JOURNAL OF SOFT COMPUTING AND DATA MINING
e-ISSN: 2716-621X

JSCDM

Vol. 6 No. 3 (2025) 292-306
https://publisher.uthm.edu.my/ojs/index.php/jscdm

An Approach of Hopfield Network-Based Optimization for Task Offloading and Resource Allocation in Mobile Edge Computing

Le Van Hoa[1], Dang Thanh Chuong[2], Nguyen Hoang Ha[2], Vo Viet Minh Nhat[3]*

[1] *School of Hospitality and Tourism, Hue University, Hue City, VIETNAM*
[2] *University of Sciences, Hue University, Hue City, VIETNAM*
[3] *Department of Academic and Students' Affairs Hue University, Hue City, VIETNAM*

*Corresponding Author: vvmnhat@hueuni.edu.vn
DOI: https://doi.org/10.30880/jscdm.2025.06.03.019

Article Info

Received: 14 July 2025
Accepted: 17 November 2025
Available online: 30 December 2025

Keywords

MEC, task offloading, resource allocation, optimization, Hopfield network

Abstract

Mobile edge computing (MEC) has become an effective solution to alleviate the delay and energy burdens of cloud computing infrastructures. By enabling task execution at edge servers located near user devices, MEC reduces transmission delay and improves responsiveness. Nevertheless, uncontrolled task offloading may degrade overall system performance, and optimizing task offloading and resource allocation (TORA) can ensure efficient workload distribution. Prior studies have primarily applied metaheuristic techniques such as genetic algorithms (GA) and particle swarm optimization (PSO), while machine learning-based solutions remain relatively limited. This work introduces a Hopfield network-based optimization model, where the TORA target is reformulated as an energy function, and a Hopfield network performs the minimization process. Simulation experiments confirm that the Hopfield approach reduces both delay and energy consumption more effectively than GA and PSO, with its longer runtime. However, this limitation can be mitigated by advances in modern processing power.

1. Introduction

The fast advancement of communication technologies and computer networking has accelerated studies on deploying Internet of Things (IoT) systems. Today, IoT stands out as a core technology applied in diverse areas, including inventory management, manufacturing supervision, self-driving vehicles, and smart farming. With the widespread adoption of IoT, user devices are producing massive volumes of data, creating challenges in processing and analysis for planning, operations, and monitoring. Traditional solutions rely on transmitting all data to centralized clouds, but this overwhelming traffic often leads to network congestion and significant delays. Therefore, more effective solutions for cloud computing with big data are needed [1,2].

Edge computing has gained significant interest from both academia and industry as a practical extension to cloud computing. Edge computing uses a distributed computing approach to address the congestion and transmission latency issues caused by centralized cloud computing. Specifically, multiple distributed servers are deployed to receive tasks offloaded from user devices (UD), reducing the computing need to cloud servers and thus reducing network congestion. In addition, servers at the network edge (ES) are closer to user devices, which helps reduce data transmission delay. Figure 1 illustrates an example of a cloud-edge computing

architecture, in which an edge layer is added as a computing intermediate to reduce the load and delay of cloud computing by providing computing services closer to user devices [3,4].

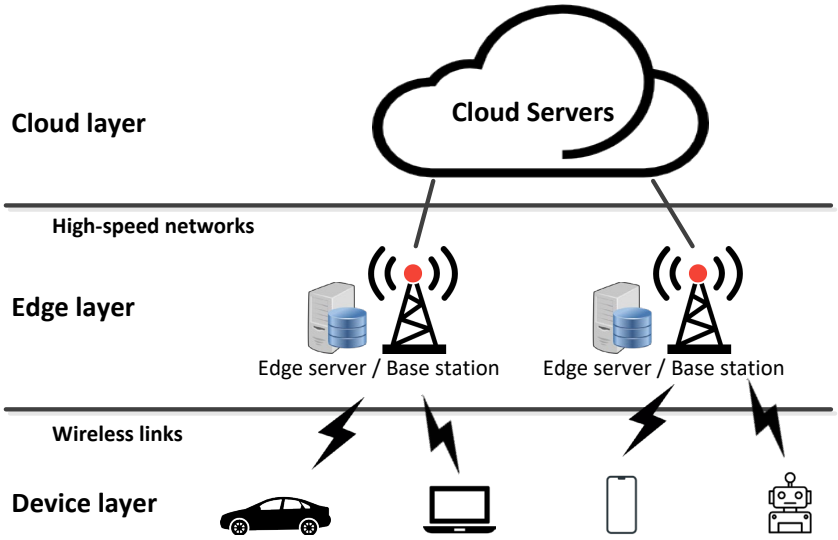

Fig. 1 *A typical architecture of cloud-edge computing*

In an edge computing system, tasks may be executed locally on user devices or offloaded to nearby edge servers. The main challenge lies in determining when local execution is sufficient and when offloading is more efficient. Task offloading typically occurs when a device with limited resources cannot process a task effectively, in which case the task is offloaded to an edge server that executes it and sends the result back. To satisfy performance requirements such as low latency and reduced energy consumption, efficient task offloading and resource allocation (TORA) strategies are essential. However, TORA optimization is NP-hard [5,6], and metaheuristic approaches are commonly employed to address this complexity [7,8]. Recently, a new direction has emerged where machine learning techniques, particularly deep reinforcement learning (DRL), are being applied to address TORA optimization [9,10]. The experimental results of these DRL models are impressive, but the experience-based learning approach always has its limitations.

This paper analyzes and evaluates the TORA optimization using artificial neural networks (ANN) in mobile edge computing (MEC) systems. Specifically, we propose a Hopfield network (HN)-based approach as a combinatorial optimization method to solve the TORA optimization. The Hopfield network is an associative memory whose "memorization" operation never increases Hopfield network energy. Leveraging this characteristic, the Hopfield Network (HN) is employed to reduce delay and energy consumption in TORA. Its performance is evaluated against genetic algorithms (GA)- and particle swarm optimization (PSO)-based models, with experimental results demonstrating that the HN-based approach achieves superior outcomes in both metrics. Although the HN-based approach requires more runtime, the powerful computing power of today's microprocessors has overcome this limitation and thus affirmed the strength of the HN-based approach.

The main contributions of this paper are summarized as follows:

- Formulating task offloading and resource allocation as a multi-objective optimization problem considering both delay and energy consumption;
- Transforming the multi-objective optimization function into a Hopfield Network energy equation;
- Designing an appropriate HN structure and developing a HN-based minimization algorithm; and
- Evaluating the effectiveness of the proposed HN-based optimization through experiments, with performance comparisons against GA- and PSO-based approaches.

The rest of the paper is organized as follows. Section 2 reviews and evaluates recent related works, focusing on machine learning-based TORA optimization approaches. Section 3 details the TORA optimization formulation, including the local computation model, the offloading model, the target function, and formulating the TORA optimization as a Hopfield network energy minimization. Section 4 presents the experimental process and compares the performance of the HN-based approach with metaheuristic-based models. Finally, Section 5 presents the conclusion and outlines potential directions for future research.

