Computing (2025) 107:34
https://doi.org/10.1007/s00607-024-01380-0

REGULAR PAPER

ALBLA: an adaptive load balancing approach in edge-cloud networks utilizing learning automata

Mehdi Ghorbani[1] · Navid Khaledian[2] · Setareh Moazzami[3]

Received: 16 August 2024 / Accepted: 22 November 2024

Abstract
In the Internet of Things (IoT) era, the demand for efficient and responsive computing systems has surged. Edge computing, which processes data closer to the source, has emerged as a promising solution to address the challenges of latency and bandwidth limitations. However, the dynamic nature of edge environments necessitates intelligent load-balancing strategies to optimize resource utilization and minimize service latency. This paper proposes a novel load-balancing approach that leverages learning automata (LA) to distribute real-time tasks between edge and cloud servers dynamically. By continuously learning from past experiences, the algorithm adapts to changing workloads and network conditions, ensuring optimal task allocation. The proposed algorithm employs a Service Time Measurement (STM) metric to evaluate servers' performance and make informed decisions about task distribution. The algorithm effectively balances the workload between edge and cloud servers by considering factors such as task complexity, server capacity, and network latency. Through extensive simulations, we demonstrate the superior performance of our proposed algorithm compared to existing techniques. Our approach significantly reduces average service time, minimizes task waiting time, optimizes network traffic, and increases the number of successful task executions on edge servers. Compared to previous approaches that partially addressed workload balancing, ALBLA offers a more comprehensive solution that optimizes resource utilization and minimizes energy consumption. Additionally, ALBLA's adaptive nature makes it well-suited for dynamic edge-cloud environments with fluctuating workloads. Our proposed approach contributes to developing more efficient, responsive, and scalable IoT systems by addressing the challenges inherent in edge computing environments.

Keywords Edge computing · Cloud computing · Load balancing · Learning automata · Machine learning

Mathematics Subject Classification 68M20

Extended author information available on the last page of the article

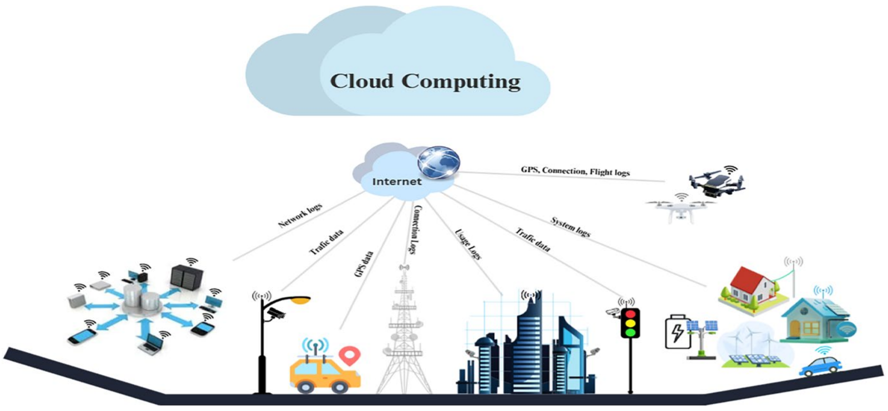

Fig. 1 A general architecture of edge cloud environment and end-users

1 Introduction

The growth and development of Internet of Things (IoT) devices and smart mobile phones have led to a significant increase in data generated at the network's edge [1–3]. However, these devices' limited computing, storage, and battery life hinder their effective implementation. Cloud computing has been proposed as a solution for outsourcing computational tasks [4, 5]. Nevertheless, the significant distance between the cloud and the network's edge presents challenges such as high bandwidth consumption, latency, and security and privacy concerns [6]. Mobile edge computing (MEC) has emerged as a promising solution to address these challenges, particularly with the advent of fourth and fifth-generation mobile communication technologies [7]. This approach involves deploying numerous servers at the network's edge to process outsourced computational tasks closer to users, thereby reducing latency and enhancing the quality of service experienced by users. A general architecture of edge cloud computing and its relationship with end-users is shown in Fig. 1.

Although mobile edge computing has many advantages, it also faces challenges due to its emerging nature. Load balancing is considered a key challenge in this field. Effective load balancing is crucial for optimizing edge computing systems. Its primary objective is assigning tasks to available resources to enhance overall response time and ensure efficient resource utilization. With efficient load-balancing methods, users can avoid issues related to quality of service, service level agreement violations, unreliable data processing, and unresponsiveness [8, 9]. Consequently, developing efficient energy and load-balancing techniques and strategies is essential for the success of edge computing systems [10]. Recent studies [11–14] have applied reinforcement learning techniques to mobile edge computing to address computing offloading, task scheduling, workflow scheduling and load balancing, aiming to reduce energy consumption and improve system utilization. However, other studies [15–18], such as those utilizing deep reinforcement learning in heterogeneous IoT networks, have shown that neural network models can lead to increased task completion due to prolonged training. This delay can negatively impact network efficiency, responsiveness, and energy consumption. While these approaches have improved, learning automata [19] to reduce average task completion time remains relatively unexplored. Learning automata, as a reinforcement learning method, are highly adaptable to changes in their environment, which makes them an excellent fit for dynamic edge and cloud environments. By communicating and making local decisions, they can collectively reach the best possible outcome for the entire system. This approach is beneficial in edge and cloud environments, where centralized algorithms can struggle with issues related to data accumulation and dissemination [20, 21]. Additionally, learning automata have minimal computational requirements and incur low communication costs when interacting with the environment [21, 22]. Furthermore, they can accurately model the topology of edge and cloud environments and simulate the evolving behaviour of nodes, considering their capacity to learn and adapt. The iterative process through which learning automata gather information for decision-making ensures that potential errors do not significantly impact their performance, setting them apart from other algorithms [23]. Traditional load-balancing methods often rely on static or heuristic approaches, which may not be optimal in dynamic edge computing environments. These methods struggle to adapt to changes in network conditions and workload patterns, leading to suboptimal performance. Reinforcement learning-based methods, while promising, often require significant training time and computational resources [24]. Additionally, balancing exploration and exploitation can be challenging, and complex models may be difficult to implement in resource-constrained edge devices. To address these limitations, we propose a novel load-balancing algorithm that leverages learning automata (LA) to adjust task distribution between edge and cloud environments dynamically. Unlike traditional methods, LA updates in real-time based on feedback (STM), making it highly suitable for dynamic, real-time IoT environments where task demands change frequently. The innovation of our research lies in developing an adaptive load-balancing algorithm that utilizes Learning Automata (LA) to minimize time delays in allocating real-time tasks between edge and cloud servers, thereby enhancing average service time. Our novel approach for joint cloud-edge data centres maximizes the allocation of tasks with low latency requirements while surpassing conventional methods in performance. Unlike traditional centralized algorithms that struggle with data accumulation and dissemination challenges, LA's inherent adaptability to environmental changes makes it particularly suitable for dynamic edge-cloud environments. The system's effectiveness stems from its ability to make local decisions through real-time updates based on Short-Term Memory (STM) feedback, allowing LA to collectively achieve optimal outcomes for the entire system while maintaining minimal computational requirements and low communication costs. This combination of features makes our solution highly effective for dynamic, real-time IoT environments where task demands change frequently, and resource efficiency is crucial. Our research makes several significant contributions to the field of edge computing. First, we propose a novel load-balancing algorithm that leverages learning automata (LA) to adjust task distribution between edge and cloud servers dynamically. This approach enables real-time adaptation to changing network conditions and workload patterns, improving task completion times and resource utilization. Second, we introduce a novel server selection mechanism that considers task complexity, server capacity, and network latency to optimize task allocation. This mechanism enhances load-balancing strategies in dynamic distributed computational environments. Finally, we propose a comprehensive algorithm for load balancing in joint cloud-edge data centres that maximizes the allocation of latency-sensitive tasks while outperforming traditional methods. By leveraging the adaptive nature of LA and real-time feedback, our algorithm significantly reduces average service time and optimizes network traffic.

