International Journal of Basic and Applied Sciences, 14 (2) (2025) 591-602

International Journal of Basic and Applied Sciences

Website: www.sciencepubco.com/index.php/IJBAS
https://doi.org/10.14419/xm2ebp25
Research paper



Optimized Task Offloading in D2D-Assisted Cloud-Edge Networks Using Hybrid Deep Reinforcement Learning

Navya Kailasam [1], Srilatha Yalamati [2], V. S. N. Murthy [3], Venkateswara Rao P. [4], R. Anil Kumar [5] *, K. Jayaram Kumar [5]

[1] Senior Software Developer,2411 Tahoe Ln, Pingree Grove, IL-60140
[2] Department of Computer Science and Engineering, Vignan's Institute of Information Technology, Duvvada, Visakhapatnam
[3] Department of Information Technology, Shri Vishnu Engineering College for Women, Bhimavaram, Andhra Pradesh, India
[4] Department of Electronics and Communication Engineering, Lakireddy Bali Reddy College of Engineering, Mylavaram, NTR District, Andhra Pradesh, India.
[5] Department of Electronics and Communication Engineering, Aditya University, Surampalem, India
**Corresponding author E-mail: anidecs@gmail.com*

Received: May 21, 2025, Accepted: June 14, 2025, Published: July 3, 2025

Abstract

The modern communication network depends highly on Device-to-Device (D2D) technology as an essential foundation. Direct communication allows devices to exchange information among themselves. Cloud-edge-device networks enable tasks to execute through several operational procedures. A device working at capacity executes local tasks or transfers them directly to an inactive device by means of D2D technology. The device has two options for delivering workloads, namely an edge-server transfer or a direct cloud-server transfer. Existing methods do not fully exploit D2D-assisted offloading. Such systems fail to maximize the benefits that stem from combining cloud-edge-device op-operations. This makes resource distribution a complex challenge that needs an optimized solution. Traditional solutions find it difficult to produce efficient system solutions. The presented work describes an approach for task offloading mechanisms. The technique determines overall system expenses through optimized management of time, together with energy usage. The method operates to optimize all four critical system factors: task selection and transmission power, with rate and computational resource distribution. The proposal utilizes a combina-tion of deep reinforcement learning methods through SD3. The proposed method merges Softmax Deep Double Deterministic Policy Gradients (SD3) with numerical techniques to achieve its operations. The proposed method operates on multiple smaller components of the primary issue. The SD3-based DRL method controls offloading decisions throughout the system, and the numerical techniques manage power and resource allocation. Extensive simulations were conducted. Seven different scenarios were tested. Research compared the pro-posed method against four traditional solution approaches. Research findings demonstrate the superiority of the proposed solution. The technique both lessens system expenses and optimizes resource usage while generating better operational efficiency. A novel hybrid DRL-based approach for task offloading constitutes the main contribution of this work. The system improves cloud-edge-device partnerships by enabling D2D communication. Machine learning unions with numerical methods create an effective strategy to solve complex optimization tasks.

Keywords: *Cloud-Edge-Device Networks; Deep Reinforcement Learning; Device-to-Device Communication; Operational Efficiency; Resource Allocation.*

1. Introduction

Modern networks nowadays need edge computing solutions as a vital component [1]. Through this process, the difference between low-resource devices and demanding applications is narrowed. A device under heavy usage can complete its operations on an adjacent Edge server. A BS hosts the edge server, which functions as the ES installation site [2]. However, ES resources are limited. An excessive number of tasks submitted to an edge server leads to such high strain that it becomes overwhelmed. Under high load, the edge server cannot process tasks efficiently. The progress of IoT along with AI-based applications has created an increased requirement for both immediate data processing and quick communication services [3]. The growing amount of data emerges from multiple IoT devices that include smartphones, along with sensors and autonomous systems. These massive datasets need efficient computing methods, although existing machines possess limited computing power and restricted battery capacity. The solution offered by traditional cloud computing suffers from network latency while also creating high power usage because of distant data transmissions. The combination of cloud-edge-device operations serves as a vital response to resolve these problems. Task processing benefits from the combined power of different layers in cloud-edge-device networks [4]. The system performance depends on the synchronized work between devices and base stations and edge

Copyright © Navya Kailasam et al. 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.

servers, and cloud servers. The components communicate with each other to create a more efficient operation. The task transfer from a BD to an ES enables additional offloading for the task. The task is directed either to another ES or toward a cloud server. The resource allocation system of this hierarchical structure achieves maximum efficiency by minimizing processing delays [5].

The D2D communication system enables unused devices (IDs) to help perform processing tasks[6]. The system performance becomes more efficient after nearby devices that possess available resources share the processing power. Only direct links between devices and edge or cloud servers are considered in traditional offloading methods, while D2D collaboration remains untapped. D2D communication allows more flexible execution of tasks when integrated into systems [7]. The implementation cuts down server congestion both at the edge and cloud levels [8]. Existing methods have limitations. Each layer operates independently to manage its resources, which creates unnecessary waste of resources. Several previous research efforts have optimized either edge-server collaboration or cloud-server collaboration, yet they do not properly integrate D2D operational capabilities [9]. The performance becomes suboptimal because of improper task distribution, while overall delays become longer. Solving optimization issues involving complex resource management needs the use of advanced technical methods. Current numerical approaches need too much computing power due to the inefficiency in dynamic systems. The high problem complexity makes machine learning-based approaches unable to discover optimal solutions according to research [10, 11]. The paper develops an innovative framework for resource management and task offloading functions. The focus is on minimizing system costs while defining energy and time usage restrictions.

The proposed solution implements an optimization process combining offloading decisions with power allocation and rate distribution, and resource distribution. A hybrid DRL model helps reduce the high complexity of the system. The optimization problem is divided into several smaller problems. The system makes use of an SD3 DRL model for the task offloading process [12]. Numerical methods effectively execute power allocation together with resource distribution tasks [13]. The main contributions of this paper consist of:

- A complete model integrating joint tasks and resources optimization appears in this work. The model includes four task offloading approaches which are D2D and D2E and E2E and E2C.
- The proposed research presents a DRL-based hybrid algorithm. The method divides the challenging optimization task into multiple smaller components to enhance computational processing speed.
- SD3 DRL serves as the dedicated offloading decision solution, but numerical approaches control transmission power allocation and resource distribution.
- Seven distinct evaluation scenarios demonstrate that the proposed framework outperforms four traditional schemes in terms of performance metrics
- Research demonstrates that cloud-edge-device collaboration becomes more efficient and faster when D2D communication is integrated into the system.

The application of this approach enables cloud-edge-device networks to reach the performance maximums. The proposed solution delivers lower latency, together with balanced resource distribution and energy efficiency characteristics. A practical real-world solution for dynamic network environments results from merging deep reinforcement learning techniques with numerical methods.