2. Related Works

Efficient task offloading and resource allocation is crucial for enhancing the performance of MEC systems. Since TORA is NP-hard, metaheuristic and machine learning-based methods have been widely adopted as practical solutions. This paper emphasizes machine learning approaches, particularly artificial neural networks, and reviews recent studies addressing the TORA optimization problem.

In [11], studied a MEC system where multiple UDs can offload computation tasks to an MEC server via wireless channels. The target function was formulated as the total cost of delay and energy consumption across UDs. To minimize this cost, task offloading and resource allocation were jointly optimized. Since finding an optimal policy in such a dynamic environment is difficult, the authors proposed a reinforcement learning-based framework, incorporating both Q-learning and Deep Reinforcement Learning (DRL) schemes. Simulation results demonstrated that their model achieved a significant reduction in total cost compared with traditional approaches.

Building on advances in deep neural networks (DNNs), the work in [12] investigated partial computation offloading to the cloud. They developed an adaptive DNN-based framework that considers mobile battery limitations and constrained cloud resources. Experimental results showed that the framework achieved an average speedup of 1.42 times, improved mobile execution time by 3.07 times, reduced energy consumption by 2.11 times, and shortened execution time by 4.26 times compared with baseline methods

The authors in [13] investigated a MEC network employing a binary offloading policy, where each task is either executed locally or fully offloaded to an MEC server. To handle time-varying wireless channel conditions, they developed an online algorithm that dynamically optimizes offloading and resource allocation decisions. Since conventional numerical methods struggle with the resulting complex combinatorial optimization problems, the authors proposed a Deep Reinforcement Learning-based Online Offloading (DRLOO) framework. This framework uses a scalable deep neural network to learn binary offloading strategies from experience, thereby eliminating the need for direct combinatorial optimization and substantially reducing computational complexity. An additional parameter-tuning procedure further improves efficiency. Experimental results confirmed that DRLOO achieves near-optimal performance while significantly lowering computation time compared to traditional methods

In [14], Yu et al. addressed computation offloading and resource allocation in a multi-user, multi-server MEC environment by proposing a deep learning-based strategy. The objective function was formulated by integrating computation and communication models to minimize task completion time and terminal energy consumption under delay constraints. In their framework, system utility and resource usage were modeled as rewards and penalties within a multi-agent deep reinforcement learning setting, and the Dueling-DQN algorithm was applied to determine the optimal resource allocation policy. Experimental results showed that with a learning rate of 0.001 and a discount factor of 0.90, the proposed approach achieved the best performance, reducing energy consumption by 52.18% and task completion time by 34.72%.

A study in [15] investigated task offloading scheduling, communication bandwidth allocation, and edge server resource management for multiple user devices (UEs) in an MEC system, aiming to minimize system latency and local energy consumption. The study examined both binary and partial task offloading by introducing a Dual-Agent TD3 (DA-TD3) algorithm based on deep reinforcement learning (DRL). Two cooperative agents were designed to jointly optimize task offloading, scheduling, and resource allocation. Experimental results demonstrated that DA-TD3 outperforms other schemes, significantly reducing latency and local energy consumption in both binary and partial offloading scenarios.

The authors in [16] proposed a deep reinforcement learning-based method for resource allocation and task offloading under coverage constraints. The approach focuses on offloading tasks that cannot be executed locally to edge servers while minimizing resource conflicts among UDs. The problem was analyzed considering system dynamics, coverage, handover decisions, communication relationships, and resource limitations. To balance energy consumption and execution time, an entropy weighting method was applied to optimize resource allocation. Experimental results indicated that the number of tasks and UDs has a significant impact on both execution time and energy consumption in the offloading process.

In [17], Qin et al. proposed a deep reinforcement learning approach based on dense clustering and ensemble learning (DCEDRL) for task offloading in MEC. The method leverages deep neural networks to explore the environment and employs ensemble learning to combine prediction results from multiple models. An optimized dense clustering technique is then applied to group computational tasks with similar characteristics, enhancing scheduling and resource allocation efficiency. Furthermore, priority weights are used to dynamically resample and adjust the strategy in real time, improving adaptability and robustness. Simulation results showed that DCEDRL reduces task offloading by over 21% compared with other algorithms.

Most recent studies rely on reinforcement learning, where neural networks serve as function approximators to facilitate the learning process. However, no prior work has utilized Hopfield Networks as a direct optimization engine for the TORA problem in MEC. Addressing this research gap, this paper reformulates TORA

as a HN energy minimization problem, designs an appropriate Hopfield Networkarchitecture, and develops a corresponding optimization algorithm. The following section provides a detailed description of the proposed HN-based TORA optimization.

It is worth noting that DRL and HN-based optimization represent two fundamentally different paradigms. DRL relies on experience-based learning, where optimal policies are gradually obtained through interaction with the environment, requiring iterative training and parameter tuning. In contrast, the HN-based approach is a deterministic energy minimization model that directly searches for the optimal configuration without learning from data. Therefore, their objectives and computational mechanisms differ significantly, and a direct comparison is not entirely appropriate. This work demonstrates that the HN-based formulation provides an alternative optimization direction for TORA in MEC systems, emphasizing convergence stability and lightweight implementation.

3. Hopfield Network-Based Optimization for TORA in MEC

3.1 MEC System Model

Consider the MEC system illustrated in Fig. 2, consisting of N user devices (UDs) and M edge servers (ESs). The edge servers are deployed at the edge and communicate wirelessly with the user devices.

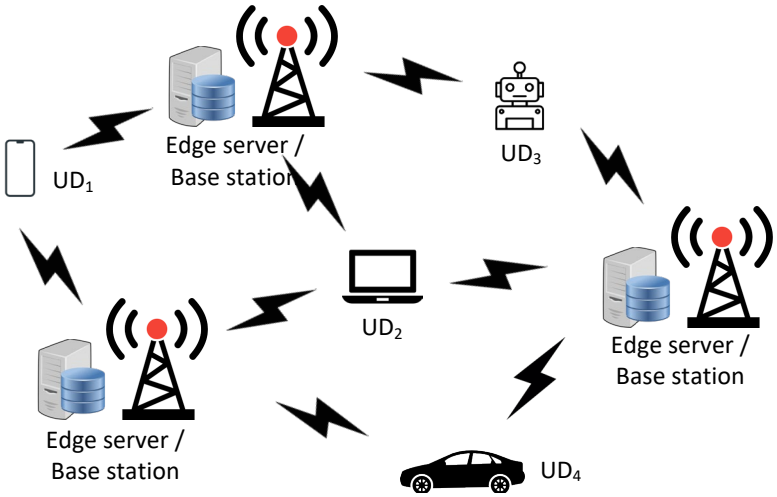

Fig. 2 *The MEC system model*

Each user device UD_i, $i \in \{1, 2, \ldots, N\}$, can be connected to some edge servers ES_j, $j \in \{1, 2, \ldots, M\}$. To represent the connection diagram between user devices and edge servers, a binary matrix $L \in R^{N \times M}$, as shown in Fig. 3, is defined, where each row represents a user device UD_i, and each column represents an edge server ES_j. A user device UD_i connected to an edge server ES_j is represented by $L_{i,j} = 1$ at the intersection of the corresponding row and column; otherwise, the value here is 0.