The rest of this paper is organized as follows: Sect. 2 summarizes related research. Section 3 explains learning automata, a type of reinforcement learning. Section 4 introduces the proposed algorithm. Section 5 describes the simulation methodology, and Sect. 6 concludes the paper.

2 Related work

Recently, a surge of research has focused on improving load balancing in collaborative cloud-edge computing. These efforts aim to reduce latency, conserve energy, and achieve an optimal trade-off between latency and energy consumption in task-offloading scenarios. This section will explore different studies focusing on making task allocation and offloading more efficient in cloud-edge computing systems. Table 1 provides a summary of these studies.

Dong et al. [25] introduced a method that optimizes task distribution among devices and minimizes energy consumption while offloading tasks to the cloud. Their approach incorporates strategies like Component Tagging and Merging to reduce complexity, an Offloading Valuable Basic (OVB) constraint to enhance performance and a reliable system design based on the hierarchical structure between fog and edge servers. Wang et al. [26] proposed a delay-focused architecture to improve performance in diverse fog-based systems. By utilizing dynamic queues and risk theory for reliable communication, this approach offers enhanced resource utilization and reduced total delay despite the inherent complexity of optimization problems. Lim et al. [27] presented a load-balancing algorithm for mobile devices in edge cloud computing, addressing workload distribution and edge server selection while accommodating device mobility. The primary advantage of this approach lies in its ability to distribute tasks to nearby edge servers efficiently. However, scalability and complexity remain challenges that need to be addressed.

Hosseini et al. [28] proposed a QoS-aware and cost-efficient task-scheduling method for fog-cloud resources in volunteer computing systems. This approach demonstrated superior performance regarding deadline-satisfied tasks and reduced violation costs. While achieving up to 95% PDST and 99.5% violation cost reduction, future improvements could focus on energy optimization and evaluation with diverse real-world datasets. However, the solution's applicability is limited to volunteer computing systems. Liu et al. [29] introduced a near-optimal approach for online task offloading and resource allocation in edge-cloud orchestrated computing systems. By reformulating the problem and developing an algorithm for resource allocation and task offloading, the approach reduces completion time and energy

Table 1 Related work

References	Contribution	Algorithm
[25]	Reliability-aware offloading and allocation	N-fold integer programming based
[26]	Dynamic computation offloading with reliability	Greedy heuristic
[27]	Efficient Load balancing	Genetic Algorithm
[28]	Efficient heuristic scheduling algorithms	Heuristic Algorithm
[29]	Fairness in resource allocation	Online semi-decentralized algorithm
[30]	Edge computing total utility maximization	Heuristic algorithm
[31]	Deep recurrent reinforcement learning	Deep Q-network
[32]	Efficient task offloading approach	Evolutionary algorithms
[33]	Collaborative cloud-edge-end offloading	Deep reinforcement learning
[34]	Parallel scheduling of large-scale tasks	Evolutionary algorithm
[35]	Reinforcement-learning-based task assignment strategy	Q-learning
[36]	Learning Automata-based Resource Allocation	Learning automata
[37]	Q-learning-based load balancing	Q-learning
[38]	Fast and efficient task offloading	PBTO algorithm
[39]	Double Deep Q-Network (DDQN) for optimal offloading in dynamic edge-cloud environments	A deep reinforcement learning algorithm for optimal decision-making in dynamic environments
[40]	Machine Learning-based Load Balancing	Real-time Load Balancing with Execution Time Prediction
[41]	Intelligent Bilateral Client Selection in Federated Learning with Newcomer IoT Devices	Game theory, bootstrapping, and machine learning
[42]	Optimized node selection for UAV-assisted FL, enhancing training efficiency and accuracy by accounting for convergence gaps and dynamic network issues	Node selection algorithm with convergence gap optimization

consumption and ensures high performance. Simulation results demonstrate its superiority over other methods across various metrics. However, empirical validation is necessary to solidify the proposed approach further. Li et al. [30] aimed to enhance user satisfaction for time-sensitive IoT applications in edge computing by maximizing average total utility. The introduced algorithms and admission control policy address critical challenges and demonstrate promising evaluation performance. However, the approach overlooks security and privacy concerns, involves high computational complexity, and requires consideration of long-term data patterns.

Dai et al. [31] introduced a deep reinforcement learning approach for task offloading in vehicular edge computing within a cloud-edge environment. Long et al. [32] proposed TO-EEC, an algorithm for task offloading in the End-Edge-Cloud architecture. By addressing edge node delay, energy consumption, and load balancing through multi-objective optimization, TO-EEC employs an improved AR-MOEA method to achieve optimal solutions. Key advantages include avoiding local optima and faster convergence to the global optimum. Key contributions involve modelling the multi-objective problem, enhancing AR-MOEA components, and emphasizing cloud-edge collaboration. Tang et al. [33] focused on a deep reinforcement learning method for IoT device task offloading in a collaborative cloud-edge-end setup. Addressing task dependencies, the approach solves a multi-objective optimization problem, prioritizing energy consumption reduction and low latency. The method efficiently allocates computing resources while considering edge servers' communication and computation capabilities. However, it requires extensive training data and can be computationally expensive due to deep reinforcement learning model training.

Laili et al. [34] proposed a parallel scheduling solution for large-scale tasks in industrial cloud-edge collaboration. By utilizing a hybrid evolutionary algorithm and a merge strategy, their approach minimizes energy consumption and delay. Siyadatzadeh et al. [35] presented ReLIEF, a real-time task assignment strategy using reinforcement learning for fault-tolerant fog computing. This approach outperforms other reliability and workload balancing methods. Ebrahim Pourian [36] suggested a learning automata-based approach to address the resource allocation problem in fog computing. Considering service time, cost, and energy consumption, the algorithm excels in makespan and average response time compared to other methods. However, the proposed solution demands additional processing power and time for optimal solution computation, which may pose challenges in resource-constrained environments.

Du et al. [37] proposed a load-balancing technique based on Q-learning for processing real-time tasks in edge-cloud networks. The method addresses issues such as uneven load distribution, network congestion, and limited processing capability of edge servers. By employing reinforcement learning with the Q-learning algorithm to distribute the workload between edge and cloud servers, the simulation results indicate that this approach is more effective in reducing processing time than traditional static load balancing techniques. Liu et al. [38] introduced a three-layer architecture for efficient task offloading in Edge-Cloud environments centred around the Edge-Cloud Broker (ECB). The ECB manages task requests, monitors resources, and employs offloading algorithms to improve task offloading, reduce energy usage, and enhance IoT application performance. However, implementing the proposed architecture may require infrastructure changes and careful placement of the ECB.