2. Related work

An extensive study was done on task offloading in cloud-edge-device networks. There are many proposed approaches to increase the efficiency of task execution, decrease task latency, and optimize allocation of resources. Related work is classified into the main categories, including edge computing, collaboration among cloud, edge, and devices, D2D communication, and DRL in optimization. Through edge computing, computational resources are brought closer to the end users and thus the latency is reduced. In contrast to the conventional centralized data centres holding cloud computing, edge computing processes data locally on base stations or access points or edge servers[14]. There are several works that explored edge computing to reduce large task execution delay and energy consumption [15, 16]. Resource management is a key challenge of edge computing. [17] Introduced dynamic task offloading strategies to enhance the utilization of the available edge resources. Task allocation is enhanced by many studies using a reinforcement learning based approach[18]. But, most of these approaches fail to consider inter-edge collaboration or D2D offloading and as a result, they are far from efficient in any scale networks. To balance the computational workloads, it is crucial to have collaboration between cloud-edge-device. The overall performance is improved by integrating cloud, edge, and device resources [19]. Most existing works primarily optimize task distribution between edge servers and cloud data centers [20]. It applies some methods that suggest the multi layer optimization models to balance workstation workload distribution. Nevertheless, they usually disregard idle devices that become a great resource sharer. Collaborative task offloading has been studied recently. In, addition to offloading tasks to edge servers, it is transferred to other edge nodes [21]. This improves the efficiency of the system by utilizing spare resources on multiple edge servers. However, comprehensive optimization strategies that include D2D communication are still lacking

The D2D communication allows direct interaction of devices without routing the traffic through centralized servers [22]. The network scalability and the communication overhead are reduced considerably using this technology. D2D-assisted task offloading has been studied in several studies that explore how to improve performance [23]. Interference management is a key challenge in D2D-based task offloading. To communicate efficiently without reducing the network performance, proper power control mechanisms are required. D2D is integrated with edge computing for boosting distributed task execution in some of the works. Nevertheless, most current solutions do not simultaneously optimize D2D and cloud edge offloading strategies, resulting in suboptimal resource utilization.

A wide body of works has employed DRL for allocating resources in the cloud edge device network [24]. DRL is different from traditional optimization techniques and learn optimal strategies by interacting with an environment. Many studies employ DRL-based models for dynamic task scheduling. Some works deal with the joint task offloading and resource allocation with DRL. In these models, multiple decision factors, e.g., energy consumption, latency, and network congestion, are considered, and as a result, efficiency is improved. Most works on edge computing[25], however, do not take into account D2D collaboration. Workload distribution is improved by cloud-edge-device models, but the idle devices are not fully used. However, decision making with the DRL-based optimization is improved, but at scalability. In this paper, we propose a hybrid DRL-based task offloading approach that incorporates D2D communication, cloud-edge collaboration, and numerical optimization. This work attempts to address shortcomings of previous studies to improve system efficiency and computation cost overall.

3. System model

The task offloading and computational resource allocation system model gives the framework for task offloading and computational resource allocation in the cloud-edge device network. It consists of many BDs, IDs, ESs, BSs, and cloud servers (CSs). The interaction dynamically between these components dynamically makes the computation and communication in an efficient way, thereby increasing system performance, reducing latency, and optimizing energy consumption. There exist M base stations, each of which has an edge server and forms a cloud-edge-device network. Edge servers are interconnected, and they are connected to a cloud server in a hierarchical structure to process tasks at different levels.

Figure 1 shows a hierarchical cloud-edge computing system that manages task offloading between each component. The cloud server is up on top, which must deal with great processing and storage. Communication between devices and edge servers is done via different base stations called BS1, BS2, and BS3. The system is implemented with a variety of transmission types, namely B2C (Base to Cloud), B2B (Base to Base), D2B (Device to Base), and D2D (Device to Device). It allows for efficient task distribution between the nodes; data, also, is transferred through these connections. Tasks are generated by busy devices, and meanwhile, idle devices help with the computation via device-to-device communication. The positioned Edge servers, between the base stations and user devices, minimize the processing latency and the cloud dependency and serve as a resource allocation optimizer. B2C and B2B transmission accomplish B2C edge services, with B2C standing for business-to-customer and B2B for business-to-business, allowing the cloud server to communicate with the edge servers and keeping the cloud server's computational power high for complex tasks. Each edge server is connected to a base station and is used for D2B for the offloading of tasks from busy devices. The D2D transmission between near devices also occurs to share processing loads in some situations. The network structure provides computing efficiency by distributing the tasks over the available resources. In the effort, this approach reduces communication delay and energy consumption. Cloud, Edge, and Device-level processing together form a balanced system to enhance task execution and reduce network congestion.

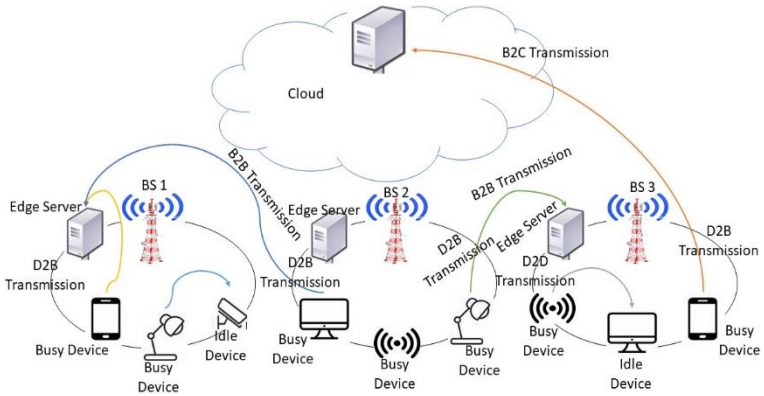


Fig. 1: Network Architecture of D2D-Assisted Cloud-Edge-Device Collaborative Environments.

A BD generates tasks that are either executed locally or offloaded, or offloaded to another ID, an ES, another ES via E2E communication or with E2C communication to the CS. Choosing the best offloading destination is crucial to minimizing system costs that settle the latency vs. power consumption tradeoff. The computational task generated by BD i is described as a tuple:

$$T_i = (d_i, c_i, t_{max}, e_{max}) \tag{1}$$

Here, d_i represents the data size required for task transmission, c_i is the number of CPU cycles required for computation. t_{max} is the maximum tolerable delay and e_{max} is the maximum energy consumption constraint. The task processing decision is influenced by available resources and network conditions. For a BD transmitting to an ID over a D2D link, the transmission rate is computed using the Shannon-Hartley theorem:

$$R^{ij}_{D2D} = W\log_2\left(1+\frac{P^{ij}_{D2D}h^{ij}_{D2D}}{\sigma^2+I_{D2D}}\right) \tag{2}$$