A task generated by an UD_i can be described by a quadruple $\langle s_i, c_i, E_i^{local}, t_i^{local} \rangle$, where s_i is the task size (e.g., code length or number of programs/subroutines), c_i is the number of CPU cycles per second (CPU cycles/s), E_i^{local} and t_i^{local} are the maximal consumed energy, and time, respectively, to complete the task.

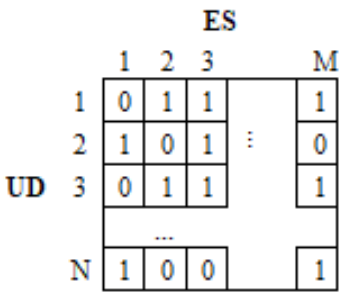

Fig. 3 *The connection matrix*

The capacity of a user device UD_i is represented by a triple $\left\langle f_i^{local}, P_i, E_i^{max} \right\rangle$, where f_i^{local}, P_i, and E_i^{max} are its computational power (CPU cycles/s), transmission power, and maximum energy, respectively.

An edge server ES_j maintains a triple $\left\langle \lambda_{i,j}, SR_j, f_j^{serv} \right\rangle$, where $\lambda_{i,j}$ is the offload rate of the task i, SR_j and f_j^{serv} are the service rate and capacity (CPU cycles/s), respectively, of the edge server ES_j.

3.2 Local Computing Model

Suppose that $f_i^{local} > 0$ denotes the local computational capacity (CPU cycles/s) of a user device UD_i, the execution time of a task processed locally can be expressed as follows:

$$t_i^{local} = \frac{c_i}{f_i^{local}} \tag{1}$$

The energy consumed by UD_i when executing a task locally can be estimated using the Shannon's formula [18], $\varepsilon = Kf^2$, where K is the energy factor determined by the chip architecture and f is the CPU frequency. Accordingly, if K_{UD} is represents the energy consumption factor of user device UD_i, the energy required to execute task i locally can be calculated as follows:

$$E_i^{local} = K_{UD}\left(f_i^{local}\right)^2 c_i \tag{2}$$

3.3 Offloading and Remote Execution Model

If a user device UD_i cannot execute a task locally, the task must be offloaded to a connected edge server ES_j. In this case, the total delay consists of three components: (i) the transmission time from the user device UD_i to the edge server ES_j ($t_{i,j}^{trans}$); (ii) the execution and queuing time at the edge server ES_j (t_j^{exe}); the feedback time for returning the result to the user device UD_i. Since the result size is typically much smaller than the input data size, the feedback time can be neglected [19].

The transmission rate is also calculated using the Shannon's formula [18], considering the mutual interference caused by user devices and background noise. The transmission rate of offloading from UD_i to ES_j is calculated by

$$R_{i,j} = B.\log\left(1 + \frac{P_i h_{i,j}}{N_0 + \sum_{i' \in k \setminus i} P_{i'} h_{i',j}}\right) \tag{3}$$

where B is the channel bandwidth, $h_{i,j}$ represents the channel gain between UD_i and ES_j, N_0 is the background noise, and the remainder of the denominator accounts for the interference among user devices.

Given the uplink transmission rate, the data transmission time from UD_i to ES_j can be expressed by

$$t_{i,j}^{trans} = \frac{s_i}{R_{i,j}} \tag{4}$$

Then, the energy consumed for data transmission from UD_i to ES_j can be given by

$$E_{i,j}^{trans} = P_i t_{i,j}^{trans} \tag{5}$$

The edge server is capable of providing computational services to multiple user devices. After receiving a task from UD_i, the edge server executes it and then returns the result to the device. The execution time to complete a task can be obtained as follows:

$$t_{i,j}^{exe} = \frac{1}{SR_j - \lambda_{i,j}} \tag{6}$$

where $\lambda_{i,j}$ and SR_j are the arrival and service rates, respectively, of ES_j.
Also using the formula in [18], the energy consumption to complete a task i at ES_j is:

$$E_{i,j}^{exe} = K_{ES}\left(f_j^{serv}\right)^2 c_i \tag{7}$$

where K_{ES} is the ES's energy consumption coefficient.
In summary, the time and energy for remote execution from an edge server ES_j are:

$$t_{i,j}^{remote} = t_{i,j}^{trans} + t_{i,j}^{exe} \tag{8}$$

$$E_{i,j}^{remote} = E_{i,j}^{trans} + E_{i,j}^{exe} \tag{9}$$

4. Problem Formulation

A user device UD_i can have connections to some ES_k, $0 \le k \le M$, but it must decide which ES_j to offload its task. Consider a decision matrix $D = [d_{i,j}]_{N\times M}$, where a row represents an UD_i and a column represents an ES_j. A task performed locally at an UD_i is represented by $d_{i,j} = 0$, while if the task is offloaded to ES_j, $d_{i,,j} = 1$. Accordingly, an UD_i can only offload one task to an ES_j and is represented by at most one cell per row with the value $d_{i,,j} = 1$, $\forall j \in [1, M]$, or $\sum_{j=1}^{M} d_{i,j} \le 1$. Conversely, an ES_j can receive multiple tasks offloaded from many UD_i and is represented by possibly multiple cells per column with the value $d_{i,j} = 1$ or $\sum_{i=1}^{N} d_{i,j} \le N$. As depicted in Fig. 4, UD_1 computes its task locally, so $d_{0,j} = 0$, $\forall j$; while UD_2 offloads one task to ES_3, then $d_{2,3} = 1$. With ES_M, it performs two tasks offloaded from UD_3 and UD_N, so $d_{3,M} = 1$ and $d_{N,M} = 1$.

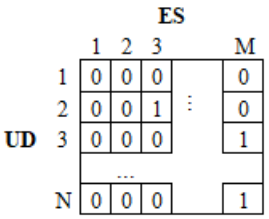

Fig. 4 *The decision matrix*

An UD_i can execute a task locally or offload it to ES_j, the system delay is that of the local computing or the remote (offloading) execution.

$$T_i^{total} = \left(1 - d_{i,j}\right) t_i^{local} + d_{i,j} L_{i,j} t_{i,j}^{remote} \tag{10}$$

Similarly, the total energy consumption corresponds to either local execution or remote execution.

$$E_i^{total} = \left(1 - d_{i,j}\right) E_i^{local} + d_{i,j} L_{i,j} E_{i,j}^{remote} \tag{11}$$

Therefore, a target function can be defined as the time and energy required to complete a task (of an UD_i) with the following formula:

$$F_i = \alpha \frac{T_i^{total}}{T_i^{\max}} + (1-\alpha)\frac{E_i^{total}}{E_i^{\max}} = \frac{\alpha}{T_i^{\max}}\left(\left(1 - d_{i,j}\right) t_i^{local} + d_{i,j} L_{i,j} t_{i,j}^{remote}\right) + \frac{1-\alpha}{E_i^{\max}}\left(\left(1 - d_{i,j}\right) E_i^{local} + d_{i,j} L_{i,j} E_{i,j}^{remote}\right) \tag{12}$$

Equation (12) formulates the TORA as a multi-objective optimization problem, simultaneously minimizing delay and energy consumption. By adjusting the weighting factor α, the system can flexibly prioritize between low latency and energy efficiency depending on the application requirements.

