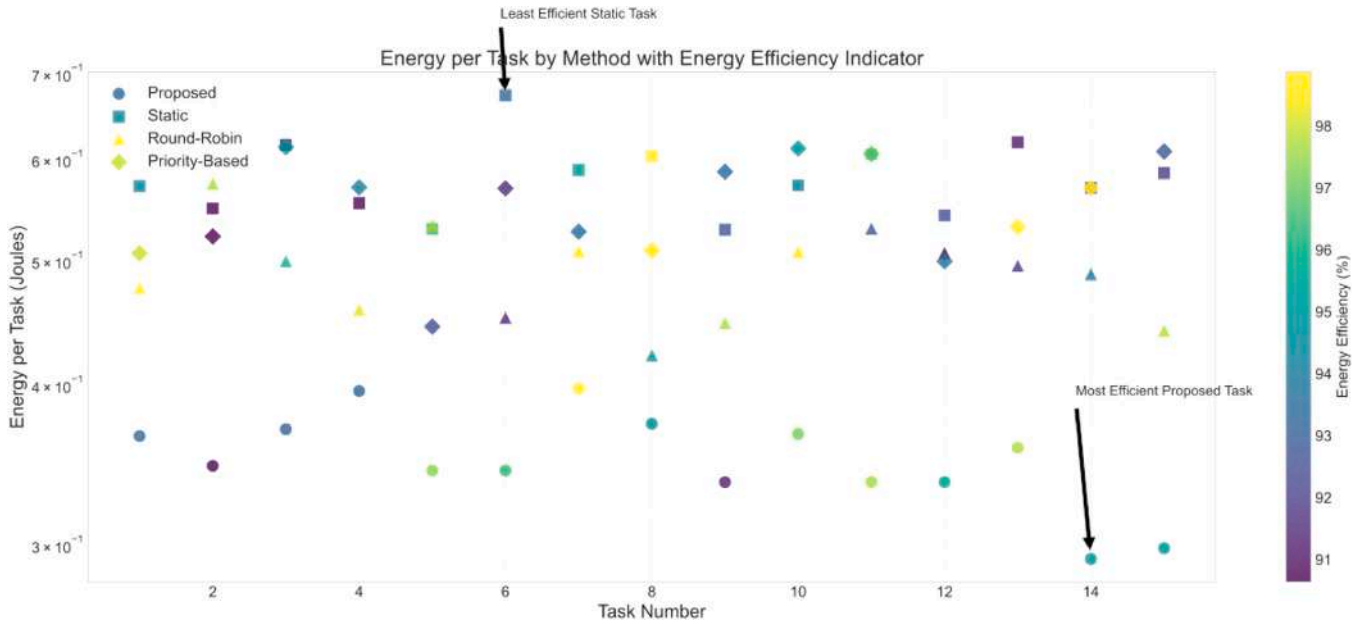

Fig. 7. Energy Efficiency in Task Performance Compared to Alternatives.