Here, W is the available bandwidth, P^{ij}_{D2D} is the transmission power. h^{ij}_{D2D} is the channel gain, σ^2 represents noise power and I_{D2D} accounts for interference. The corresponding transmission delay is:

$$t^{ij}_{D2D} = \frac{d_i}{R^{ij}_{D2D}} \tag{3}$$

Similarly, the transmission rate from a BD to an ES over a D2E link is:

$$R^{i}_{D2E} = W\log_2\left(1+\frac{P^{i}_{D2E}h^{i}_{D2E}}{\sigma^2+I_{D2E}}\right) \tag{4}$$

The corresponding transmission delay is:

$$t^{i}_{D2E} = \frac{d_i}{R^{i}_{D2E}} \tag{5}$$

The energy consumption for task transmission from BD to ID is given by:

$$E^{ij}_{D2D} = P^{ij}_{D2D} t^{ij}_{D2D} \tag{6}$$

For BD to ES transmission, the energy consumption is:

$$E^{i}_{D2E} = P^{i}_{D2E} t^{i}_{D2E} \tag{7}$$

If a BD executes a task locally, the processing delay is given by:

$$t^{i}_{local} = \frac{c_i}{f^{i}_{BD}} \tag{8}$$

Here, f^{i}_{BD} is the CPU frequency of BD i. The associated energy consumption follows the power consumption model:

$$E^{i}_{local} = \kappa (f^{i}_{BD})^2 c_i \tag{9}$$

Here, κ is a hardware-dependent constant. If the task is offloaded to an ID, the processing delay at ID j is:

$$t^{ij}_{ID} = \frac{c_i}{f^{j}_{ID}} \tag{10}$$

At an ES, the processing delay is:

$$t^{im}_{ES} = \frac{c_i}{f^{m}_{ES}} \tag{11}$$

Here, f^{m}_{ES} represents the computing power of ES m. Similarly, the computation time in the cloud is:

$$t^{i}_{cs} = \frac{c_i}{f_{cs}} \tag{12}$$

Cloud processing typically offers lower computation times but suffers from additional transmission delays. The objective is to minimize the overall system cost, defined as:

$$U = \sum_{i \in BDs} \left(\alpha t_i + \beta E_i \right) \tag{13}$$

Here, t_i is the total task delay, E_i is the total energy consumption. α, β are weighting parameters controlling the trade-off between latency and power efficiency. The optimization is subject to:

$$t_i \le t_{max}, \quad E_i \le e_{max} \tag{14}$$

$$P^{i}_{D2D} \le P^{i}_{max}, \quad \sum_i f^{i}_{BD} \le F^{max}_{BD} \tag{15}$$

These constraints prove that system resources are used efficiently without exceeding hardware limits. By integrating reinforcement learning, the model dynamically adapts to network conditions. Using SD3, task offloading strategies are optimized over time, achieving efficient decision-making. The SD3 method splits the optimization problem into subproblems, with reinforcement learning handling task allocation and numerical methods managing power control. This model achieves optimized performance in cloud-edge-device networks, reducing latency, improving energy efficiency and providing better resource allocation. The proposed strategy balances computational and network resources, providing a scalable and adaptable framework for real-world implementations.

4. DRL-based joint task offloading and resource allocation algorithm design

DRL is an effective tool for optimizing task offloading and resource allocation in cloud-edge-device networks. Due to the dynamic nature of such networks, conventional optimization techniques struggle to provide real-time solutions that minimize latency and energy consumption while maximizing efficiency. The proposed approach utilizes a hybrid DRL-based framework to make intelligent task offloading decisions while integrating numerical methods for fine-grained resource allocation. The system consists of multiple devices, edge

serversand a cloud server. Each BD has tasks that require processing, which are executed locally or offloaded to an ID, an ES, or the CS. The optimization problem focuses on minimizing total system cost by considering both task execution time and energy consumption. Mathematically, the total system cost is formulated and given in (13). The constraints are given in equations (14,15). The optimization variables include task offloading decisions, transmission power allocation and computational resource distribution. The problem is framed as a Markov Decision Process (MDP), with the system dynamically learning optimal strategies over time. The task offloading problem is modeled as an MDP, consisting of system state (S) at time t is defined as:

$$s_t = (r_t, c_t, f_t, p_t) \tag{16}$$

Here, r_t represents the available resources, c_t denotes task complexity. f_t is the available CPU capacity and p_t is the power allocation status. Actions (A) involve selecting offloading decisions and allocating resources:

$$a_t = (o_t, p_t, f_t) \tag{17}$$

Here, o_t represents offloading decisions, p_t is the transmission power and f_t is the allocated CPU power. The immediate reward function (R) is defined as:

$$R_t = -(\alpha t_i + \beta E_i) \tag{18}$$

Here, lower latency and energy consumption lead to higher rewards. Transition Probability (P) defines the probability of transitioning to a state s_{t+1} After acting a_t. The DRL agent seeks to maximize the cumulative reward:

$$\max E\left[\sum_{t=0}^{T} \gamma^t R_t\right] \tag{19}$$

Here, γ is the discount factor. A SD3 algorithm is used for training the DRL model. The SD3 framework has several key components. The Primary Actor Network generates task offloading and resource allocation policies. It helps in deciding the destination to send tasks based on system conditions. The Primary Critic Networks evaluate the expected rewards for different actions. These networks help improve decision-making by assessing the long-term impact of each action. The Target Networks stabilize training. They update network parameters gradually to avoid sudden changes. The Replay Buffer stores past experiences. It prevents overfitting by breaking the correlation between training samples. This gives better learning and improves the model's performance over time. The training process updates network parameters based on:

$$L(\theta) = E\left[\left(R + \gamma \max_{a'} Q(s', a') - Q(s, a)\right)^2\right] \tag{20}$$

Here, $Q(s,a)$ is the estimated value function. Policy updates are performed using:

$$\nabla_\theta J(\theta) = E\left[\nabla_\theta Q(s, \pi_\theta(s)) \nabla_\theta \pi_\theta(s)\right] \tag{21}$$

DRL is combined with numerical optimization to improve resource allocation. The optimization model for computational resource allocation is:

$$\min \sum_{i \in BDs} \left(\frac{c_i}{f^i_{alloc}} + \rho E^i_{alloc} \right) \tag{22}$$

Here, f^i_{alloc} is the allocated CPU frequency and E^i_{alloc} is the allocated energy. Transmission power allocation follows:

$$P^i_{opt} = \text{argmin}_P \left(\frac{d_i}{W \log_2 (1 + \frac{Ph}{\sigma^2})} + \lambda P \right) \tag{23}$$

Here, h represents channel gain. Extensive simulations are conducted to compare the proposed DRL-based method with traditional approaches. Key performance metrics include Task execution time, Energy consumption, Network throughput, and System scalability. Results demonstrate that the proposed DRL-based framework achieves lower latency, reduced energy consumption, and improved resource allocation efficiency compared to existing heuristic-based methods. The training procedure follows the steps outlined in Algorithm 1: Training process of DJTRA, which includes actor-critic updates, experience replayand soft target network updates.