Here, $0 < \alpha < 1$ denotes the weight balancing delay and energy consumption. A larger α assigns higher importance to delay, whereas a smaller value emphasizes energy consumption. Each user device can select its own α according to the specific requirement, such as prioritizing faster execution or lower energy usage.

Based on the above analysis, the target function for the entire MEC system, considering all user devices UD_i, can be formulated as follows:

$$F = \sum_{i=1}^{N} F_i \tag{13}$$

$$\text{subject to } \sum_{j=1}^{M} d_{i,j} \le 1, \forall i \tag{13a}$$

$$\sum_{i=1}^{N} d_{i,j} \le N, \forall j \tag{13b}$$

$$\left(1 - \sum_{j=1}^{M} d_{i,j}\right) E_i^{local} \le E_i^{\max} \tag{13c}$$

The constraints in Equation (13) can be interpreted as follows: the first constraint (13a) implies that an UD_i can only offload one task to an ES_j and is represented by at most one cell per row with value d_{ij} = 1, $\forall j \in [1, M]$; the second constraint (13b) implies that an ES_j can receive multiple tasks offloaded from UD_i and is represented by possibly multiple cells per columns with value d_{ij} = 1; and the third constraint (13c) implies that if a task is processed locally, the energy of UD_i must be greater than the energy required to process it.

The two first constraints, (13a) and (13b), can be rewritten as follows:

$$\sum_{i,j=1}^{N,M} d_{i,j} \le N \tag{14}$$

Let $R_E = E_i^{\max} \big/ E_i^{local}$, the third constraint (13c) is rewritten as:

$$\left(1 - \frac{1}{N} \sum_{i,j=1}^{N,M} d_{i,j}\right) \le R_E \tag{15}$$

Combining the target function (13) and the constraints (14) and (15), we have:

$$F = A_1 \sum_{i=1}^{N} F_i + A_2 \left(\sum_{i,j=1}^{N,M} d_{i,j} - N\right)^2 + A_3 \left(\left(1 - \frac{1}{N} \sum_{i,j=1}^{N,M} d_{i,j}\right) - R_E\right)^2 \tag{16}$$

where A_i, i = [1,2,3], are the target function weights and constraints, $\sum_{k=1}^{3} A_k = 1$.

Let $C_{i,k}$ be a binary square matrix with $C_{i,k}$ = 1 if $i = k$ and $C_{i,k}$ = 0 if $i \ne k$, Equation (16) is rewritten as:

$$\begin{aligned} F = {} & A_1 \sum_{i=1}^{N} \left(\frac{\alpha}{T_i^{\max}} \left(\left(1 - d_{i,j}\right) t_i^{local} + d_{i,j} L_{i,j} t_{i,j}^{remote} \right) + \frac{1-\alpha}{E_i^{\max}} \left(\left(1 - d_{i,j}\right) E_i^{local} + d_{i,j} L_{i,j} E_{i,j}^{remote} \right) \right) \\ & + A_2 \left(\sum_{i,j=1}^{N,M} \sum_{k,h=1}^{N,M} C_{i,k} C_{j,h} d_{i,j} d_{k,h} - 2 \sum_{i,j=1}^{N,M} d_{i,j} + N^2 \right) \\ & + A_3 \left(\left(1 - \frac{2}{N} \sum_{i,j=1}^{N,M} d_{i,j} + \frac{1}{N^2} \sum_{i,j=1}^{N,M} \sum_{k,h=1}^{N,M} C_{i,k} C_{j,h} d_{i,j} d_{k,h} \right) - 2R_E \left(1 - \frac{1}{N} \sum_{i,j=1}^{N,M} d_{i,j}\right) + R_E^2 \right) \end{aligned} \tag{17}$$

$$\begin{aligned} F = {} & \sum_{i,j=1}^{N,M} \sum_{k,h=1}^{N,M} \left(\left(A_2 + \frac{A_1}{N^2}\right) C_{i,k} C_{j,h} \right) d_{i,j} d_{k,h} \\ & + \sum_{i,j=1}^{N,M} \left(A_1 \left(\frac{\alpha}{T_i^{\max}} \left(L_{i,j} t_{i,j}^{remote} - t_i^{local} \right) + \frac{1-\alpha}{E_i^{\max}} \left(L_{i,j} E_{i,j}^{remote} - E_i^{local} \right) \right) - 2A_2 + \frac{2A_1}{N} \left(R_E - 1 \right) \right) d_{i,j} \\ & + \left(A_1 \left(\frac{\alpha}{T_i^{\max}} t_i^{local} + \frac{1-\alpha}{E_i^{\max}} E_i^{local} \right) + A_2 N^2 + A_3 \left(1 - R_E\right)^2 \right) \end{aligned} \tag{18}$$

Hopfield network-based optimization is a process of minimizing an error function, also known as the Hopfield network energy, which is given by Equation (20), with the principle "*minimum energy, optimal solution*" as demonstrated in [20], [21]:

$$E = -\frac{1}{2} \sum_{i,j=1}^{N,M} \sum_{k,h=1}^{N,M} w_{i,j;k,h} x_{i,j} x_{k,h} + \sum_{i,j=1}^{N,M} \theta_{i,j} x_{i,j} \tag{20}$$

where $w_{i,j;k,h}$ denotes the connection weight between the neuron at position (i,j) and the neuron at position (k,h), $\theta_{i,j}$ represents the activation threshold of the neuron at (i,j) and $x_{i,j}$ is the input value of the neuron at (i,j).i,

By comparing Equations (19) and (20), the weights $w_{i,j;k,h}$ and the activation threshold $\theta_{i,j}$ are derived:

$$w_{i,j;k,h} = -2\left(A_2 + \frac{A_3}{N^2}\right)C_{i,k}C_{j,h} \tag{21}$$

$$\theta_{i,j} = A_1\left(\frac{\alpha}{T_i^{\max}}\left(L_{i,j}t_{i,j}^{remote} - t_i^{local}\right) + \frac{1-\alpha}{E_i^{\max}}\left(L_{i,j}E_{i,j}^{remote} - E_i^{local}\right)\right) - 2A_2 + \frac{2A_3}{N}\left(R_E - 1\right) \tag{22}$$

5. Hopfield Network Architecture and Optimization Process

The Hopfield network used for TORA optimization is a 2-dimensional neural network architecture, in which each neuron connects to all other neurons. There is no autoregressive connection at each neuron. With N user devices and M edge servers, the Hopfield network is a matrix of $M \times N$ neurons, as shown in Fig. 5. Each input i represents a decision of UD_i to compute locally or offload to an edge server, corresponding to the values $d_{i,j}$ is equal to 0 or 1. The output value of the highest activated neuron corresponds to a ES_j selected to perform the task offloaded from UD_i.