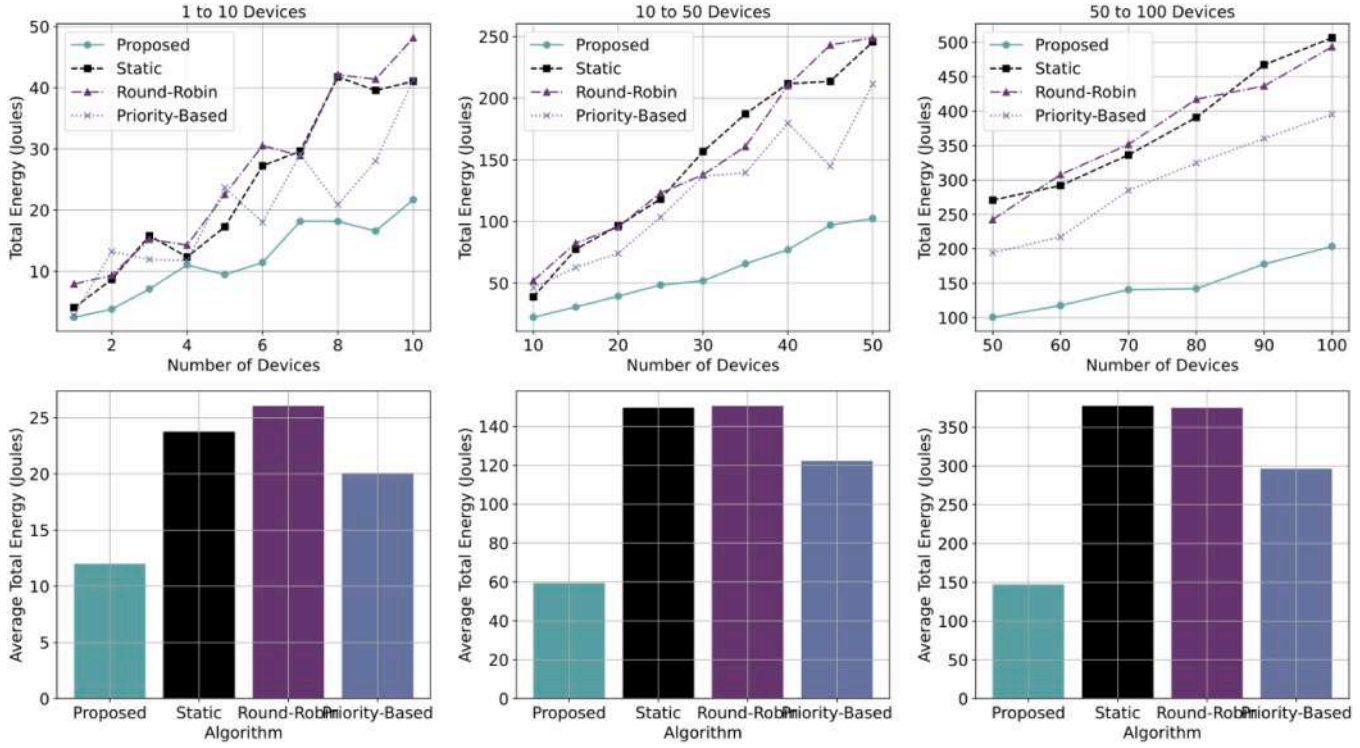

Fig. 8. Total Energy Consumption in Different Settings.

Table 8
The statistical analysis of energy consumption (J) for the algorithms in a network with 1–10 IoT devices.

	Proposed Method	Static Method	Round-Robin Method	Priority-Based Method
Min	2.50E+ 00	4.07E+ 00	7.93E+ 00	2.80E+ 00
Max	2.17E+ 01	4.18E+ 01	4.82E+ 01	4.13E+ 01
Mean	1.20E+ 01	2.38E+ 01	2.61E+ 01	2.01E+ 01
50 %	1.12E+ 01	2.23E+ 01	2.57E+ 01	1.95E+ 01
Std.	6.50E+ 00	1.40E+ 01	1.45E+ 01	1.10E+ 01

Table 9
The statistical analysis of energy consumption (J) for the algorithms in a network with 10–50 IoT devices.

	Proposed Method	Static Method	Round-Robin Method	Priority-Based Method
Min	2.23E+ 01	3.89E+ 01	5.21E+ 01	4.64E+ 01
Max	1.02E+ 02	2.46E+ 02	2.49E+ 02	2.12E+ 02
Mean	5.95E+ 01	1.50E+ 02	1.51E+ 02	1.22E+ 02
50 %	5.19E+ 01	1.57E+ 02	1.38E+ 02	1.37E+ 02
Std.	2.83E+ 01	7.07E+ 01	7.09E+ 01	5.51E+ 01

Table 10
The statistical analysis of energy consumption (J) for the algorithms in a network with 50–100 IoT devices.

	Proposed Method	Static Method	Round-Robin Method	Priority-Based Method
Min	1.01E+ 02	2.71E+ 02	2.43E+ 02	1.95E+ 02
Max	2.04E+ 02	5.07E+ 02	4.93E+ 02	3.95E+ 02
Mean	1.47E+ 02	3.77E+ 02	3.75E+ 02	2.96E+ 02
50 %	1.42E+ 02	3.64E+ 02	3.85E+ 02	3.05E+ 02
Std.	3.80E+ 01	9.52E+ 01	9.19E+ 01	7.93E+ 01

losses using the Mean Squared Error method.

The performance indicators shown in Table 7 were crucial for evaluating how well the resource allocation system worked in the simulation. The system kept CPU usage between 70 % and 90 %, which is good because the system was powerful without overloading. Memory use was between 60 % and 80 %, and storage use was kept from 50 % to 75 %. This balance helps prevent using too many resources. Network delay, significant for applications that need immediate responses, stayed between 10 ms and 30 ms. The system could handle 200 tasks every second, using about 0.5 Joules of energy. It could also manage up to 1000 tasks simultaneously, completing 99 % and only failing to complete 1 %. The network did not get too crowded; it used up to 20 % of its capacity, and there were never more than 50 tasks in line. The efficiency of spreading tasks across different edge nodes was 90 %, which helped keep the use of resources even, while the efficiency of assigning resources for network functions virtualization was 85 %.