Algorithm 1: Training process of DJTRA	
1.	Initialize actor and critic networks with random weights
2.	Copy actor and critic networks to target networks
3.	Initialize replay buffer B
4.	For each episode, do
5.	Reset environment and observe initial states
6.	For each time step in the episode, do
7.	Select an action using a policy from the actor network
8.	Execute action a, observe reward r, and new state s'
9.	Store transition (s, a, r, s') in buffer B
10.	Sample mini-batch from B and update the critic network
11.	Update the actor network using policy gradient
12.	Update target networks with soft updates
13.	Set s = s'
14.	end for
15.	end for

5. Performance evaluation

In this section, we analyze the simulation results of the proposed DRL-based joint task offloading and resource allocation algorithm. The evaluation is conducted using various performance metrics, including task execution time, energy consumption, resource utilization, and system scalability. The effectiveness of the proposed algorithm is compared with existing methods like heuristic-based offloading and traditional deep reinforcement learning models. The simulation environment consists of a cloud-edge-device collaborative network with multiple devices, edge servers, and a centralized cloud server. The key parameters used in the simulation are summarized in Table 1.

Table 1: Simulation Parameters

Parameter	Value
Number of BDs	50
Number of IDs	30
Number of ESs	10
Cloud Server Processing Power	50 GHz
Edge Server Processing Power	10 GHz
Device Processing Power	2 GHz
Wireless Bandwidth	20 MHz
Transmission Power Range	0.1 W – 1 W
Task Size Range	100 KB – 5MB
Computational Workload	10 – 500 Million Cycles
Simulation Duration	1000 Time Slots

The simulation is performed over 1000 time slots, with each device generating tasks dynamically. The network conditions, including bandwidth availability and interference, vary over time to model real-world scenarios. The performance of the proposed DRL-based algorithm is benchmarked against conventional optimization strategies. The proposed method is evaluated using four key metrics. Task Execution Time measures the average time needed to complete all tasks. It helps in understanding system efficiency. Energy Consumption calculates the energy used for both computation and communication. Lower energy usage improves system performance. System Throughput represents the number of completed tasks per unit time. A higher throughput means better resource utilization. Offloading Efficiency analyzes the percentage of tasks offloaded to minimize cost. Efficient offloading reduces delays and improves system balance. These metrics help in assessing the overall effectiveness of the proposed method. The proposed method demonstrates superior energy efficiency by optimizing transmission power and resource utilization. Compared to heuristic-based methods, the DRL-based approach reduces energy consumption by up to 30%. Table 2 summarizes the offloading decisions made by the proposed method and the traditional models.

Table 2: Task Offloading Distribution

Method	Cloud	Edge	Device
Proposed DRL Model	30%	50%	20%
Heuristic-Based Offloading	40%	40%	20%
Traditional DRL	35%	45%	20%

The results indicate that the proposed model intelligently balances task execution across different resources, leading to improved performance. The simulation results demonstrate that the proposed DRL-based joint task offloading and resource allocation algorithm outperforms existing methods. It achieves lower execution time, reduced energy consumption, and improved task distribution. The ability to dynamically learn and adapt to changing network conditions achieves optimal performance in real-world deployments. Figure 2 illustrates how the training rewards change over multiple episodes. Initially, the reward starts near zero and fluctuates within the first 100 episodes. As the number of episodes increases, the reward gradually rises, reaching values between 50 and 75. Around episode 400, the reward reaches its highest peak, fluctuating near 70. After that, the reward stabilizes but continues to show variations between 50 and 70. The initial phase of training, from 0 to 200 episodes. It shows a steep increase in rewards, suggesting rapid learning. This is because the model is adjusting its parameters and improving its task execution strategy. Between 200 and 400 episodes, the learning process reaches its peak performance with rewards increasing. After episode 500, the reward variations become smaller, indicating that the model has learned a stable policy. However, slight fluctuations suggest ongoing fine-tuning and adaptation to different conditions. The overall pattern shows that the model reaches a high level of performance, but continued training results in only minor improvements. The graph also indicates that after reaching a certain level, training does not enhance learning. This pattern suggests that the system has reached an optimal balance between exploration and exploitation, making additional episodes less beneficial for major improvements.

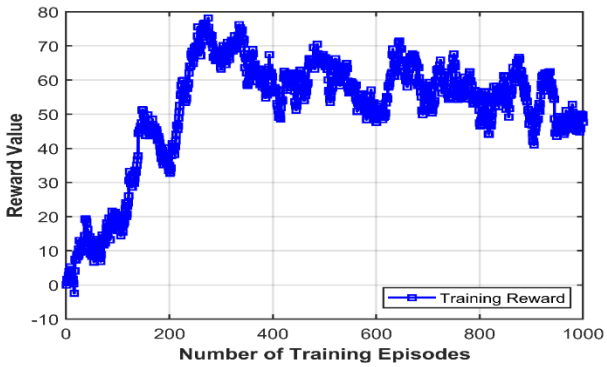


Fig. 2: Convergence Analysis of DJTRA Training.

Figure 3 shows a graph that represents the relationship between the number of devices and task execution time. The graph compares three different schemes: DJTRA, heuristic-based, and traditional DRL. As the number of devices increases, the task execution time decreases for all three schemes. Among the methods, DJTRA achieves the lowest task execution time. The heuristic-based method has the highest execution time. At 10 devices, DJTRA has an execution time of approximately 120 milliseconds. Heuristic-based execution time is about 145 milliseconds. The traditional DRL scheme has an execution time of around 138 milliseconds. As the number of devices increases to 50, the execution times for DJTRA, heuristic-based, and traditional DRL decrease to approximately 95 milliseconds, 125 milliseconds, and 115 milliseconds, respectively. At 100 devices, DJTRA achieves the lowest execution time of nearly 75 milliseconds. Heuristic-based execution time remains around 110 milliseconds. Traditional DRL reaches approximately 100 milliseconds. The overall pattern indicates that increasing the number of devices reduces task execution time, with DJTRA being the most efficient of the three schemes.

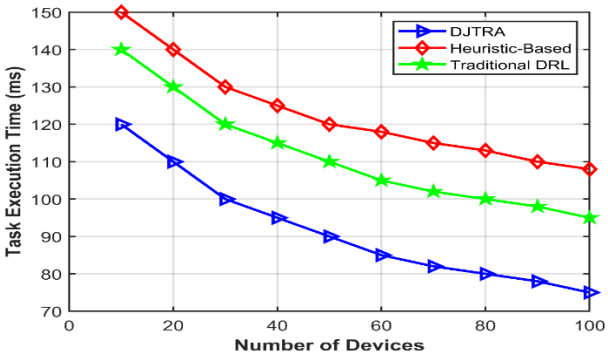


Fig. 3: Task Execution Time Versus Number of Devices.