Ullah et al. [39] proposed a Double Deep Q-Network (DDQN)-based framework to optimize task offloading and resource allocation in dynamic edge-cloud environments. The proposed framework aims to minimize task completion time, energy consumption, and network congestion by learning optimal offloading decisions through interaction with the dynamic edge-cloud environment. Rahmani et al. [40] proposed a novel machine learning-based load-balancing technique for real-time heterogeneous systems. This approach aims to improve system performance and energy efficiency by predicting application execution times and assigning tasks to underloaded devices dynamically. Wehbi et al. [41] propose FedMint, a novel client selection approach for federated learning that addresses the challenges of random selection and newcomer device integration. FedMint employs a game-theoretic approach to match IoT devices with federated servers based on their preferences. Additionally, a bootstrapping mechanism assigns initial accuracy values to newcomer devices, enabling their effective participation in the FL process.

In the paper by Bai and Chen [42], the authors propose a robust node selection strategy for federated learning (FL) tailored to UAV-assisted edge computing environments, addressing challenges like limited data quality and network dynamics. The study introduces a three-layer FL architecture designed to optimize training by strategically selecting unmanned terminals (UTs), UAVs, datasets, and delays to reduce convergence gaps in model training. The authors validate their approach through simulations, showing significant improvements in training efficiency and accuracy under varying wireless conditions, making the framework highly effective for federated learning in mobile, resource-constrained settings.

The studies provide a comprehensive overview of the state-of-the-art techniques for load balancing in cloud-edge computing. While these approaches offer valuable insights, a significant opportunity remains to optimize task allocation and resource utilization further. Our proposed approach addresses these challenges by leveraging a novel learning automata-based algorithm to achieve efficient and adaptive load balancing in dynamic edge-cloud environments.

3 Preliminaries

This section provides a concise introduction to learning automata as the foundation for the rest of the paper.

3.1 Learning automata theory

A learning automaton [16] is designed to make adaptive decisions by learning from interactions with a random environment. It selects actions from a limited set, with each action carrying a probability of being rewarded by the environment. The automaton consistently learns to choose the optimal action through an iterative process. The adaptation mechanism begins with initializing each server with a uniform probability distribution, making each equally likely to be selected for task allocation. Tasks are then assigned to servers based on this probability distribution. If a server completes a task within the Service Time Measurement (STM) threshold, its probability of being chosen increases, reinforcing the selection of high-performing servers. Conversely, servers failing to meet the STM have their selection probabilities decreased, penalizing underperformance. This probability update process allows the learning automaton to adapt continuously to changing conditions, dynamically selecting the most suitable server for each task. Adjusting probabilities in response to feedback enables the adaptive learning-based load balancing algorithm (ALBLA) to efficiently manage workload variations, network conditions, and server availability, enhancing performance and energy efficiency in edge-cloud environments.

The environment [19] can be represented by a triple $E\{\alpha, \beta, c\}$, where $\alpha\{\alpha1, \alpha2, \ldots, \alpha r\}$ indicates the finite set of inputs,$\beta\{\beta1, \beta2, \ldots, \beta m\}$ is the set of the values that the reinforcement signal can take, and $c\{c1, c2, \ldots, cr\}$ denotes the set of the penalty probabilities. Based on the type of reinforcement signal b, environments can be categorized as P-model, Q-model, and S-model. In P-model environments, the reinforcement signals can only take binary values, 0 and 1. Q-model environments allow a finite number of values between 0 and 1 for the reinforcement signal. In S-model environments, the reinforcement signal falls within the range $[a, b]$. Figure 2 illustrates the interaction between the learning automaton and the random environment.

Learning automata can be divided into two primary categories: fixed-structure learning and variable-structure learning. Variable structure learning automata are denoted by a triple $\langle \beta, \alpha, L \rangle$, where β represents the set of inputs, α represents the set of actions, and L represents the learning algorithm. The learning algorithm is a recursive formula that adjusts the action probability vector. Let $\alpha i(k)$ and $p(k)$ represent the action chosen by the learning automaton and the probability vector defined. Over the action set at time k, respectively. Parameters $aandb$ determine the degree of increase and decrease in action probabilities. The variable r represents the number of actions the learning automaton can take. At each time k, the action probability vector $p(k)$ is updated using the linear learning algorithm described in (1) if the random environment rewards the chosen action αi(k), and it is updated as described in (2) if the action is penalized [19].

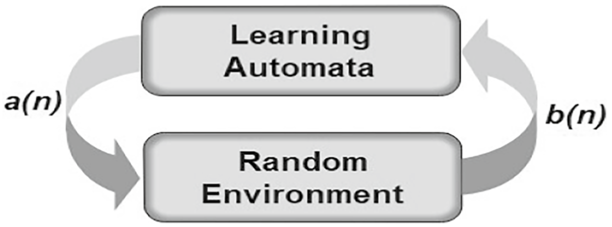

Fig. 2 The connection between the learning automata and its random environment

$$P_j(n+1) = \begin{cases} P_j(n) + a\left[1 - P_j(n)\right] & j = i \\ (1-a)P_j(n) & \forall j \neq ji \end{cases} \tag{1}$$

$$P_j(n+1) = \begin{cases} (1-b)P_j(n) & j = i \\ \left(\frac{b}{r-1}\right) + (1-b)P_j(n) & \forall j \neq ji \end{cases} \tag{2}$$

When $\alpha = b$, Eqs. (1) and (2) are referred to as the linear reward penalty (LRP) algorithm; if α is significantly greater than b, these equations are known as the linear reward-ε penalty (LRεP) algorithm. Finally, when b equals 0, they are referred to as the linear reward-inaction (LRI) algorithm. In the latter scenario, the action probability vectors remain unaltered when the environment penalizes the action.

4 Proposed methodology

We propose using reinforcement learning to decrease the average time to complete tasks and transmit data between edge and cloud servers, commonly called service time. Service time encompasses the duration required for processing tasks and communicating between edge servers and cloud servers. The Service Time Measurement parameter (STM) is based on the predicted time for processing each task and resource requirements. Following the assignment of each task to a server, the expected time and resource requirements are evaluated. The optimal condition for two tasks, i and j, is as follows [36]:

$$if\left(Predictedtime_{Ti} \leq Predictedtime_{Tj}\right)\&\&\left(Request_{Ti} \leq Request_{Tj}\right), \forall Ti \in T, i \neq j \tag{3}$$

This paper utilizes the S environment model, an LRP learning automata. Initially, we will outline the data structure of the proposed algorithm, followed by an explanation of the load-balancing algorithm. As described in Sect. 3, the probabilistic values adjusted based on environmental feedback are highly significant for decision-making within learning automata.