The following section will present the simulation results and analyze

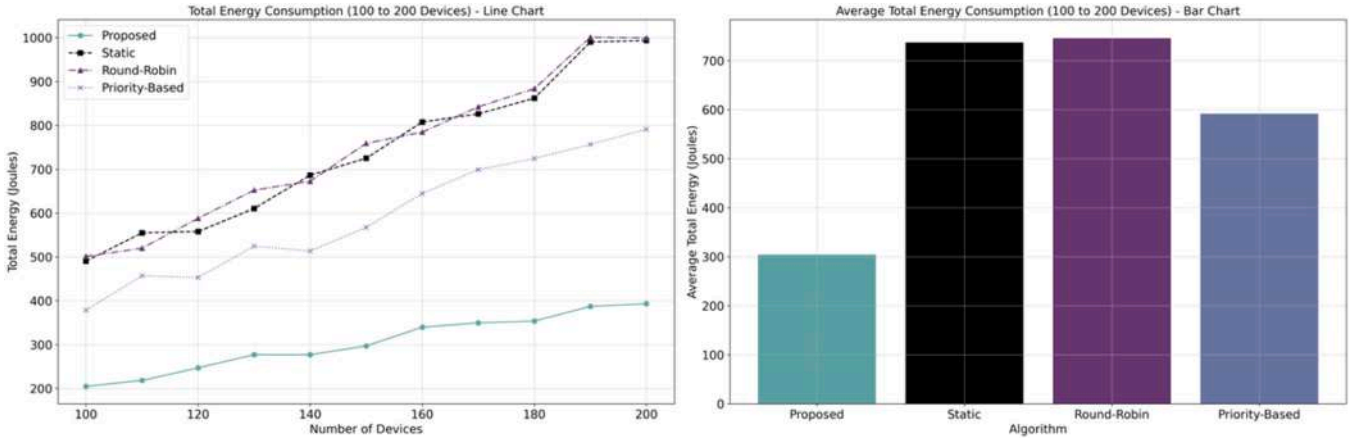

Fig. 9. Energy Consumption Analysis in Extended Edge Computing Deployments.

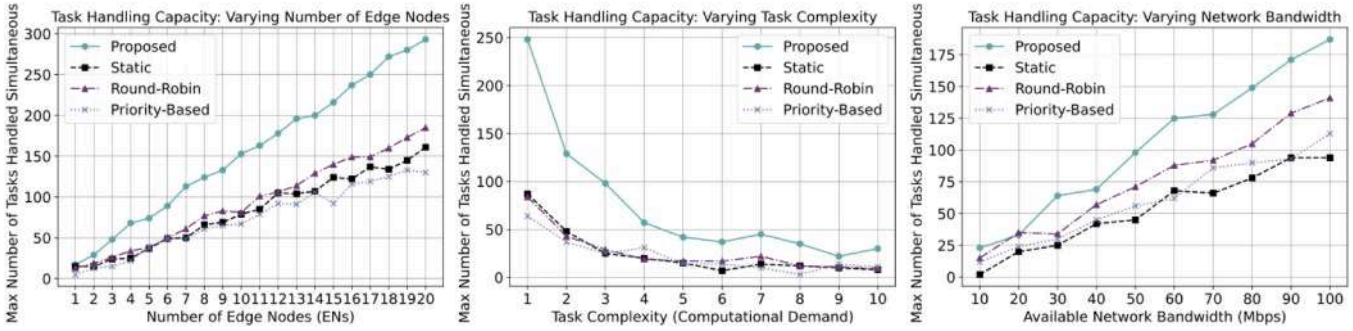

Fig. 10. Task Capacity Variability with Edge Nodes, Complexity, and Bandwidth.

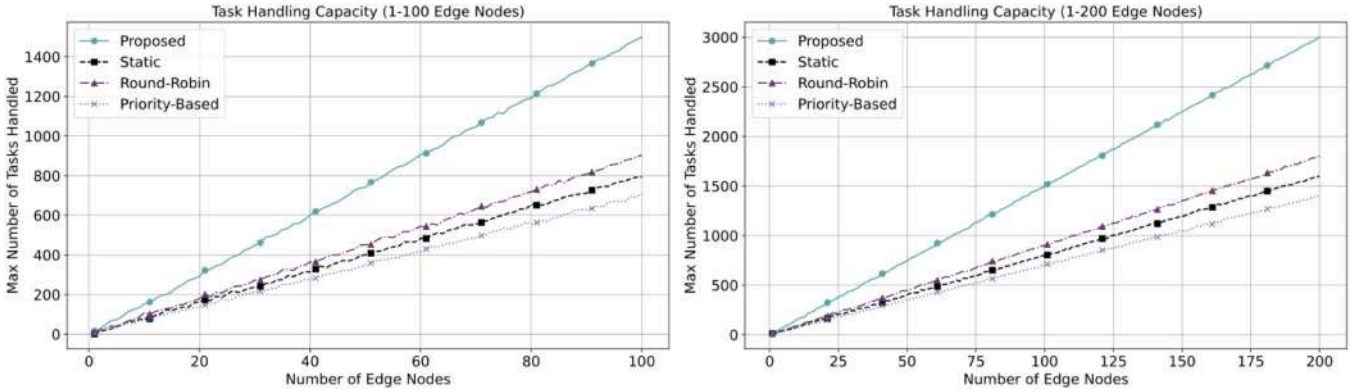

Fig. 11. Scalability Analysis of Task Handling Capacity in Edge Computing Environments.

the proposed resource allocation framework's performance based on the evaluation metrics.