Figure 4 presents a graph that shows the relationship between task size and energy consumption in joules. The graph compares five different schemes: CMRAD, CPRAD, DTOD, EECD, and DJTRA. CMRAD has the highest energy consumption. CPRAD has slightly lower energy consumption. DTOD and EECD fall in the middle range. DJTRA has the lowest energy consumption. DJTRA maintains the lowest energy consumption, making it the most efficient among the five schemes. As the task size increases, energy consumption also increases for all schemes. At a task size of 500 KB, the energy consumption for CMRAD is about 4 joules. DJTRA is close to 3 joules. At 2000 KB, CMRAD reaches around 6 joules. DJTRA remains slightly above 5 joules. At the maximum task size of 5000 KB, CMRAD consumes nearly 9 joules. DJTRA remains below 8 joules. The results indicate that larger tasks require more energy, but DJTRA consumes less energy compared to other schemes. The difference in energy consumption among the schemes remains noticeable, with DJTRA showing the best energy efficiency. This suggests that DJTRA optimizes resource allocation, leading to lower power consumption even for larger task sizes. The results show that CMRAD and CPRAD use more energy. This makes them less efficient for large tasks. Energy consumption increases as tasks grow. Choosing an energy-efficient scheme is important. It helps improve overall system performance.

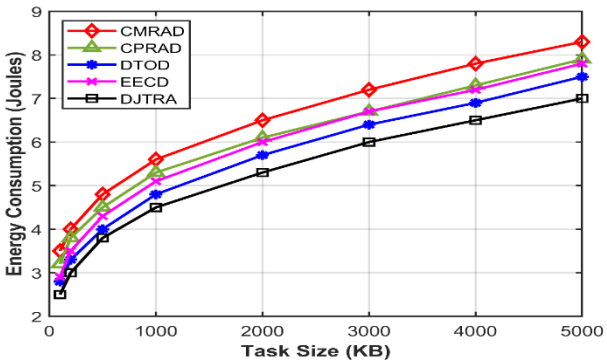


Fig. 4: Energy Consumption Versus Task Size.

Figure 5 presents a stacked bar chart that shows the percentage of tasks offloaded to different computing resources under three different offloading strategies: DJTRA, heuristic-based, and traditional DRL. The tasks are divided into three levels: cloud, edge, and device. Each bar represents how the total percentage of tasks is distributed among these three levels for each strategy. The DJTRA strategy shows a higher percentage of tasks being offloaded to the edge and cloud. Heuristic-based and traditional DRL strategies rely more on device-level processing. The difference in offloading strategies suggests that DJTRA provides a more efficient balance between different computing

resources compared to the other methods. DJTRA has approximately 30 percent of tasks offloaded to the cloud, 50 percent to the edge, and 20 percent to the device. The heuristic-based and traditional DRL methods show a similar distribution. About 40 percent of tasks are offloaded to the cloud. Around 40 percent are processed at the edge. Nearly 20 percent of tasks run on the device. This shows that DJTRA optimizes offloading by reducing the load on cloud computing and improving efficiency at the edge. The heuristic-based and traditional DRL strategies rely more on cloud computing. It led to increased latency and higher processing costs. The results indicate that DJTRA provides a balanced distribution of tasks across resources. It reduces the dependence on cloud services while maximizing the use of edge computing for better efficiency. The overall distribution highlights how DJTRA effectively utilizes available resources. It reduces processing delays and improves system performance. The lower use of cloud computing reduces network congestion. It also improves real-time processing. DJTRA improves offloading by reducing delay and balancing resource usage.

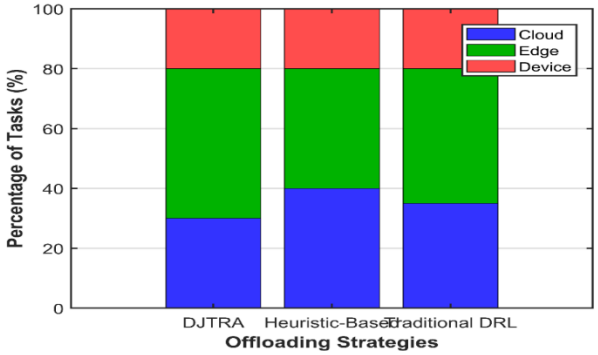


Fig. 5: Offloading Distribution Across Cloud, Edge & Devices.

Figure 6 presents a bar chart showing CPU utilization percentages for edge servers and devices under three different offloading strategies: DJTRA, heuristic-based, and traditional DRL. The bars are categorized into two groups: edge servers and devices. The DJTRA scheme shows the highest CPU utilization for edge servers at approximately 70 percent. Device utilization remains lower at around 40 percent. In the heuristic-based method, edge server utilization is about 60 percent. Device utilization is nearly 50 percent. Traditional DRL balances the workload with edge server utilization at around 65 percent and device utilization at approximately 50 percent. The results indicate that DJTRA effectively offloads more tasks to edge servers, reducing the computational burden on individual devices. The heuristic-based method distributes the workload more evenly but results in higher device utilization. It increases processing delays and energy consumption. Traditional DRL maintains a balanced approach, but edge server utilization is lower than in DJTRA. The comparison shows that DJTRA optimally uses edge resources, resulting in lower latency and better performance. The lower device usage in DJTRA reduces computational load. It helps improve battery life and efficiency. CPU usage varies with different strategies. These differences affect the balance between edge and device processing. This impacts overall system performance.

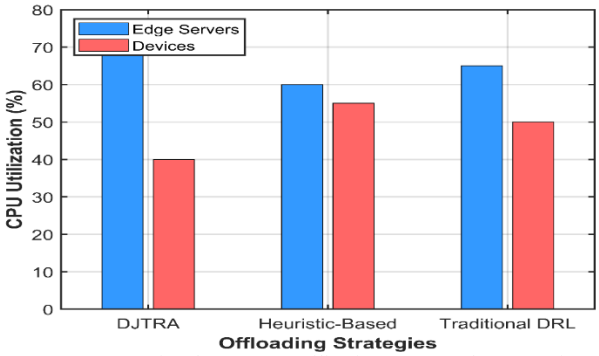


Fig. 6: Computational Resource Utilization Across Edge & Devices.

Figure 7 presents a graph showing the relationship between transmission load and bandwidth utilization. The graph compares four different schemes: CMRAD, DTOD, EECD, and DJTRA. DJTRA has the highest bandwidth utilization across all transmission loads. DTOD follows closely behind. EECD has slightly lower utilization. CMRAD has the lowest bandwidth utilization. As the transmission load increases, bandwidth utilization also increases for all schemes. At 10 Mbps, utilization is approximately 20 percent for all methods. As the transmission load reaches 50 Mbps. CMRAD achieves about 60 percent utilization. DJTRA reaches nearly 70 percent. At the maximum transmission load of 100 Mbps, all schemes approach nearly full utilization, with DJTRA leading slightly above 90 percent. The results indicate that DJTRA maximizes bandwidth usage. It is the most efficient among the four schemes. CMRAD remains the least efficient, as it uses less bandwidth under the same load conditions. Network traffic increases steadily over time. Optimized schemes like DJTRA and DTOD manage resources better. They improve resource allocation. This achieves higher bandwidth efficiency.