4.1 Structure design

To incorporate learning automata into our proposed algorithm, we introduce two key concepts: the current server (CS) and the available servers (AS). The CS refers to the server currently processing a task, while the AS represents a list of potential servers for task processing.

$$S = \left\{S1, S2, \ldots \,|Si\ is\ the\ i^{th} - neighbour\ of\ CS,\ i = 1, 2, \ldots\right\} \tag{4}$$

Here, S represents the set of neighbouring servers of the current server (CS), and Si denotes the i-th neighbour.

$$Pvector = \{P1i, P2i, \ldots, Pri | r \text{ is the number of actions for } Si, Pki = 1\} \quad (5)$$

Each server employs a learning automata algorithm that maintains a list of available servers (AS). This list contains the ID of each server and an associated probability vector (*Pvector*). The *Pvector* consists of two probabilities:

P_{1i}: The probability of offloading tasks to an edge server, and P_{2i}: The likelihood of offloading tasks to the cloud. Initially, both probability vectors are set to 0.5 at the commencement of the algorithm execution. These probabilities are dynamically updated based on the server's past performance and the feedback received from the environment. As illustrated in Fig. 3, the table structure represents the state of a specific server (CS) and its view of other available servers (AS). Each row in the table corresponds to an available server (AS_i) and associated *Pvector*.

In this example, AS_1 is the first available server, P_{11} is the probability of AS_1 downvoting tasks to an edge server, and P_{21} is the probability of AS_1 downvoting tasks to the cloud. The *Pvector* for each available server is updated dynamically based on the learning automata algorithm, allowing the system to adapt to changing conditions and optimize task allocation (Table 2).

4.2 Learning phase

Three types of stored tables are utilized for training the servers: the proximity-hit- LA table contains servers that have successfully processed tasks on edge and adhered to STM conditions based on their previous processing history. The second table, Proximity- miss-LA, stores servers with processed functions at the edge but still need to meet STM conditions according to their last processing history. The storage format of these tables is identical to the Proximity-hit-LA table. The third category consists of servers that have forwarded tasks to the cloud for processing in previous processing stages, which are stored in a table named Processing-LA. It is evident that in this scenario, no servers will be accessible in this table. Figure 4 depicts the process of analyzing tables.

Initially, a task is dispatched to a designated server for processing. If the current server does not meet the STM condition, the processing continues by examining the

Fig. 3 A simple format of CSs table for AS1

	AS1
S_1	$\{P_{11}, P_{21}\}$
S_2	$\{P_{12}, P_{22}\}$
S_3	$\{P_{13}, P_{23}\}$
	$\vdots$
S_r	$\{P_{1r}, P_{2rs}\}$

Table 2 Simulation parameters

Parameters	Descriptions
System configuration	Topology type: fully connected Memory: 8.00 GB OA: Windows 10 Enterprise
Simulator/language	EdgeCloudSim/Python
Measurement testing	Average service time Delay Network traffic Number of successful task execution on edge
Scalability Testing	Increase in the number of devices from 100 to 1000
Type of tasks	Mixed reality Remote environment monitoring Weather forecasting application
Configuration	Random task generation according to the kinds of applications

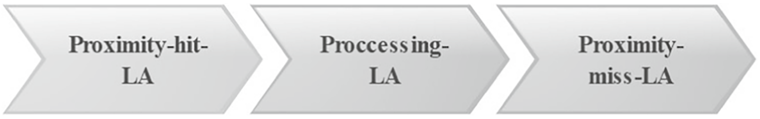

Fig. 4 The sequence of tables examination

stored tables. With each task submission, a new row is appended to the table of the respective current server. First, the proximity-hit-LA table is examined to verify if its tasks meet the STM condition. If the condition is satisfied, the servers with the highest probability of offloading tasks to the edge (according to their Pvector) are selected. These servers then process the task. The algorithm turns to the Processing-LA table if the STM condition is not met in the proximity-hit-LA table. From the available servers (AS), all servers with the highest probability value from the Processing-LA table are selected for further processing. The selected servers then process the designated number of tasks. Once these servers are chosen, the specified tasks are forwarded for processing. In this scenario, the probability values within each server's Pvector are determined based on their past performance, reflected by rewards and penalties received during previous processing cycles.

If the processing fails to meet the desired STM after analyzing the proximity-hit-LA and Processing-LA tables, the algorithm resorts to the proximity-miss-LA table. This table stores servers that have previously processed tasks at the edge but violated the STM condition. From this table, servers with the highest probability value for offloading tasks to the edge are selected for further processing. If processing violates the STM conditions, the results are sent back through the path taken for server selection. All servers along this path adjust their Pvector based on their performance and the learning automata algorithm. The processing phase concludes when all tasks are processed or the desired STM is not achieved for any remaining tasks. Figure 5 illustrates the workflow of the proposed method.

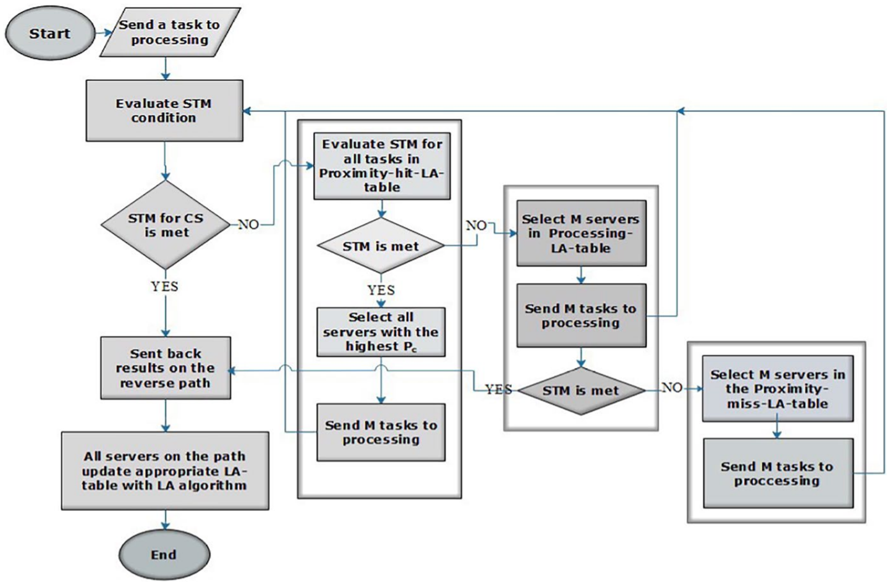
Start
Send a task to processing
Evaluate STM condition
STM for CS is met
NO
YES
Sent back results on the reverse path
All servers on the path update appropriate LA-table with LA algorithm
End
Evaluate STM for all tasks in Proximity-hit-LA-table
STM is met
YES
Select all servers with the highest P_c
Send M tasks to processing
NO
Select M servers in Processing-LA-table
Send M tasks to processing
STM is met
YES
NO
Select M servers in the Proximity-miss-LA-table
Send M tasks to proccessing

Fig. 5 Proposed load balancing algorithm flowchart

4.3 Reward and penalty

In our proposed model, each server is managed by a learning automaton initialized to allocate tasks evenly between edge and cloud processing. This initialization ensures a balanced likelihood for tasks being processed at either level, represented by the initial probability assignment, where the current server is penalized to minimize the probability of selecting unproductive servers. P1(i) = P2(i) = 0.5. This setup implies an equal likelihood of any server performing task processing, allowing the system to adapt based on performance feedback. As shown in pseudocode, the update mechanism hinges on the continuous evaluation of each server's Service Time Measurement (STM). After each task execution, the server's STM is assessed to determine if the task met a predefined performance threshold. The update process then adjusts the probability vector of the corresponding server based on this feedback:

- *Reward*: If a server meets the STM condition (indicating efficient task completion), its probability in the vector is updated according to Eq. (1), increasing the likelihood of its future selection. This mechanism rewards high-performing servers by amplifying their probabilities in the task allocation process.
- *Penalty*: Conversely, if a server fails to meet the STM condition, Eq. (2) reduces its probability, lowering its likelihood of being selected for future tasks. This penalty discourages the selection of underperforming servers, making the system more adaptive.