The optimization process corresponds to minimizing the Hopfield network energy, which is performed by the following steps:

1. **Random selection:** Randomly select a user device UD_i. The neurons belonging to the corresponding column are stimulated by inputs of 1 (Fig. 5) , and the inputs of the remaining neurons have a value of 0;
2. **Activation computation:** Calculate the activation of the neurons in this column and identify the neuron with the highest activation that exceeds the threshold $\theta_{i,j}$, called the above-threshold activation peak;
3. **Task offloading decision:**
   - If a unique neuron reaches the above-threshold activation peak, the corresponding ES_j is selected to execute the task offloaded from UD_i.
   - If multiple neurons share the same above-threshold activation peak, one ES_j is randomly selected (since UD_i offloads only one task to an edge server).
   - If no neuron in the column exceeds $\theta_{i,j}$, the task is executed locally on UD_i.
4. **Iteration:** Repeat Steps 1–3 until all UD_i have been considered.

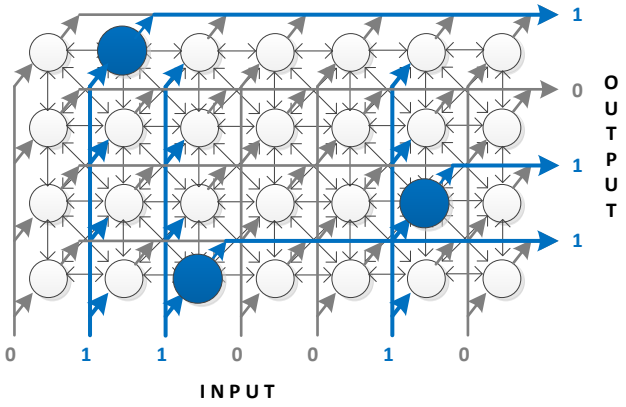

Fig. 5 *The Hopfield network architecture for TORA optimization, where each neuron with the highest activation determines the edge server (corresponding row output) to execute the task offloaded from a user device (corresponding column input)*

Hopfield network-based optimization has a relatively high computational complexity: $O\left(\left(N \times M\right)^2\right)$, so the computational time is significant. However, with the development of current microprocessor technology, the above computational complexity is no longer a concern.

The HN-based optimization algorithm iteratively updates neuron states to minimize the Hopfield network energy function defined in Eq. (20). The output of each neuron represents a decision value, $d_{i,j}$, corresponding to whether the task is offloaded to an edge server ES_j or executed locally by UD_i. The following is the HN-based optimization algorithm.

Algorithm 1. HN-based optimization for TORA Optimization:

Input: User devices $UD = \{UD_1, UD_2, \ldots, UD_N\}$,
Edge servers $ES = \{ES_1, ES_2, \ldots, ES_M\}$,
Activation thresholds $\theta_{i,j}$.

Output: Decision matrix $D = (d_{i,j})_{N \times M}$.

for each user device $UD_i \in UD$ **do**
  Randomly select an UD_i and stimulate all neurons in the corresponding column;
  Compute the activation of all neuron in the column;
  Identify the neuron(s) with the above-threshold activation.
  if (only one neuron reaches the above-threshold activation peak) **then**
    $d_{i,j} = 1$; // UD_i uploads its task to ES_j
    $d_{i,h} = 0, \forall h \neq j$;
  else if (multiple neurons share the above-threshold activation) **then**
    Randomly select one neuron: j;
    $d_{i,j} = 1$; // UD_i uploads its task to ES_j
    $d_{i,h} = 0, \forall h \neq j$;
    else
      $d_{i,j} = 0, \forall j$; // Execute the task locally on UD_i
    end if
  end if
end for
Repeat until all user devices are processed or the network energy stabilizes (no further updates), yielding the decision matrix D.

6. Simulation and Analysis

6.1 Simulation Environment and Parameters

All experiments were implemented in Python 3.10 using the NumPy and Matplotlib libraries for numerical computation and visualization on a PC equipped with an Intel(R) Core (TM) i5, 2.40GHz, 8GB RAM PC. The HN-based TORA optimization model (In short, HN-based approach) was evaluated against widely used metaheuristic methods, namely the Genetic Algorithm (GA) and Particle Swarm Optimization (PSO). For both GA and PSO, the optimization process begins with a population of candidate solutions (individuals) and iteratively searches for the optimal solution in the space of all possible candidates. In our experiments, the population size was fixed at 30, and the stopping condition was defined as either (i) reaching 100 generations or (ii) observing no improvement in fitness for 10 consecutive generations.

The general configuration parameters are summarized in Table 1, while the specific parameters for each model are detailed in Table 2.

Table 1 *General simulation parameters*

Parameter	Description	Value
N	Number of user devices (UD)	100
M	Number of Edge Servers (ES)	4
d_i	Task size (MB) generated by UD_i	[5, 15]
c_i	Number of CPU cycles to complete the task i	[2, 8] Gc
f_i	UD_i's CPU speed (MHz)	[200, 800]
T_i^{max}	Maximum time allowed to process the task i	0.08 s
E_i^{max}	Maximum energy consumed to process the task i	0.02 J
P_i	UD_i's power consumption (W)	[0.2, 0.6]
μ_j	ES_j's CPU speed (GHz)	[4, 10]
A	The trade-off between delay and energy	0.5
B	Transmission bandwidth	1×10^6 Hz
N_0	Background noise	1×10^{-7}
h_{ij}	Transmission loss coefficient	[0.01, 0.1]
$D=(d_{i,j})_{N\times M}$	Decision matrix, $d_{i,j} = 1$ if UD_i offloads task to ES_j, otherwise $d_{i,j} = 0$	[0, 1]

Table 2 *Configuration parameters specific to each GA-, PSO-, and HN-based approaches*

Algorithm	Parameter	Description	Value
GA	*mutation_rate*	Mutation probability	0.05
	crossover	Crossover operator	one-point
	selection	Selection operator	tournament 2
PSO	*W*	Inertial weight	0.7
	c_1, c_2	cognitive (c_1) and social (c_2) acceleration coefficients	1.5, 1.5
Hopfield	*update_ratio*	State update rate per round	0.2
	bias_offload	Priority factor when offloading	0.98

To assess the effectiveness of GA-, PSO-, and HN-based approaches in solving the TORA optimization problem, simulations were conducted based on the following performance criteria:

- **Delay convergence** – evaluates how quickly the average system delay stabilizes during the optimization process.
- **Energy consumption convergence** – measures the reduction and stabilization of the system's total energy consumption
- **Target convergence** – examines the convergence behavior of the overall target function (a weighted combination of delay and energy consumption)
- **Normalized radar comparison** – provides a comprehensive comparison across multiple performance metrics using a normalized radar chart.
- **Resource allocation** – analyzes the efficiency and fairness of task allocation across edge servers.

6.2 Delay Convergence

As can be seen in Fig. 6, which depicts the delay reduction progress of the three models, the HN-based approach achieves exceptionally fast convergence and stabilizes at an average value of around 0.301s after only a few iterations. With the PSO-based model, the delay drops rapidly initially but then "freezes" around 0.469s. The delay of the GA-based model fluctuates wildly and must go through many iterations to converge at an average of around 0.400s. Based on the results in Fig. 6, the HN-based approach reduces the delay more effectively than the GA- and PSO-based models and maintains a stable state, which is suitable for delay-sensitive MEC systems.