5. Simulation results and discussion

In this section, we present and examine the results from our simulation, evaluating and discussing our study's performance outcomes. We conducted a series of verification steps to ensure the accuracy and reliability of our simulation results. These included cross-validating simulation outputs with theoretical predictions and sensitivity analyses to assess the impact of parameter variations on performance outcomes.

Fig. 3 illustrates the comparative analysis of CPU utilization across three different frequencies (2 GHz, 3 GHz, and 4 GHz) for four distinct scheduling algorithms: proposed, static, Round-Robin, and Priority-Based. At 2 GHz, the proposed algorithm achieves a 40.73 % improvement over the static algorithm, 29.28 % over Round-Robin, and 24.75 % over Priority-Based, highlighting its superior efficiency in resource management. The improvements are even more pronounced at 3 GHz, where the proposed algorithm surpasses static by 46.27 %, Round-Robin by 43.25 %, and Priority-Based by 17.24 %, indicating its exceptional adaptability at this frequency. At 4 GHz, although overall utilization decreases, the proposed algorithm continues to outperform with a 41.23 % improvement over static, 25.31 % over Round-Robin, and 11.00 % over Priority-Based.

The CPU's improved performance with the proposed algorithm comes from its innovative way of allocating resources, which uses up-to-date information about network conditions and the nature of the tasks being handled. This algorithm includes DQN learning methods that help predict the best places to assign tasks. This ensures that the workload on CPUs is evenly distributed among all the edge nodes, which helps reduce delays in processing data and increases the system's overall efficiency. This is especially helpful when the amount of computing needed can change often.

According to Fig. 4, the proposed scheduling algorithm performs better than other methods in managing memory across different levels of workload demand. It shows stable and effective memory use, based on how intense the workload is, with the best performance seen in situations of high demand where it maintains memory use between 70 % and 90 %. On the other hand, the static and Round-Robin methods have lower and more inconsistent memory use, especially under high demand, which may suggest they are not as efficient. The Priority-Based method sometimes works well but often shows significant changes in performance, indicating it might not handle changing workloads smoothly.

The proposed method's excellent memory performance comes from intelligent planning strategies, considering the current and expected workload needs. By using AI to make decisions, it actively adjusts the memory needed, avoiding situations where too little or too much memory is used. This helps the system manage different workload levels efficiently, ensuring it works well and uses resources effectively.

Fig. 5 shows how different scheduling algorithms use storage for various tasks, such as data-intensive tasks, frequent task switching, and long-term archival. The proposed algorithm consistently achieves higher, stable, and more efficient storage utilization. This is particularly evident during the data-intensive task scenario, where it effectively balances load and minimizes fluctuations. The proposed method adapts well to the frequent task-switching scenario, maintaining steady utilization. In contrast, the static and Round-Robin methods struggle with efficiency, showing lower and more inconsistent usage patterns. The Priority-Based method, although sometimes capable of high utilization, displays significant volatility, especially in the long-term archival scenario, where its performance is less reliable.

The proposed method constantly checks storage needs and usage patterns and adjusts resource allocation to fit current needs while also planning for future demands. This helps avoid storage problems and makes data processing more efficient, improving overall system performance.

In Fig. 6, the proposed method improves network performance by reducing latency in different traffic situations. The most significant improvement is seen in low traffic conditions, where the proposed algorithm reduces latency by up to 79.49 % compared to the static algorithm. In moderate traffic scenarios, it still performs well, reducing latency by more than 62 % compared to the Priority-Based scheduling method. Even in high traffic conditions, the proposed algorithm remains effective, consistently outperforming other methods with reductions of 37.45 % over static and 55.67 % over Priority-Based. This shows that the proposed algorithm is very good at handling different levels of network traffic, making it better than the alternatives in all scenarios.

The proposed framework reduces network latency mainly because it uses SDN-controlled routing and AI-driven resource allocation. By keeping an overall network view, the framework can smartly direct data through the best routes and allocate enough network resources to essential tasks. The DQN part helps predict and avoid possible traffic jams, ensuring data moves smoothly and quickly through the network.