The adaptation mechanism in our learning automata (LA) dynamically refines each server's probability allocation through continuous feedback integration. This adaptive updating approach allows the LA to adjust effectively to changes in workload and server performance over time, ultimately steering task assignments towards consistently productive servers. This ongoing feedback and probability adjustment ensures that the algorithm remains robust, even under varying workloads and network conditions, thereby maximizing performance and efficiency in edge-cloud systems.

This structured initialization, updating, and adaptation approach reinforces the system's ability to prioritize reliable servers, enhancing performance stability across dynamic network conditions and workloads.

Pseudocode of P_{vector} Update

Algorithm: Pvector Update

Input: Current Server (CS), Available Servers (S), Feedbacks (F)

Output: Updated Pvector

```
1. for each server s in S, do
2.    if CS sent task to s then
3.        if F[s] == "HIT" then
4.            Update Pvector[s] using equation (1) // Reward
5.        else
6.            Update Pvector[s] using equation (2) // Penalty
7.        end if
8.    end if
9. end for
```

4.4 Time complexity

The time complexity of the proposed algorithm is influenced by several factors, primarily the number of servers (n) and the number of tasks (m). Each server requires an initialization step to set its initial probability vector (P), resulting in an O(n) time complexity. For each task, the algorithm updates the probability vectors of all servers based on feedback, which takes O(n) time. Additionally, the algorithm selects the optimal server based on the highest STM score, which involves sorting or finding the maximum value in a list of size n, resulting in a time complexity of O (n log n). Considering both initialization and task assignment, the overall time complexity of the algorithm can be approximated as O (m * n log n). This indicates that the algorithm scales linearly with the number of tasks and logarithmically with the number of servers.

5 Simulation

In this section, we outline the simulation environment and present the findings of the experimental results of our proposed algorithm. Section 5.1 details the simulation parameters and the evaluation approach, while Sect. 5.2 discusses the experimental results and their analysis.

5.1 Simulation setup

To assess the proposed algorithms' efficacy compared to other state-of-the-art techniques, we leveraged EdgeCloudSim [43], a specialized tool based on CloudSim [44] tailored for edge computing environments. This tool is designed for edge computing environments, enabling comprehensive experimentation considering computational and networking resources. By utilizing EdgeCloudSim, we could thoroughly evaluate the proposed method's performance across diverse edge computing scenarios, ensuring a holistic analysis of its effectiveness in real-world conditions, including the impact of computing and networking resources.

Various metrics have been evaluated, including average service time, delay, network traffic, and number of successful task execution on the edge across different scales. We examined three distinct task types related to Mixed Reality (MR) [45], remote environment monitoring [46], and weather forecasting applications [47]. These applications have unique requirements; for example, MR requires low latency for seamless user experience [45]. However, remote environment monitoring demands timely processing and communication of results to enable prompt responses to changing environmental conditions [46]. Weather forecasting involves analyzing and processing vast amounts of data from various sources such as satellites, weather stations, and climate models, thus requiring significant computational resources [47]. The algorithms' policies play an essential role in deciding how tasks from mobile devices are distributed within a hotspot. To simulate this scenario, we configured a mobile device to generate random functions with three to eight items simultaneously, assuming sufficient edge servers in edge cloud environments based on specific application types.

Leveraging sufficient specialized edge servers in the edge cloud is crucial for optimizing task execution, enhancing performance, and effectively managing resources. The required number of edge servers depends on the specific workload and the desired level of parallelism for task completion.

5.2 Experimental evaluation

We evaluated the performance of our proposed algorithm by comparing it to four other methods: Cloud First (CF), Probability, Graph Coloring-based Genetic Algorithm (GG), Round Robin (RR), and An Energy-efficient VM Allocation Algorithm for IoT Applications in a Cloud Data Center (AFED-EF). Cloud First prioritizes the central cloud, connecting to it when accessible and resorting to edge servers only when the cloud is busy [26]. The Probability method assigns tasks between edge and cloud servers using a fixed probability of 50%. The Graph Coloring-based Genetic Algorithm (GG) employs graph colouring and a genetic algorithm to efficiently distribute offloaded tasks to nearby edge servers [26]. An alternative approach is Round Robin, where mobile devices sequentially select edge servers for task offloading.

5.2.1 Average service time

This section evaluates the average service time for five algorithms when the number of devices is 1000. As previously mentioned, average service time explains the average time it takes for a task to be processed and completed by a server. This includes the time it takes for the task to be assigned to a server and the time it takes for the server to execute the task. In other words, the service time is the overall duration needed to accomplish tasks and facilitate communication between edge and cloud servers [48]. Reducing average service time is essential for improving the network's overall performance and user experience, as shorter service times lead to faster response times and higher throughput. Figure 6 compares the average time required to complete a task or service for our proposed algorithm, the Cloud First method, and the GG approach. Our proposed algorithm exhibits the shortest average time of slightly more than 0.8 s, primarily due to its adaptive behaviour in selecting the optimal server for task processing. The Cloud First method, relying solely on the central cloud server for task offloading, incurs the longest average service time at around 1.4 s. The proposed algorithm employs a learning-based approach to determine the best server for task processing, whether on the edge or in the cloud. As the number of devices increases, the algorithm's ability to make informed decisions improves through iterative learning. In contrast, the GG approach uniformly allocates tasks to adjacent edge servers, limited by the maximum number of colours. This results in an average service time of approximately 0.9 s. Our proposed algorithm outperforms Cloud First, GG and AFED-EF approaches regarding average service time,

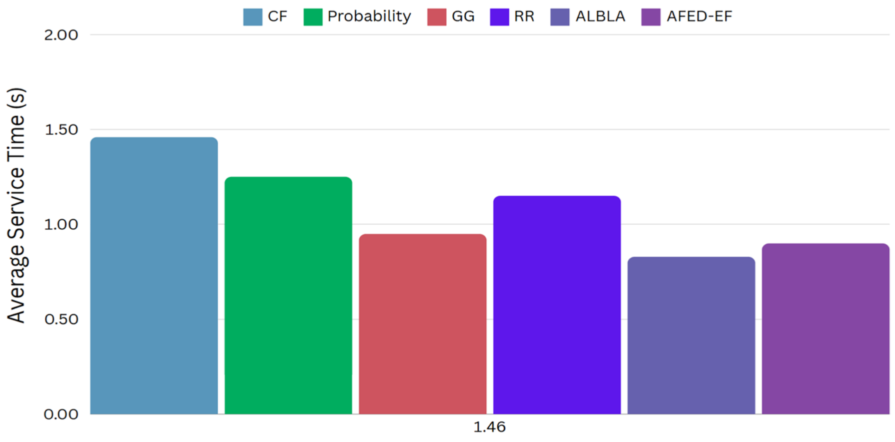

Fig. 6 The average service time for different algorithms with a scale of 1000 devices

demonstrating its adaptive and efficient behaviour in selecting optimal servers for task processing.