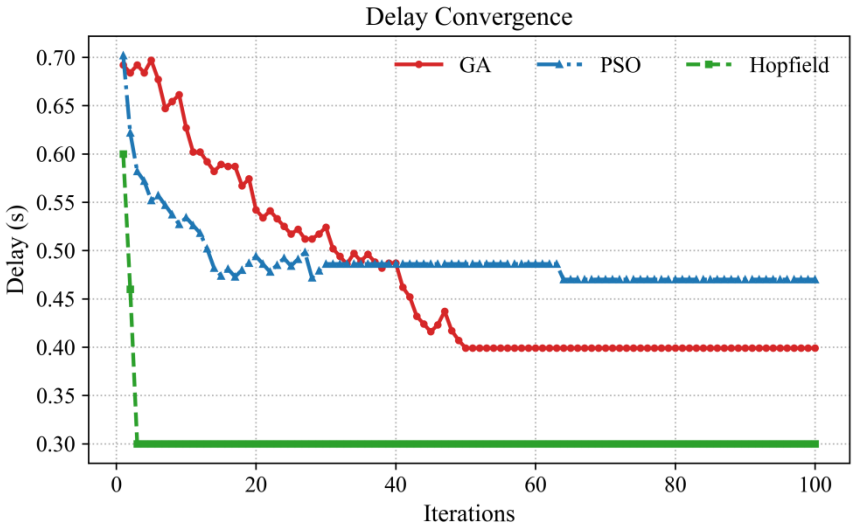

Fig. 6 *Delay convergence of GA-, PSO-, and HN-based approaches*

The superior efficiency of the HN-based approach compared with GA- and PSO-based models arises from its direct mapping mechanism. Specifically, the target function and system constraints are reformulated into the Hopfield network energy function, where the connection weights and activation thresholds are explicitly determined at the outset. This eliminates the need for iterative parameter tuning or a training phase, as required in conventional neural network models, thereby reducing computational overhead and accelerating convergence. For the GA-based approach, the inherent randomness in crossover and mutation operations

promotes population diversity and helps escape local minima. However, this randomness also leads to frequent oscillations and consequently slower convergence. In contrast, the PSO-based approach relies heavily on orientation and exploitation of information from the current best solutions, which enables faster convergence. Nonetheless, this exploitation tendency increases the risk of premature convergence and getting trapped in local optima. These distinct characteristics of GA and PSO are clearly illustrated in Fig. 6.

6.3 Energy Consumption Convergence

In terms of energy consumption, the results from Fig. 7 show that the HN-based approach continues to demonstrate its superiority by quickly reaching a low of about 0.305J and maintaining a stable level at this level. the GA-based approach fluctuates around 0.408J, while the PSO-based approach consumes the most energy, averaging about 0.475J. Comparing the correlation, the HN-based approach saves about 24–36% of energy compared to the GA and PSO-based approaches. This result is important because MEC systems need to conserve their limited energy to serve many user devices.

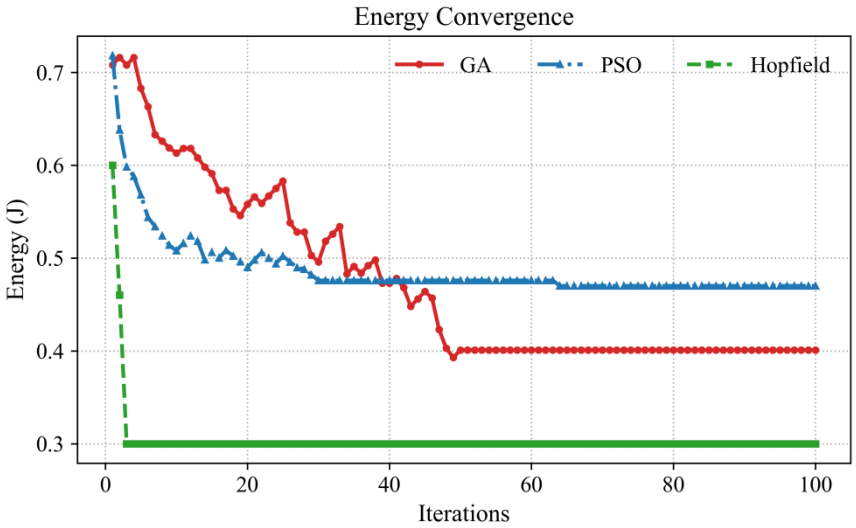

Fig. 7 *Energy convergence of GA-, PSO-, and HN-based approaches*

To explain this result, in the HN-based approach, the connection weights ($w_{i,j;k,h}$) and activation thresholds ($\theta_{i,j}$) are determined (Equations (21) and (22)) to reduce delay and energy consumption and simultaneously (Equation (13)), instead of optimizing each criterion separately. Therefore, when the Hopfield network energy reaches a minimum, this also means that the local computation or offloading decisions also simultaneously reach a minimum energy consumption and delay. However, with the GA- and PSO-based approaches operating on population evolution and trajectory velocity, there is no guarantee of a direct link between energy consumption and delay, so there is a fluctuation in the trade-off between them.

6.4 Target Convergence

As shown in Equation (13), the optimization aims two targets: delay and energy consumption. Assuming both targets are given equal weight, i.e., $\alpha = 0.5$, Fig. 8 shows that the HN-based approach quickly reaches the lowest target value around 0.300. In contrast, the GA- and PSO-based approaches stop at around 0.400 and 0.470, respectively. The fast convergence speed and lower achieved target value reflect the performance of the HN-based approach in optimizing delay and energy consumption simultaneously.

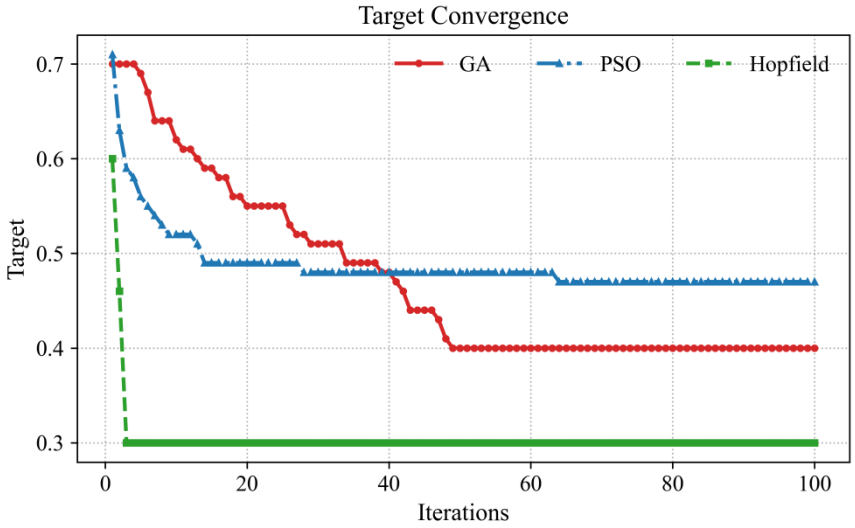

Fig. 8 *Target convergence of GA-, PSO-, and HN-based approaches*

The results of target convergence show that, thanks to directly mapping the target function (delay–energy) to the Hopfield network energy function, the algorithm is essentially solving the global optimization problem at the same level between delay and energy. The simultaneous interaction between neurons ensures that both criteria are optimized simultaneously, instead of alternating as in the GA-based approach (equilibrium by selection) or the PSO-based approach (based on the velocity trajectory of the swarm).