The energy used for each task by different methods is depicted in Fig. 7, along with energy efficiency percentages. The data show that the proposed method always uses less energy for each task, making it more efficient, especially in the early tasks where energy use is much lower than other methods. The static method is the least efficient, significantly using the most energy for each task as the functions increase. The Round-Robin and Priority-Based methods perform moderately, using energy between the proposed and static methods. The energy efficiency percentages highlight that the proposed method is much better than the static method, which falls behind.

Moreover, the analysis of total energy use in Fig. 8 for IoT devices shows that the proposed scheduling algorithm is highly efficient. Small setups with 1–10 devices use the least energy, about 12.00 Joules on average, much lower than other methods, as seen in Table 8. The proposed method uses less energy for setups with 10–50 devices, averaging 59.5 Joules, as shown in Table 9. It remains the most energy-efficient in larger setups with 50–100 devices, with an average use of 147 Joules, as mentioned in Table 10. This high energy efficiency is because the proposed framework smartly allocates and scales resources, using computational and network resources effectively without waste. The DQN's predictive abilities help make proactive resource use adjustments, aligning energy consumption with actual workload needs. Efficient load balancing and task scheduling reduce unnecessary processing and idle times, saving energy.

Fig. 9 shows the line and bar chart for total energy consumption across an extended range of 100–200 devices. The total energy (in Joules) increases as more devices are added. Yet, our proposed scheduling framework consistently consumes less energy than the static, Round-Robin, and Priority-Based methods. This trend indicates that the proposed approach is more energy efficient even as system scale grows.

Fig. 10 shows how different scheduling algorithms handle tasks under various conditions, like the number of edge nodes, task difficulty, and network bandwidth. The proposed algorithm performs better than the others in all situations. It does exceptionally well when there are more edge nodes and the tasks are more complex. For example, in Scenario 1, where the number of edge nodes changes, the proposed algorithm handles 74 tasks with five edge nodes, while the static algorithm only handles 37, Round-Robin 38, and Priority-Based 39. In Scenario 2, where the task complexity changes, the proposed algorithm handles 248 tasks at the most accessible level, much more than the static algorithm (87 tasks), Round-Robin (84 tasks), and Priority-Based (64 tasks). In Scenario 3, with different network bandwidths, the proposed algorithm handles 98 tasks at 50 Mbps, compared to 45 tasks by static, 71 by Round-Robin, and 56 by Priority-Based.

The proposed framework handles tasks better because it has advanced ways to manage resources and schedule tasks. It can adjust quickly to changes in how the system is working. The framework ensures

fast and efficient processing by evenly distributing tasks across different edge nodes and smartly prioritizing tasks based on their complexity or urgency. SDN and NFV technologies help the system provide flexible and scalable resources, allowing it to handle different workloads and network situations easily.

While the proposed framework has demonstrated significant simulation improvements, validating these results in real-world environments is essential.

The scenario with 1–100 edge nodes, the line chart clearly shows that our proposed framework consistently outperforms traditional scheduling methods. As the number of edge nodes increases, the maximum number of tasks handled rises steadily. Extending the analysis to 1–200 edge nodes, the performance gap between our proposed framework and the traditional approaches widens considerably. The line chart reveals that while all methods benefit from increased resources, the proposed method scales significantly better, handling many tasks. These analyses are shown in Fig. 11. Together, these analyses confirm the robust scalability of our SDN-NFV-based resource allocation framework. The simulation results indicate that the DQN-based scheduling mechanism adapts well to increasing network sizes and leverages the additional edge resources to achieve higher task handling capacities.

Our study focuses on a simulation-based evaluation of SDN-NFV-driven resource allocation in edge computing and provides a controlled environment to analyze performance under varying workloads. Simulations allow us to test different configurations, optimize scheduling policies, and measure improvements in CPU utilization, memory efficiency, and network performance. However, real-world deployment can be significantly more complex due to hardware limitations, network variability, and security risks, which are difficult to fully replicate in a simulated environment.

To address these and make our simulation more reflective of real-world conditions, we incorporate several key adaptations. First, we modelled hardware heterogeneity by configuring edge nodes with varying CPU capacities (8–32 cores), memory (32–128 GB), and network bandwidth (1 Gbps to 10 Gbps) to assess the impact of resource constraints on performance. Second, we introduce dynamic network conditions by varying network latencies (10–30 ms), fluctuating bandwidth, and task migrations (5 % per minute) to simulate unpredictable traffic loads and offloading decisions. Third, we accounted for energy efficiency by integrating power-aware scheduling mechanisms, where each CPU cycle consumes 0.5nJ, ensuring energy-efficient task execution. Additionally, our framework includes real-time workload variations with 100 IoT devices generating diverse tasks (500 MIPS) at different priority levels, simulating fluctuating demand patterns in a real deployment.