5.2.2 Delay task processing

The time a task spends waiting in a queue before it is assigned to a server for processing is referred to as its waiting time in the context of task management [49]. This can occur when more tasks are submitted to the network than the available servers can process. We evaluate the different algorithms across various scales to analyze task waiting times and compare their performance. As the number of devices increases, the delay increases, as shown in Fig. 7.

The Cloud First method exhibits a significantly long waiting time of more than 2.5 s due to the high number of tasks simultaneously accessing the central cloud server. In contrast, the GG method has a relatively higher waiting time than the probability and the RR methods. This is because the GG method considers both the latency for edge servers and the duration needed to transfer tasks from edge servers to the central cloud server during offloading. In contrast, other methods only consider waiting times for edge servers. Our proposed algorithm demonstrates superior performance in terms of task waiting time, especially for larger scales with a high number of devices. This is attributed to its ability to efficiently distribute tasks across edge and cloud servers, minimizing congestion and reducing waiting times. The experimental results demonstrate that both ALBLA and AFED-EF outperform other methods, showcasing their superior performance and scalability. However, our ALBLA algorithm exhibits slightly better performance, especially in high device counts scenarios, indicating its potential for even more efficient task execution in large-scale edge computing systems.

5.2.3 Network traffic

Network traffic, encompassing the volume and flow of data transmitted between devices and servers, is critical in edge cloud networks. High traffic levels can lead

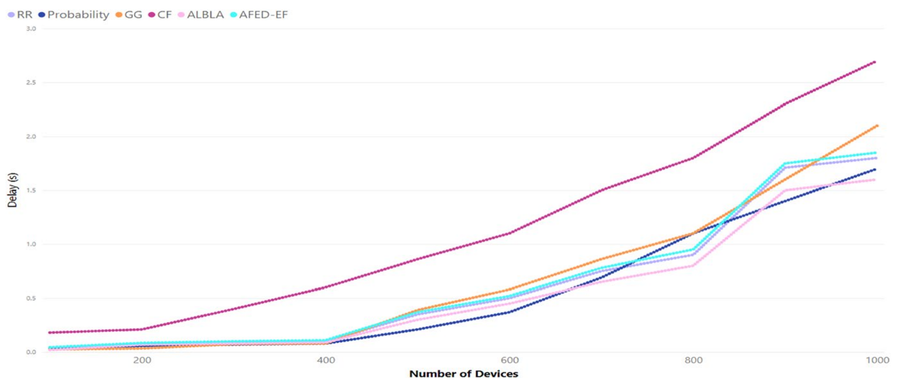

Fig. 7 There is a delay with the different scales of devices

to congestion, delays, and reduced throughput. This can include data generated by IoT devices and transmitted by other connected devices such as smartphones, laptops, and tablets. Managing network traffic is critical for ensuring the reliability and performance of the network, as high traffic levels can lead to congestion, delays, and reduced throughput [50]. Load-balancing algorithms aim to optimize network traffic by efficiently distributing tasks across available servers. The Graph Coloring-based Genetic Algorithm (GG) method is the most effective in reducing network traffic, as demonstrated in Fig. 8. This is primarily attributed to its exceptional load-balancing capabilities, which significantly minimize the risk of network failures.

Our proposed algorithm performs similarly to the GG method, reducing overheads and unnecessary data transmission between servers. This is achieved through intelligent network traffic management and optimized resource utilization. The proposed algorithm closely matches the GG method in terms of network efficiency, as it minimizes the amount of data that needs to be transmitted over the network, thereby reducing network congestion and improving overall system performance. On the other hand, the round-robin (RR) method experiences less network traffic than the cloud-first method. This is because the RR method experiences a higher rate of task failures primarily because there are no guarantees regarding the delay time. As a result, tasks are reassigned to other servers, which reduces the overall amount of data that needs to be transmitted over the network.

In contrast, implementing Cloud First substantially increases network traffic due to its heavy dependence on wide area networks (WANs). WANs are prone to failures and necessitate resending data and application files because of their vulnerability to failures. This increases traffic as data is repeatedly transmitted over the network, leading to congestion and reduced system performance. Our proposed algorithm, along with the AFED-EF method, offers a more efficient and reliable solution for managing network traffic in edge cloud networks and exhibits better performance in network traffic reduction, especially in scenarios with many devices.

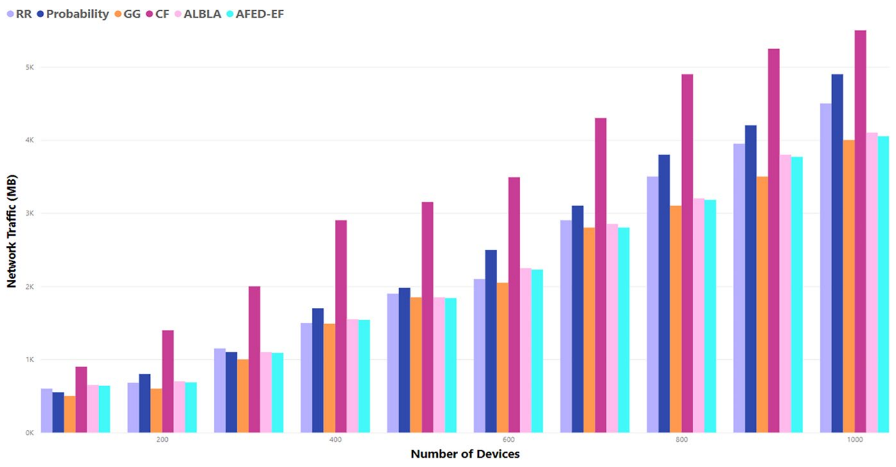

Fig. 8 The network traffic when the number of devices is different

5.2.4 Number of successful task executions on edge

Figure 9 compares our proposed algorithm with four other algorithms based on the number of successful task executions on edge servers. The Cloud First approach, which does not utilize edge servers and relies solely on the central cloud server for task offloading, had the lowest success rate of approximately 33% due to its implementation despite having the highest number of task executions in the cloud. This is because the Cloud First approach results in heightened network traffic for task offloading, which can lead to congestion and decreased success rates. In contrast, our proposed algorithm effectively learns to forward tasks to optimal edge servers as the number of devices scales up. Consequently, our proposed method outperforms the GG method, achieving a success rate of 84% when the number of devices is 600. This improvement is attributed to the algorithm's ability to efficiently distribute tasks across edge servers, resulting in higher success rates [51].

Moreover, as the learning process is repeated, the system becomes better equipped to comprehend various scenarios, resulting in proficient load distribution. This is evident in Fig. 10, as our proposed algorithm's success rate continues to improve as the number of devices increases. Our proposed algorithm performs superior to other algorithms regarding successful task executions on edge servers.