6.5 Normalized Radar Comparison

Figure 9 compares four normalized metrics: delay, energy, target, and runtime. Among the three methods, the HN-based approach demonstrates superior performance in delay, energy consumption, and target value. However, this comes at the cost of a higher runtime (2.3468s). The GA-based approach provides moderate performance across all four metrics, without excelling in any particular aspect. By contrast, the PSO-based approach achieves the fastest runtime (0.3241s), but its performance is inferior in the other three metrics, namely delay, energy consumption, and target value.

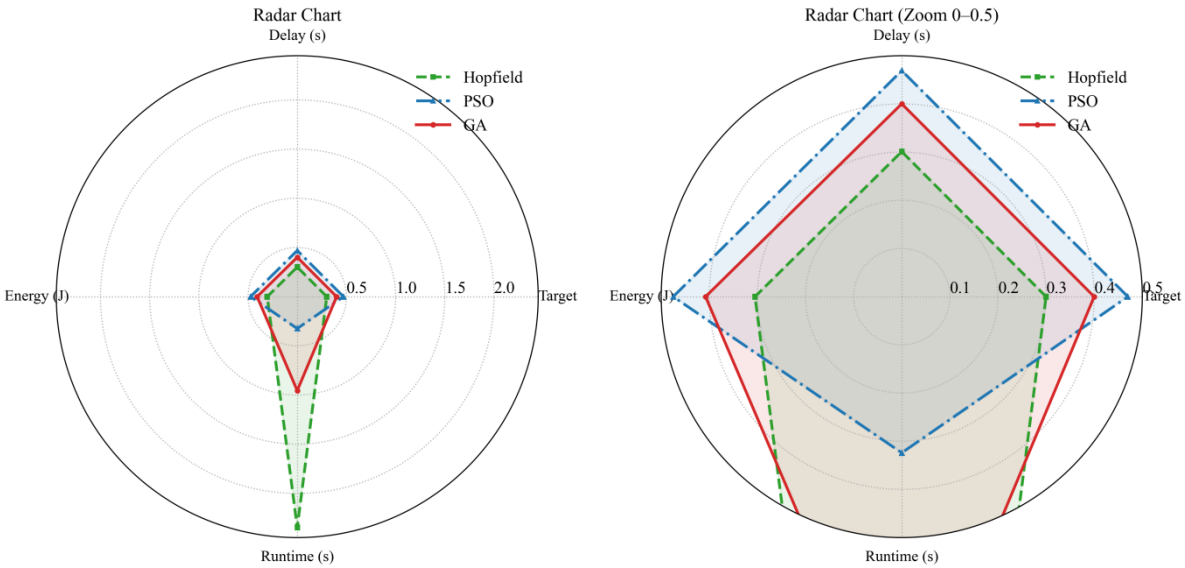

Fig. 9 *Normalized radar comparison of GA-, PSO-, and HN-based approaches*

The results in Figure 9 reflect a characteristic of Hopfield, which is the high computational cost due to the large number of neurons. However, it always ensures convergence to the minimum energy. With recent advances in the execution speed of microprocessors, especially GPUs, the runtime will obviously be significantly reduced.

6.6 Resource Allocation

Table 3 details the resource allocation results: local execution or offloading to edge servers (ES_1–ES_4) and performance metrics (Target, Delay, Energy, Runtime) of the three approaches: GA-, PSO-, and HN-based.

Table 3 *Comparison of numerical results of the GA-, PSO-, and HN-based approaches*

Model	Local	ES_1	ES_2	ES_3	ES_4	Target	Delay (s)	Energy (J)	Runtime (s)
Hopfield	70	7	7	8	8	0.300004	0.500008	0.500530	2.3468
PSO	53	1	6	11	26	0.470002	1.400014	1.400105	0.3241
GA	60	11	10	12	7	0.410002	1.300093	1.267801	0.9554

The results indicate that the HN-based approach achieves a more balanced and efficient resource allocation strategy compared to GA and PSO. Specifically, out of 100 tasks generated from user devices, 70 tasks are processed locally, and each edge server only receives 7 or 8 offloaded tasks. Thanks to that, the system avoids overloading at edge servers and reduces remote execution latency (transmission and remote processing). Meanwhile, the PSO-based approach allocates resources unbalanced, putting up to 26 tasks into ES_4 while ES_1 has only 1 task, creating a "bottleneck" phenomenon. The GA-based approach provides a more even distribution than PSO (7 to 12 tasks per edge server), but is still not as even as Hopfield in balancing local processing and offloading.

6.7 Implementation of HN-based TORA Optimization Appoach in IoT Systems

The proposed HN-based optimization approach can be deployed in Mobile Edge Computing (MEC) and Internet of Things (IoT) systems as a decision engine for task offloading and resource scheduling. In such deployments, the Hopfield network weights and thresholds can be precomputed offline based on system characteristics (e.g., number of user devices, edge servers, available bandwidth, and CPU capacity). Once deployed, the HN-based optimization approach can operate at the edge controller or gateway level, enabling near real-time offloading decisions without extensive online training.

The HN-based optimization, therefore, is suitable for latency-sensitive IoT applications such as intelligent transportation systems, industrial IoT, and smart surveillance, where fast and energy-efficient task distribution is essential. Moreover, due to its deterministic convergence and relatively low computational overhead, the HN-based optimizer can be integrated into existing MEC orchestration platforms (e.g., Kubernetes-based edge clusters) as a module for adaptive workload management.

7. Conclusion

The TORA optimization plays a critical role in ensuring the efficiency of MEC systems, enabling high-quality services while satisfying heterogeneous user demands. With the rapid growth of IoT devices, solving TORA as a multi-objective optimization problem has become increasingly challenging. This paper presented a Hopfield neural network-based optimization approach in which the TORA problem is reformulated as an energy minimization model. Comparative experiments with GA- and PSO-based approaches demonstrated that the HN-based approach achieves better performance in delay, energy consumption, and overall objective optimization. Although the model exhibits a relatively high computational runtime, advances in modern microprocessor and parallel processing technologies have significantly mitigated this limitation.

While the current evaluation was conducted in a small-scale MEC setting with 100 user devices and four edge servers, the proposed HN-based framework is inherently scalable. Its energy-function formulation and parallel neuron update mechanism allow extension to large-scale and heterogeneous MEC–IoT environments with minimal structural modification. Future work will validate scalability and robustness under large-scale, dynamic conditions involving mobile devices and heterogeneous edge nodes. In addition, integrating Hopfield networks with deep learning or reinforcement learning models represents a promising direction to leverage deterministic convergence and adaptive learning capabilities for real-time IoT applications.