Despite these adaptations, further challenges remain, including hardware limitations, security vulnerabilities, scalability constraints, and interoperability issues, which must be carefully addressed when transitioning to real-world implementations. The following paragraph discusses these challenges in detail.

One of the most important challenges in real world is edge nodes have constrained computational resources compared to traditional cloud servers. The efficiency of workload scheduling depends on the processing power, memory, and storage capacity of the deployed edge nodes. Limited hardware may lead to resource contention, affecting performance in real-world settings. Unlike controlled simulation environments, real-world networks experience latency, bandwidth, and packet loss fluctuations. Dynamic changes in network conditions can impact the efficiency of resource allocation and workload offloading strategies. Adaptability to these variations is crucial for maintaining low latency and high throughput. Deploying NFVs at the edge introduces potential vulnerabilities, such as unauthorized access, data breaches, and Distributed Denial-of-Service (DDoS) attacks. Secure authentication, encryption, and intrusion detection mechanisms are essential to protect edge computing environments from cyber threats. The heterogeneous nature of edge computing environments, where nodes differ in hardware specifications and connectivity, makes resource management complex. A scalable framework should dynamically adjust to the growing number of IoT devices and fluctuating workloads while ensuring optimal performance. Real-world edge deployments must balance energy consumption with computational efficiency. Unlike simulations where power constraints may be relaxed, practical implementations require energy-aware scheduling mechanisms to optimize battery life in mobile and resource-constrained edge nodes. Different vendors provide edge computing hardware and SDN/NFV solutions, leading to interoperability challenges. Standardized interfaces and protocols are necessary to ensure seamless integration across diverse edge nodes and SDN controllers.

Moreover, security is a major concern in edge computing, especially when handling IoT data across distributed nodes. Unlike centralized cloud systems, edge nodes are more exposed to unauthorized access, data breaches, and DDoS attacks, making them prime targets for cyber threats. In SDN-NFV-based architectures, malicious network functions could be deployed, or attackers might intercept communication between SDN controllers and edge nodes. These vulnerabilities could lead to service disruptions, data manipulation, or resource exhaustion, impacting overall system performance. Without strong authentication, encryption, and anomaly detection, the risk of security breaches increases significantly. To mitigate these risks, several security mechanisms can be integrated into the framework. Multi-factor authentication (MFA) and role-based access control (RBAC) can help prevent unauthorized access, while TLS/SSL encryption ensures secure communication between system components. Deploying AI-driven intrusion detection systems (IDS) can identify suspicious activities, such as abnormal resource usage or unauthorized NFV deployments. Additionally, SDN-based traffic filtering and rate-limiting techniques can help mitigate DDoS attacks by dynamically adjusting network rules in real time. To secure NFV deployments, sandboxing, cryptographic verification, and blockchain-based authentication can be used to validate and isolate virtual functions before deployment.

6. Conclusion and future works

This paper introduced a new resource allocation system for edge computing environments that uses dynamic scheduling algorithms and DQN optimization to improve resource use and performance. Our results showed significant improvements: CPU usage increased by up to 46 %, memory usage enhanced by 30 %, storage usage went up by 20 %, network latency decreased by 79 %, and energy use per task dropped by about 50 % compared to traditional methods. While our study successfully balances goals like performance, energy efficiency, and resource use in stable and changing network environments, some limitations remain. The current system does not fully address security, essential for protecting edge computing systems from threats. Also, even though our system is designed to adapt, its ability to handle extensive deployments with many edge nodes and IoT devices has not been thoroughly tested. Moreover, in the study's first phase, the proposed method has not been compared to more sophisticated and state-of-the-art algorithms. Future work will solve these issues by including dynamic network conditions, improving security, and testing the system in large-scale edge computing environments compared to more complex algorithms.

Funding

The result was solving the standard project "Artificial intelligence and deep learning" using institutional support for the long-term conceptual development of the University of Finance and Administration's research.

CRediT authorship contribution statement

Parisa Khoshvaght: Writing – review & editing, Visualization,

Project administration. **Sang-Woong Lee:** Writing – review & editing, Validation, Data curation. **Thantrira Porntaveetus:** Writing – review & editing, Resources, Project administration. **Mehdi Hosseinzadeh:** Writing – review & editing, Project administration. **Amir Haider:** Writing – review & editing, Data curation. **Amir Masoud Rahmani:** Writing – review & editing, Resources. **Farhad Soleimanian Gharehchopogh:** Writing – original draft, Visualization, Methodology, Conceptualization. **Shakiba Rajabi:** Writing – original draft, Visualization, Methodology, Conceptualization.