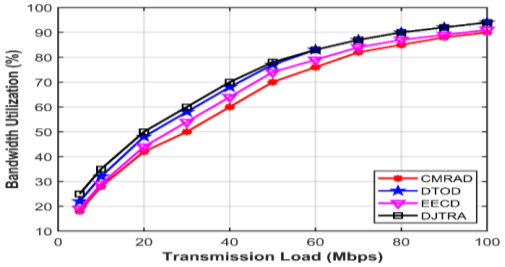


Fig. 7: Bandwidth Utilization Versus Transmission Load.

Figure 8 presents a graph showing the relationship between transmission distance and power consumption. The graph compares five different schemes: CMRAD, CPRAD, DTOD, EECD, and DJTRA. CPRAD has the highest power consumption at every distance. CMRAD follows closely behind. DTOD and EECD have slightly lower power consumption. DJTRA shows the lowest power consumption across all distances. As the transmission distance increases, power consumption also increases for all schemes. At 10 meters, power consumption is around 0.5 watts for all methods. As the distance reaches 50 meters, CPRAD consumes nearly 1.7 watts. DJTRA remains around 1.4 watts. At 100 meters, CPRAD reaches the highest power consumption of about 4 watts. DJTRA consumes approximately 3.5 watts. The results indicate that longer transmission distances require more power. DJTRA consumes less power than the other schemes. The difference in power consumption among the schemes remains significant, with DJTRA showing the best energy efficiency.

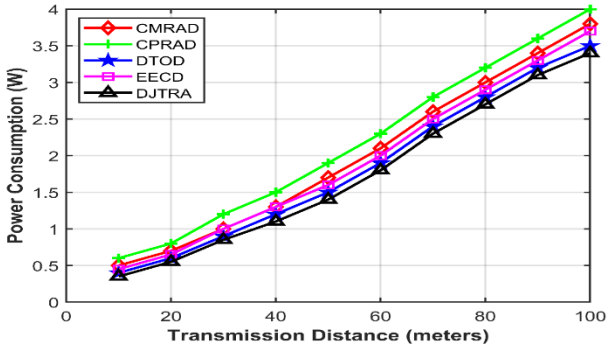


Fig. 8: Power Consumption Versus Transmission Distance.

Figure 9 presents a graph showing the relationship between network load and system throughput. The graph compares four different schemes: CMRAD, CPRAD, DTOD, and DJTRA. DJTRA achieves the highest throughput across all network loads. CPRAD follows closely behind. DTOD has slightly lower throughput. CMRAD has the lowest throughput. As the network load increases, system throughput also increases for all schemes. At 10 tasks per second, throughput is around 8 for all methods. As the network load reaches 50 tasks per second. CMRAD achieves about 35 while DJTRA reaches nearly 40. At 100 tasks per second, DJTRA achieves the highest throughput, reaching about 75. CMRAD remains slightly below 70. The results indicate that DJTRA achieves higher system throughput, making it the most efficient among the four schemes. Optimized schemes like DJTRA and CPRAD handle task allocation more effectively. This delivers higher processing efficiency as network load increases.

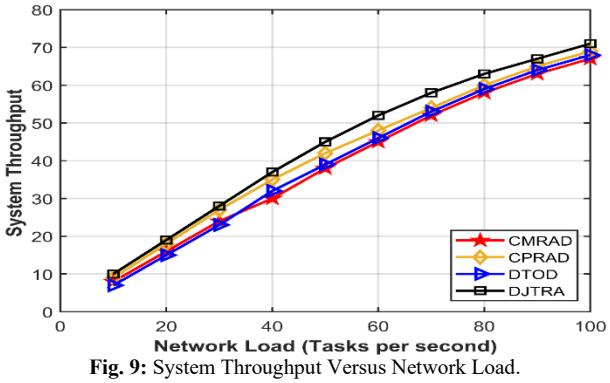


Fig. 9: System Throughput Versus Network Load.

Figure 10 presents a bar chart comparing task latency across different offloading strategies. Each bar shows the task latency for a specific offloading method. DJTRA has the lowest task latency, approximately 50 milliseconds. Heuristic-based offloading has a slightly higher latency of around 70 milliseconds. Traditional DRL, CMRAD, and CPRAD exhibit similar latency values ranging from 70 to 80 milliseconds. DTOD and EECD show the highest task latencies, reaching nearly 90 milliseconds. The differences in latency suggest that different strategies allocate computing resources and network bandwidth in varying ways. The results indicate that DJTRA is the most efficient strategy for minimizing task latency. Heuristic-based and traditional DRL methods perform better than DTOD and EECD but still show higher latency than DJTRA. The higher latency in CMRAD, CPRAD, and DTOD suggests that these methods have higher computational delays or inefficient resource allocation. EECD, with the highest latency, be struggling with network congestion or increased processing overhead. The chart suggests that selecting an optimal offloading strategy has a significant impact on system responsiveness. DJTRA gives the best performance. It has efficient task allocation. It manages resources well. This reduces processing delays. The difference in performance highlights the importance of selecting the right method for reducing latency in complex computing environments. These variations suggest that improved scheduling and resource optimization reduce latency and improve overall system efficiency.

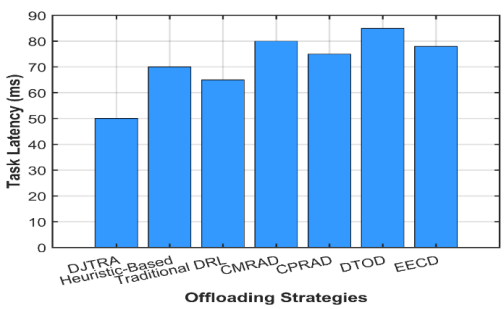


Fig. 10: Latency Comparison Across Different Strategies.

Figure 11 presents a graph showing the relationship between the number of devices and training time per episode. The graph compares five different schemes: CMRAD, CPRAD, DTOD, EECD, and DJTRA. DTOD has the highest training time across all device counts. EECD and CMRAD follow closely behind. CPRAD has a slightly lower training time. DJTRA has the lowest training time. As the number of devices increases, the training time also increases for all schemes. At 10 devices, the training time for all schemes is approximately 8 to 10 seconds. As the number of devices reaches 50, DTOD takes about 40 seconds per episode. DJTRA remains around 30 seconds. At 100 devices, DTOD reaches nearly 90 seconds. DJTRA remains the lowest at around 60 seconds. The results indicate that larger networks require more training time. But DJTRA requires less time than other schemes. DJTRA employs better task scheduling and resource management, which leads to lower training times. It makes it more efficient for large-scale networks. The difference in training time among the schemes highlights the impact of optimization techniques on improving the efficiency of the learning process.