Acknowledgement

This work was partially supported by Hue University under the Core Research Program, Grant No. NCTB.DHH.2025.09.

Conflict of Interest

Authors declare that there is no conflict of interests regarding the publication of the paper.

Author Contribution

L. V. Hoa: ***investigation, methodology, coding, data analysis****; D. T. Chuong:* ***data curation, writing an original draft****; N. H. Ha:* ***data curation, data analysis****; V. V. M. Nhat:* ***conceptualization, research design, reviewing and editing, project administration****; All authors have read and agreed to the published version of the manuscript.*

Declaration of AI Use in Manuscript Preparation

The authors used Grammarly to assist in grammar checking and language editing. All content generated was reviewed and verified by the authors, who take full responsibility for the final submission.

References

[1] Fatema Vhora & Jay Gandhi (2020) A Comprehensive Survey on Mobile Edge Computing: Challenges, Tools, Applications, Proceedings of the 4th International Conference on Computing Methodologies and Communication (ICCMC), 49–55, https://doi.org/10.1109/ICCMC48092.2020.ICCMC-0009

[2] Francesco Cosimo Andriulo, Marco Fiore, Marina Mongiello, Emanuele Traversa & Vera Zizzo (2024) Edge computing and cloud computing for Internet of Things: A review, *Informatics, 11(4)*, 71, https://doi.org/10.3390/informatics11040071

[3] Lina A. Haibeh, Mustapha C. E. Yagoub & Abdallah Jarray (2022) A survey on mobile edge computing infrastructure: Design, resource management, and optimization approaches, *IEEE Access, 10*, 27591–27610, https://doi.org/10.1109/ACCESS.2022.3152787

[4] Chuan Feng, Pengchao Han, Xu Zhang, Bowen Yang, Yejun Liu & Lei Guo (2022) Computation offloading in mobile edge computing networks: A survey, *Journal of Network and Computer Applications, 202*, 103366, https://doi.org/10.1016/j.jnca.2022.103366

[5] Maryam Keshavarznejad, Mohammad Hossein Rezvani & Sepideh Adabi (2021) Delay-aware optimization of energy consumption for task offloading in fog environments using metaheuristic algorithms, *Cluster Computing, 24(3)*, 1825–1853, https://doi.org/10.1007/s10586-020-03230-y

[6] Baskar Vijayaram & Venkatesh Vasudevan (2022) Wireless edge device intelligent task offloading in mobile edge computing using hyper-heuristics, *EURASIP Journal on Advances in Signal Processing, 2022(1)*, 1–23, https://doi.org/10.1186/s13634-022-00965-1

[7] Qiang You & Bo Tang (2021) Efficient task offloading using particle swarm optimization algorithm in edge computing for industrial internet of things, *Journal of Cloud Computing, 10(1)*, 1–15, https://doi.org/10.1186/s13677-021-00256-4

[8] Mohamed Elkawkagy, Ibrahim A. Elgendy, Sid-Ali Allaoua Chelloug & Hamada Elbeh (2025) Genetic algorithm-driven joint optimization of task offloading and resource allocation for fairness-aware latency minimization in mobile edge computing, *IEEE Access, 13*, 237–248, https://doi.org/10.1109/ACCESS.2025.3584971

[9] J Jie Chen & Xiaodong Ran (2019) Deep learning with edge computing: A review, Proceedings of the IEEE, 107(8), 1655–1674, https://doi.org/10.1109/JPROC.2019.2921977

[10] Shiqiang Wang, Min Chen, Xiangyu Liu, Chao Yin, Shuguang Cui & H. Vincent Poor (2021) A machine learning approach for task and resource allocation in mobile-edge computing-based networks, *IEEE Internet of Things Journal, 8(3)*, 1358–1372, https://doi.org/10.1109/JIOT.2020.3011286

[11] Jie Li, Hui Gao, Tao Lv & Yilong Lu (2018) Deep reinforcement learning based computation offloading and resource allocation for MEC, IEEE Wireless Communications and Networking Conference (WCNC), 1–6, https://doi.org/10.1109/WCNC.2018.8377343

[12] Amir Erfan Eshratifar & Massoud Pedram (2018) Energy and performance efficient computation offloading for deep neural networks in a mobile cloud computing environment, Proceedings of the ACM Great Lakes Symposium on VLSI (GLSVLSI), 111–116, https://doi.org/10.1145/3194554.3194565

[13] Liang Huang, Suzhi Bi & Ying-Jun Angela Zhang (2020) Deep reinforcement learning for online computation offloading in wireless powered mobile-edge computing networks, *IEEE Transactions on Mobile Computing, 19(11)*, 2581–2593, https://doi.org/10.1109/TMC.2019.2928811

[14] Zhiyuan Yu, Xin Xu & Wen Zhou (2022) Task offloading and resource allocation strategy based on deep learning for mobile edge computing, *Computational Intelligence and Neuroscience, 2022*, Article 1427219, https://doi.org/10.1155/2022/1427219

[15] Jiahui Dong, Kun Pan, Chao Zheng, Lei Chen, Shuo Wu & Xiaoliang Zhang (2023) A dual-agent approach for coordinated task offloading and resource allocation in MEC, *Journal of Electrical and Computer Engineering, 2023,* Article 6134837, https://doi.org/10.1155/2023/6134837

[16] Dongxue Zhang, Xin Li, Bin Wei & Yansong Shi (2024) A joint coverage constrained task offloading and resource allocation method in MEC, *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences, E107.A(8)*, 1277–1285, https://doi.org/10.1587/transfun.2023EAP1139

[17] Yuhan Qin, Jiahui Chen, Ling Jin, Rui Yao & Zhiqiang Gong (2025) Task offloading optimization in mobile edge computing based on a deep reinforcement learning algorithm using density clustering and ensemble learning, *Scientific Reports, 15(1)*, 1–25, https://doi.org/10.1038/s41598-024-84038-3

[18] Eric P. Caspers, Paul P. Castiglione, Kyung P. Choi & David W. Matthews (2013) Analysis of information and power transfer in wireless communications, *IEEE Antennas and Propagation Magazine, 55(3),* 82–95, https://doi.org/10.1109/MAP.2013.6586629

[19] Guangming Jiang, Rui Huang, Zhi Bao & Guowei Wang (2024) A task offloading and resource allocation strategy based on multi-agent reinforcement learning in mobile edge computing, *Future Internet, 16(9)*, 333, https://doi.org/10.3390/fi16090333

[20] Solomon Tsegaye, Fikadu Shewarega & Getachew Bekele (2021) Hopfield neural network-based security constrained economic dispatch of renewable energy systems, *EAI Endorsed Transactions on Energy Web, 8(35)*, 1–14, https://doi.org/10.4108/eai.25-1-2021.168224

[21] Le Van Hoa, Nguyen Van Tung & Vo Van Minh Nhat (2024) An approach to Hopfield network-based energy-efficient RFID network planning, *Cybernetics and Information Technologies, 24(2)*, 50–66, https://doi.org/10.2478/cait-2024-0015