Declaration of Generative AI and AI-assisted technologies in the writing process

During the preparation of this paper, the authors used Grammarly to improve the language and readability of the manuscript. After using this tool, the authors reviewed and edited the content as needed and take full responsibility for the content of the publication.

Declarationof Competing interest

The authors declare that they have no known competing financial interests 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] E.H. Houssein, M.A. Othman, W.M. Mohamed, M. Younan, Internet of things in smart cities: comprehensive review, open issues and challenges, pp. 1-1, IEEE Internet Things J. (2024), https://doi.org/10.1109/JIOT.2024.3449753.

[2] D.A. Zainaddin, Z.M. Hanapi, M. Othman, Z. Ahmad Zukarnain, M.D.H. Abdullah, Recent trends and future directions of congestion management strategies for routing in IoT-based wireless sensor network: a thematic review, /04/01 2024, Wirel. Netw. 30 (3) (2024) 1939–1983, https://doi.org/10.1007/s11276-023-03598-w.

[3] A.I. Awad, M.M. Fouda, M.M. Khashaba, E.R. Mohamed, K.M. Hosny, Utilization of mobile edge computing on the Internet of medical things: a survey, /06/01/ 2023, ICT Express 9 (3) (2023) 473–485, https://doi.org/10.1016/j.icte.2022.05.006.

[4] K. Moghaddasi, S. Rajabi, F.S. Gharehchopogh, A. Ghaffari, An advanced deep reinforcement learning algorithm for three-layer D2D-edge-cloud computing architecture for efficient task offloading in the Internet of things, /09/01/ 2024, Sustain. Comput. Inform. Syst. 43 (2024) 100992, https://doi.org/10.1016/j.suscom.2024.100992.

[5] K. Moghaddasi, S. Rajabi, F. Soleimanian Gharehchopogh, M. Hosseinzadeh, An energy-efficient data offloading strategy for 5G-Enabled vehicular edge computing networks using double deep Q-Network, /12/01 2023, Wirel. Pers. Commun. 133 (3) (2023) 2019–2064, https://doi.org/10.1007/s11277-024-10862-5.

[6] H. Min, A.M. Rahmani, P. Ghaderkourehpaz, K. Moghaddasi, M. Hosseinzadeh, A joint optimization of resource allocation management and multi-task offloading in high-mobility vehicular multi-access edge computing networks, /01/01/ 2025, Ad Hoc Netw. 166 (2025) 103656, https://doi.org/10.1016/j.adhoc.2024.103656.

[7] A.M. Rahmani, et al., Optimizing task offloading with metaheuristic algorithms across cloud, fog, and edge computing networks: a comprehensive survey and state-of-the-art schemes, /01/01/ 2025, Sustain. Comput. Inform. Syst. 45 (2025) 101080, https://doi.org/10.1016/j.suscom.2024.101080.

[8] M. Reiss-Mirzaei, M. Ghobaei-Arani, L. Esmaeili, A review on the edge caching mechanisms in the mobile edge computing: a social-aware perspective, /07/01/ 2023, Internet Things 22 (2023) 100690, https://doi.org/10.1016/j.iot.2023.100690.

[9] M.Y. Akhlaqi, Z.B. Mohd Hanapi, Task offloading paradigm in mobile edge computing-current issues, adopted approaches, and future directions, /03/01/ 2023, J. Netw. Comput. Appl. 212 (2023) 103568, https://doi.org/10.1016/j.jnca.2022.103568.

[10] 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.

[11] A. Nain, S. Sheikh, M. Shahid, R. Malik, Resource optimization in edge and SDN-based edge computing: a comprehensive study, /08/01 2024, Clust. Comput. 27 (5) (2024) 5517–5545, https://doi.org/10.1007/s10586-023-04256-8.

[12] V. Tyagi, S. Singh, Network resource management mechanisms in SDN enabled WSNs: a comprehensive review, /08/01/ 2023, Comput. Sci. Rev. 49 (2023) 100569, https://doi.org/10.1016/j.cosrev.2023.100569.

[13] P.P. Ray, N. Kumar, SDN/NFV architectures for edge-cloud oriented IoT: a systematic review, /03/01/ 2021, Comput. Commun. 169 (2021) 129–153, https://doi.org/10.1016/j.comcom.2021.01.018.

[14] W. Rafique, L. Qi, I. Yaqoob, M. Imran, R.U. Rasool, W. Dou, Complementing IoT services through software defined networking and edge computing: a comprehensive survey, IEEE Commun. Surv. Tutor. 22 (3) (2020) 1761–1804, https://doi.org/10.1109/COMST.2020.2997475.

[15] M.R. Belgaum, S. Musa, M.M. Alam, M.M. Su'ud, A systematic review of load balancing techniques in Software-Defined networking, IEEE Access 8 (2020) 98612–98636, https://doi.org/10.1109/ACCESS.2020.2995849.