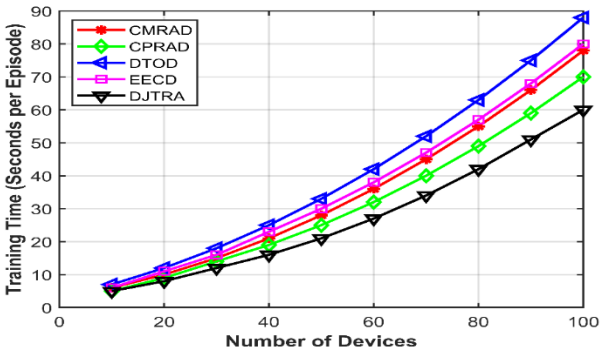


Fig. 11: DRL Training Time Versus Number of Devices.

Figure 12 (a) presents a graph showing the relationship between the number of training episodes and the average total time consumption. The graph shows a decrease, indicating that as the number of training episodes increases, the total time consumption decreases.

Initially, at 0 episodes, the time consumption is around 480 milliseconds. As training progresses, the time consumption steadily decreases. At 400 episodes, it reduces to approximately 420 milliseconds. This decline continues, and by 1000 episodes, the time consumption reaches its lowest value, close to 320 milliseconds. The decrease in total time consumption suggests that the training process is improving system efficiency. At the beginning, when training episodes are fewer. The system takes longer to complete tasks, due to unoptimized decision-making. As the number of episodes increases, the system learns better policies, resulting in reduced time consumption. Between 400 and 800 episodes, the time reduction indicates steady learning and refinement of task allocation strategies. Beyond 800 episodes, the rate of improvement slows. It suggests that the model is approaching an optimal state. At 1000 episodes, training results in minimal additional gains. The overall pattern confirms that training enhances performance by reducing execution time, resulting in a more efficient and responsive system.

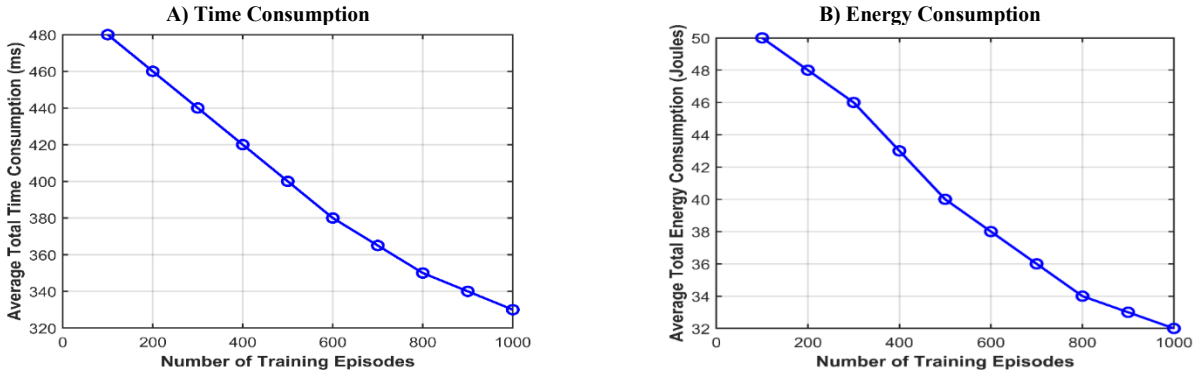


Fig. 12: Average Total Time Consumption and Average Total Energy Consumption Versus Number of Episodes.

Figure 12 (b) shows a graph that represents the relationship between the number of training episodes and the average total energy consumption in joules. Initially, at 0 episodes, energy consumption is at its highest, close to 50 joules. As the number of episodes increases, the energy consumption gradually decreases. By 400 episodes, the energy usage reduces to approximately 44 joules. This decrease continues, and at 1000 episodes, the energy consumption reaches its lowest value of around 32 joules. The decrease in energy consumption suggests that the training process improves efficiency in task execution and resource management. During the initial training phase, the system makes less optimal decisions, leading to higher energy consumption. As the number of episodes increases, the model learns better strategies that help in reducing energy usage. Between 400 and 800 episodes, the decline in energy consumption is steady. It shows that the learning process is actively optimizing resource allocation. After 800 episodes, the rate of energy reduction slows, indicating that the model is approaching an optimal state. At 1000 episodes, energy consumption is minimized, and training results in only slight improvements.

Figure 14 presents a graph showing latency as the number of users increases. The graph compares five different schemes: CMRAD, CPRAD, DTOD, EECD, and DJTRA. The DTOD scheme has the highest latency values. The DJTRA scheme shows the lowest latency. The other schemes, CMRAD, CPRAD, and EECD, show intermediate latency values. The EECD scheme has slightly higher latency than CPRAD. As the number of users increases, the latency rises for all schemes in a nearly linear fashion. At 20 users, the latency ranges from 30 to 35 ms. At 100 users, it reaches nearly 90 ms for DTOD and around 85 ms for CMRAD and EECD. The DJTRA scheme shows better performance, maintaining lower latency at every user count, reaching just below 80 ms at 100 users. The difference in latency among the schemes remains consistent. It indicates that DJTRA is more efficient in managing task scheduling and resource allocation. As more users join the system, network congestion and processing delays increase. It leads to higher latency across all methods. The DJTRA scheme reduces latency by optimizing computational resources and minimizing network congestion.

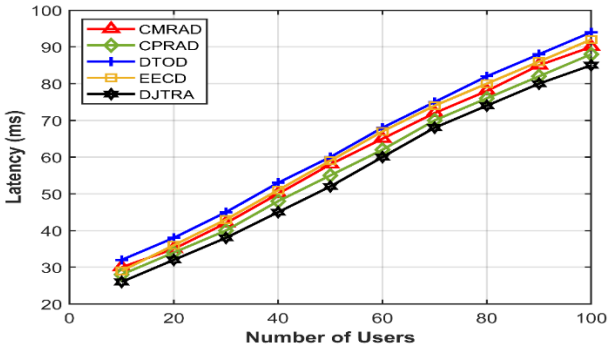


Fig. 14: Latency Versus Number of Users.

6. Conclusion

This paper proposed a DRL-based joint task offloading and resource allocation algorithm for cloud-edge-device networks. The proposed approach effectively balances computation workloads across cloud, edge, and device resources to minimize system cost, execution time, and energy consumption. The combination of DRL and numerical optimization methods achieves a dynamic and adaptive solution for task management in heterogeneous networks. Compared to heuristic-based approaches, our method optimizes the use of cloud, edge, and device computing power, leading to better load distribution and reduced latency. The integration of MDP and the SD3 reinforcement learning algorithm enables the proposed model to learn optimal policies over time. The ability to adapt to changing workloads and network dynamics improves the system continuously in its decision-making process. The replay buffer and target network mechanisms stabilize training, allowing the model to converge efficiently. Future work focuses on extending the model to accommodate real-time applications with strict latency constraints like autonomous driving and industrial automation. The integration of federated learning was also explored to enhance privacy and decentralized decision-making.