In addition to achieving a high success rate for task executions, our proposed algorithm is inherently robust against node failures. When a node failure occurs, the algorithm's decentralized structure allows it to dynamically reassign tasks to other available nodes, avoiding single points of failure. This ensures that the success rate remains stable despite unexpected node issues, as tasks are seamlessly rerouted to maintain continuous processing. As a result, our method maintains high performance and reliability across varied network conditions, further enhancing its suitability for dynamic edge-cloud environments.

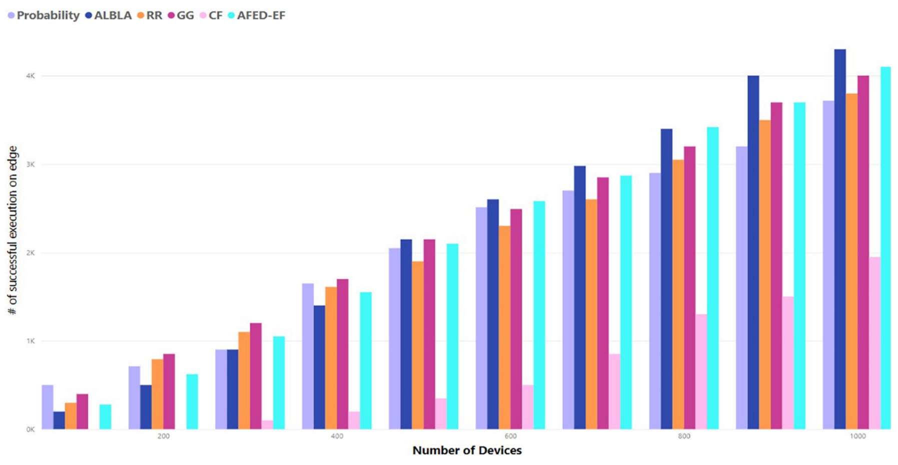

Fig. 9 The number of successful executions with different scales of devices

5.2.5 Sensitivity analysis

To evaluate the robustness of our proposed algorithm, we conducted a sensitivity analysis using varying key parameters, including the number of devices/tasks and edge server capacity. Table 3 summarizes the results under four scenarios: Low Traffic, Medium Traffic, High Traffic, and High Traffic with Increased Capacity. Each scenario illustrates the algorithm's adaptability to varying network load levels and available resources. Impact of Device/Task Load: As shown in Table 3, an increase in the number of devices from 100 to 1000 leads to an increase in average service time and network traffic. For instance, under the high-traffic scenario, service time rises to 1.4 s with a success rate of 75%. This demonstrates the expected challenges with higher traffic loads; however, our algorithm maintains performance by dynamically redistributing tasks, ensuring stable throughput even with increased network congestion. Effect of Edge Server Capacity: When edge server capacity is increased to 20 GB in the High Traffic scenario, the service time is reduced from 1.4 to 1.0 s, while the success rate improves to 90%. This suggests that enhancing edge server capacity significantly mitigates delays and improves task execution success rates. Our algorithm efficiently utilizes the increased capacity, highlighting its scalability and suitability for high-load environments.

These results underscore the algorithm's adaptability to varying conditions, maintaining optimal service times and success rates with appropriate resource allocation adjustments. Consequently, the proposed method proves resilient to shifts in workload intensity and resource availability, reinforcing its utility in dynamic edge-cloud networks.

5.3 Comparative analysis

This section provides a comparative analysis of our proposed ALBLA algorithm against several state-of-the-art techniques, including Round Robin (RR), Cloud First (CF), Probability-based, Graph Coloring-based Genetic Algorithm (GG), and recent approaches by Zhou et al. [47, 48]. These algorithms are evaluated based on key performance metrics such as energy efficiency, adaptability, scalability, and task handling, particularly in edge-cloud environments.

As shown in Table 4, ALBLA outperforms existing energy efficiency, adaptability, scalability, and task-handling approaches, especially in dynamic edge-cloud environments. By leveraging learning automata, ALBLA can effectively adapt to

Table 3 Sensitivity analysis results

Scenario	Tasks	Capacity (GB)	Service time (s)	Network traffic	Success rate (%)
Low traffic scenario	100	10	0.8	Low	95
Medium traffic scenario	500	10	1.1	Moderate	85
High traffic scenario	1000	10	1.4)	High	75
High traffic with increased capacity	1000	20	1.0	Moderate	90

Table 4 Comparative analysis of load balancing algorithms for edge-cloud systems

Algorithm	Energy efficiency	Adaptability	Scalability	Task handling	Edge-cloud compatibility
Round Robin (RR)	May not optimize energy efficiency due to static scheduling	Less adaptive to dynamic workloads and resource availability	Scalable for cloud-only environments but may not be efficient for edge-cloud scenarios	Suitable for static tasks, may not handle dynamic IoT workloads efficiently	Less ideal for edge-cloud environments due to lack of dynamic adaptation
Cloud First (CF)	Prioritizes cloud resources and may lead to suboptimal energy efficiency, especially in edge-heavy scenarios	Less adaptive to dynamic workloads and resource availability	Scalable for cloud-only environments but may need to be more efficient for edge-cloud scenarios	It is suitable for static tasks and may need to handle dynamic IoT workloads more efficiently	It is less suitable for edge-cloud environments due to its cloud-centric approach
Probability	Random task allocation may lead to unbalanced resource utilization and suboptimal energy efficiency	Less adaptive to changes in workload and resource availability	Scalable for cloud-only environments but may need to be more efficient for edge-cloud scenarios	It is suitable for static tasks and may need to handle dynamic IoT workloads more efficiently	Less suitable for edge-cloud environments due to its random nature
Graph Coloring-based Genetic Algorithm (GG)	Efficient resource allocation using graph colouring techniques	Less adaptive to dynamic changes in network topology and workload	Scalable for static cloud environments but may struggle in dynamic edge-cloud scenarios	It is suitable for static tasks and may need to handle dynamic IoT workloads more efficiently	Less suitable for edge-cloud environments due to its static nature
AFED-EF	Optimizes VM allocation and placement for energy efficiency	Focused on static cloud environments, it needs more adaptability to dynamic edge-cloud scenarios	Scalable for large-scale cloud data centres but may struggle with edge-cloud scenarios	They are primarily designed for static tasks and less suitable for dynamic IoT workloads	It is less suitable for edge-cloud environments because it focuses on static cloud scenarios
LLD	Proposes adaptive energy-aware algorithms for VM allocation and deployment	Adapts to changes in cloud environments but may not fully leverage edge computing capabilities	Scalable for cloud-only architectures but may face challenges in dynamic IoT scenarios	Suitable for long-term tasks but may need to be optimized for real-time or dynamic IoT applications	It is partially suitable for edge-cloud environments but may not fully leverage the benefits of edge computing

Table 4 (continued)

Algorithm	Energy efficiency	Adaptability	Scalability	Task handling	Edge-cloud compatibility
Proposed Method (ALBLA)	Optimizes energy consumption by balancing tasks between edge and cloud environments using learning automata	Highly adaptive to changes in both edge and cloud environments through iterative learning automata updates	Scales effectively with the number of devices and tasks due to LA's real-time decision-making based on feedback	Designed for dynamic real-time IoT tasks, it improves response time and resource utilization across edge and cloud	Highly suitable for edge-cloud environments due to its dynamic and adaptive nature

changing network conditions and optimize resource utilization, making it a promising solution for future edge-cloud systems.