[16] K. Kaur, V. Mangat, K. Kumar, A comprehensive survey of service function chain provisioning approaches in SDN and NFV architecture, /11/01/ 2020, Comput. Sci. Rev. 38 (2020) 100298, https://doi.org/10.1016/j.cosrev.2020.100298.

[17] F. Schardong, I. Nunes, A. Schaeffer-Filho, NFV resource allocation: a systematic review and taxonomy of VNF forwarding graph embedding, /02/11/ 2021, Comput. Netw. 185 (2021) 107726, https://doi.org/10.1016/j.comnet.2020.107726.

[18] Q. Luo, S. Hu, C. Li, G. Li, W. Shi, Resource scheduling in edge computing: a survey, IEEE Commun. Surv. Tutor. 23 (4) (2021) 2131–2165, https://doi.org/10.1109/COMST.2021.3106401.

[19] R.S. Alonso, I. Sittón-Candanedo, R. Casado-Vara, J. Prieto, J.M. Corchado, Deep reinforcement learning for the management of software-defined networks and network function virtualization in an edge-IoT architecture, Sustainability 12 (14) (2020) 5706, https://doi.org/10.3390/su12145706.

[20] T. Rakkiannan, et al., An automated network slicing at edge with software defined networking and network function virtualization: a federated learning approach, Wirel. Pers. Commun. 131 (1) (2023) 639–658, https://doi.org/10.1007/s11277-023-10450-z.

[21] Z. Lv, W. Xiu, Interaction of edge-cloud computing based on SDN and NFV for next generation IoT, IEEE Internet Things J. 7 (7) (2019) 5706–5712, https://doi.org/10.1109/JIOT.2019.2942719.

[22] C. Yang, F. Liao, S. Lan, L. Wang, W. Shen, G.Q. Huang, Flexible resource scheduling for software-defined cloud manufacturing with edge computing, Engineering 22 (2023) 60–70, https://doi.org/10.1016/j.eng.2021.08.022.

[23] T. He, A.N. Toosi, R. Buyya, Efficient large-scale multiple migration planning and scheduling in SDN-enabled edge computing, IEEE Trans. Mob. Comput. (2023), https://doi.org/10.1109/TMC.2023.3326610.

[24] Y. Li, B. Wu, Software-Defined heterogeneous edge computing network resource scheduling based on reinforcement learning, Appl. Sci. 13 (1) (2022) 426, https://doi.org/10.3390/app13010426.

[25] S.S. Jazaeri, P. Asghari, S. Jabbehdari, H.H.S. Javadi, Composition of caching and classification in edge computing based on quality optimization for SDN-based IoT healthcare solutions, J. Supercomput. 79 (15) (2023) 17619–17669, https://doi.org/10.1007/s11227-023-05332-x.

[26] C.-L. Hu, C.-Y. Hsu, W.-M. Sung, FitPath: QoS-based path selection with fittingness measure in integrated edge computing and software-defined networks, IEEE Access 10 (2022) 45576–45593, https://doi.org/10.1109/ACCESS.2022.3170056.

[27] J. Du, C. Jiang, A. Benslimane, S. Guo, Y. Ren, SDN-based resource allocation in edge and cloud computing systems: an evolutionary stackelberg differential game approach, IEEE/ACM Trans. Netw. 30 (4) (2022) 1613–1628, https://doi.org/10.1109/TNET.2022.3152150.

[28] K. Sadatdiynov, L. Cui, L. Zhang, J.Z. Huang, S. Salloum, M.S. Mahmud, A review of optimization methods for computation offloading in edge computing networks, /04/01/ 2023, Digit. Commun. Netw. 9 (2) (2023) 450–461, https://doi.org/10.1016/j.dcan.2022.03.003.

[29] J. Wu, F. Dong, H. Leung, Z. Zhu, J. Zhou, S. Drew, Topology-aware federated learning in edge computing: a comprehensive survey, p. Article 262, ACM Comput. Surv. 56 (10) (2024), https://doi.org/10.1145/3659205.

[30] T. Qiu, J. Chi, X. Zhou, Z. Ning, M. Atiquzzaman, D.O. Wu, Edge computing in industrial Internet of things: architecture, advances and challenges, IEEE Commun. Surv. Tutor. 22 (4) (2020) 2462–2488, https://doi.org/10.1109/COMST.2020.3009103.

[31] Y. Mansouri, M.A. Babar, A review of edge computing: features and resource virtualization, /04/01/ 2021, J. Parallel Distrib. Comput. 150 (2021) 155–183, https://doi.org/10.1016/j.jpdc.2020.12.015.