Acknowledgement

The authors would like to express their sincere gratitude to all those who contributed to the success of this research work.

References

[1] Y. Zhao, W. Wang, Y. Li, C. C. Meixner, M. Tornatoreand J. Zhang, “Edge computing and networking: A survey on infrastructures and applications,” IEEE Access, vol. 7, pp. 101213–101230, 2019. https://doi.org/10.1109/ACCESS.2019.2927538.

[2] A. Jahid, M. K. H. Monju, M. E. Hossainand M. F. Hossain, “Renewable energy assisted cost aware sustainable off-grid base stations with energy cooperation,” IEEE Access, vol. 6, pp. 60900–60920, 2018. https://doi.org/10.1109/ACCESS.2018.2874131.

[3] A. Salh, L. Audah, N. S. M. Shah, A. Alhammadi, Q. Abdullah, Y. H. Kim, S. A. Al-Gailani, S. A. Hamzah, B. A. F. Esmailand A. A. Almohammedi, “A survey on deep learning for ultra-reliable and low-latency communications challenges on 6g wireless systems,” IEEe Access, vol. 9, pp. 55098–55131, 2021. https://doi.org/10.1109/ACCESS.2021.3069707.

[4] S. Yao, M. Wang, Q. Qu, Z. Zhang, Y.-F. Zhang, K. Xuand M. Xu, “Blockchain-empowered collaborative task offloading for cloud-edge-device computing,” IEEE Journal on Selected Areas in Communications, vol. 40, no. 12, pp. 3485–3500, 2023. https://doi.org/10.1109/JSAC.2022.3213358.

[5] Z. Wu, X. Liu, Z. Ni, D. Yuanand Y. Yang, “A market-oriented hierarchical scheduling strategy in cloud workflow systems,” The Journal of Super-computing, vol. 63, pp. 256–293, 2013. https://doi.org/10.1007/s11227-011-0578-4.

[6] M. Ahmed, Y. Li, M. Waqas, M. Sheraz, D. Jinand Z. Han, “A survey on socially aware device-to-device communications,” IEEE Communications Surveys & Tutorials, vol. 20, no. 3, pp. 2169–2197, 2018. https://doi.org/10.1109/COMST.2018.2820069.

[7] R. Chai, J. Lin, M. Chenand Q. Chen, “Task execution cost minimization-based joint computation offloading and resource allocation for cellular d2d mec systems,” IEEE Systems Journal, vol. 13, no. 4, pp. 4110–4121, 2019. https://doi.org/10.1109/JSYST.2019.2921115.

[8] R. K. Bharti, D. Suganthi, S. Abirami, R. A. Kumar, B. Gayathriand S. Kayathri, “Optimal extreme learning machine based traffic congestion control system in vehicular network,” p. 597 – 603, 2022. https://doi.org/10.1109/ICECA55336.2022.10009111.

[9] S. Duan, D. Wang, J. Ren, F. Lyu, Y. Zhang, H. Wuand X. Shen, “Distributed artificial intelligence empowered by end-edge-cloud computing: A survey,” IEEE Communications Surveys & Tutorials, vol. 25, no. 1, pp. 591–624, 2023. https://doi.org/10.1109/COMST.2022.3218527.

[10] K. Bian and R. Priyadarshi, “Machine learning optimization techniques: a survey, classification, challengesand future research issues,” Archives of Computational Methods in Engineering, vol. 31, no. 7, pp. 4209–4233, 2024. https://doi.org/10.1007/s11831-024-10110-w.

[11] S. Ghyasuddin Hashmi, V. Balaji, M. U. Ahamed Ayoobkhan, M. Shabbir Alam, R. Anilkuamr, N. Nishant, J. Prasad Patraand A. Rajaram, “Machine learning-based renewable energy systems fault mitigation and economic assessment,” Electric Power Components and Systems, 2024. https://doi.org/10.1080/15325008.2024.2338557.

[12] H. Yang, M. Zheng, Z. Shao, Y. Jiangand Z. Xiong, “Intelligent computation offloading and trajectory planning for 3d target search in low-altitude economy scenarios,” IEEE Wireless Communications Letters, 2025. https://doi.org/10.1109/LWC.2025.3527005.

[13] Z. Wei, X. Yu, D. W. K. Ngand R. Schober, “Resource allocation for simultaneous wireless information and power transfer systems: A tutorial overview,” Proceedings of the IEEE, vol. 110, no. 1, pp. 127–149, 2021. https://doi.org/10.1109/JPROC.2021.3120888.

[14] X. Zhang et al., “A survey on edge computing: Architectures, applicationsand future directions,” IEEE Communications Surveys Tutorials, vol. 22, no. 2, pp. 1234-1267, 2020.

[15] J. Liu and Y. Lu, “Energy-efficient task offloading in edge computing networks,” IEEE Transactions on Network and Service Management, vol. 18, no. 3, pp. 789-802, 2021.

[16] K. Wang et al., “Latency optimization in mobile edge computing: A deep learning approach,” IEEE Journal on Selected Areas in Communications, vol. 39, no. 7, pp. 1345-1357, 2021.

[17] R. Gupta et al., “Resource allocation strategies for edge computing: A comprehensive survey,” IEEE Access, vol. 9, pp. 87632-87654, 2021.

[18] P. Sun et al., “Reinforcement learning for adaptive resource management in edge computing,” IEEE Transactions on Mobile Computing, vol. 20, no.6, pp. 1921-1934, 2021.

[19] Y. Huang et al., “Collaborative cloud-edge computing: Principles and applications,” IEEE Internet of Things Journal, vol. 8, no. 4, pp. 2981-2997,2021.

[20] Z. Tang et al., "Task distribution strategies for cloud-edge collaboration," IEEE Transactions on Cloud Computing, vol. 10, no. 3, pp. 1121-1135, 2022.
[21] B. Li et al., "Collaborative task offloading in edge networks," IEEE Transactions on Mobile Computing, vol. 21, no. 5, pp. 2141-2153, 2022.
[22] A. Kumar et al., "Device-to-device communication: Technologies, challengesand applications," IEEE Communications Magazine, vol. 58, no.12, pp. 74-80, 2020.
[23] L. Chen et al., "Task offloading in D2D-assisted mobile edge computing," IEEE Transactions on Wireless Communications, vol. 20, no. 8, pp. 5121-5134, 2021.
[24] H. Wang et al., "Deep reinforcement learning for network optimization," IEEE Communications Surveys & Tutorials, vol. 23, no. 1, pp. 234-256, 2021.
[25] R. Gowtham, V. Anand, Y. V. Suresh, K. L. Narasimha, R. A. Kumarand V. Saraswathi, "Enhancing incentive schemes in edge computing through hierarchical reinforcement learning," Journal of Engineering and Technology for Industrial Applications, vol. 11, no. 52, p. 226 – 236, 2025. https://doi.org/10.5935/jetia.v11i52.1637.