6 Conclusion

This paper presents a novel adaptive load-balancing algorithm for edge-cloud systems, leveraging learning automata to optimize task allocation and resource utilization. The proposed algorithm addresses the challenges of limited processing capacity, inefficient load balancing, and extended communication links by dynamically selecting servers for task processing based on real-time feedback. The algorithm adaptively learns from past performance by employing learning automata to make informed decisions, improving its efficiency and effectiveness over time. The proposed algorithm has been rigorously evaluated through simulations, demonstrating superior performance compared to existing methods regarding average service time, task waiting time, network traffic, and successful task executions on edge servers. Future research directions include integrating learning automata with energy-aware [54] and deep learning techniques to enhance the algorithm's adaptability and resilience to dynamic network conditions. Additionally, incorporating fault models for mobile devices and edge servers will improve the algorithm's robustness and scalability.

Author contributions Mehdi Ghorbani: Methodology, Software, Visualization, Investigation. Navid Khaledian: Supervisor, Conceptualization, Methodology, Writing- Original draft preparation. Setareh Moazzami: Data curation, Writing- Reviewing and Editing, Validation.

Funding No funding was obtained for this study.

Availability of data and materials No datasets were generated or analysed during the current study.

Declarations

Conflict of interests The authors declare no competing interests.

References

1. Apat HK, Nayak R, Sahoo B (2023) A comprehensive review on Internet of Things application placement in Fog computing environment. Internet Things 100866
2. Oliveira F, Costa DG, Assis F, Silva I (2024) Internet of intelligent things: a convergence of embedded systems, edge computing and machine learning. Internet Things 101153
3. Choudhury A, Ghose M, Islam A (2024) Machine learning-based computation offloading in multi-access edge computing: a survey. J Syst Arch 103090
4. Khaledian N, Voelp M, Azizi S, Shirvani MH (2024) AI-based & heuristic workflow scheduling in cloud and fog computing: a systematic review. Clust Comput 27(8):10265–10298. https://doi.org/10.1007/s10586-024-04442-2
5. Shahmirzadi D, Khaledian N, Rahmani AM (2024) Analyzing the impact of various parameters on job scheduling in the Google cluster dataset. Clust Comput 27(6):7673–7687. https://doi.org/10.1007/s10586-024-04377-8

6. Huaranga-Junco E, González-Gerpe S, Castillo-Cara M, Cimmino A, García-Castro R (2024) From cloud and fog computing to federated-fog computing: a comparative analysis of computational resources in real-time IoT applications based on semantic interoperability. Futur Gener Comput Syst 159:134–150
7. Cao J, Lam K-Y, Lee L-H, Liu X, Hui P, Xiang Su (2023) Mobile augmented reality: user interfaces, frameworks, and intelligence. ACM Comput Surv 55(9):1–36
8. Chen Y, Lin Y, Zheng Z, Yu P, Shen J, Guo M (2022) Preference-aware edge server placement in the internet of things. IEEE Internet Things J 9(2):1289–1299
9. Wu H, Geng J, Bai X, Jin S (2024) Deep reinforcement learning-based online task offloading in mobile edge computing networks. Inf Sci 654:119849
10. Khaledian N, Khamforoosh K, Akraminejad R, Abualigah L, Javaheri D (2024) An energy-efficient and deadline-aware workflow scheduling algorithm in the fog and cloud environment. Computing 106(1):109–137. https://doi.org/10.1007/s00607-023-01215-4
11. Sarhadi A, Torkestani JA (2023) Cost-effective scheduling and load balancing algorithms in cloud computing using learning automata. Comput Inf 42(1):1. https://doi.org/10.31577/cai-2023-1-37
12. Liang H, Zhang X, Zhang J, Li Q, Zhou S, Zhao L (2019) A novel adaptive resource allocation model based on SMDP and reinforcement learning algorithm in vehicular cloud system. IEEE Trans Veh Technol 68(10):10018–10029. https://doi.org/10.1109/TVT.2019.2937842
13. Gosavi A (2009) Reinforcement learning: a tutorial survey and recent advances. Inf J Comput 21(2):178–192. https://doi.org/10.1287/ijoc.1080.0305
14. Khaledian N, Khamforoosh K, Azizi S, Maihami V (2023) IKH-EFT: an improved method of workflow scheduling using the krill herd algorithm in the fog-cloud environment. Sustain Comput Inform Syst 37:100834
15. Lin J, Huang S, Zhang H, Yang X, Zhao P (2023) A deep-reinforcement-learning-based computation offloading with mobile vehicles in vehicular edge computing. IEEE Internet Things J 10(17):15501–15514
16. Zhao J, Li Q, Ma X, Richard YuF (2023) Computation offloading for edge intelligence in two-tier heterogeneous networks. IEEE Trans Netw Sci Eng 11(2):1872–1884
17. Wang D, Yi Y, Yan S, Wan Na, Zhao J (2023) A node trust evaluation method of vehicle-road-cloud collaborative system based on federated learning. Ad Hoc Netw 138:103013
18. Oroojlooy A, Hajinezhad D (2023) A review of cooperative multi-agent deep reinforcement learning. Appl Intell 53(11):13677–13722
19. Abofathi Y, Anari B, Masdari M (2024) A learning automata based approach for module placement in fog computing environment. Expert Syst Appl 237:121607
20. Vafashoar R, Morshedlou H, Rezvanian A, Meybodi MR (2021) Cellular learning automata: theory and applications, vol 307. Springer
21. Su S, Xiang Ju (2023) A cellular learning automata-based approach for self-protection and coverage problem in the Internet of Things. Internet Things 22:100718
22. Wang X, Gaoyang Wu (2024) Learning automata based routing and content delivery for vehicular named data networking. Eng Appl Artif Intell 136:109043
23. Billard E, Lakshmivarahan S (1998) Simulation of period-doubling behaviour in distributed learning automata. In: Proceedings of the 1998 ACM symposium on applied computing, pp 690–695
24. Zhan W, Luo C, Wang J, Wang C, Min G, Duan H, Zhu Q (2020) Deep-reinforcement-learning-based offloading scheduling for vehicular edge computing. IEEE Internet Things J 7(6):5449–5465
25. Dong L, Wu W, Guo Q, Satpute MN, Znati T, Du DZ (2021) Reliability-aware offloading and allocation in multilevel edge computing system. IEEE Trans Reliab 70(1):200–211. https://doi.org/10.1109/TR.2019.2909279
26. Wang J, Liu K, Li B, Liu T, Li R, Han Z (2020) Delay-sensitive multi-period computation offloading with reliability guarantees in fog networks. IEEE Trans Mob Comput 19(9):2062–2075. https://doi.org/10.1109/TMC.2019.2918773
27. Lim J, Lee D (2020) A load balancing algorithm for mobile devices in edge cloud computing environments. Electronics 9(4):4. https://doi.org/10.3390/electronics9040686
28. Hoseiny F, Azizi S, Shojafar M, Tafazolli R (2021) Joint QoS-aware and cost-efficient task scheduling for fog-cloud resources in a volunteer computing system. ACM Trans Internet Technol. https://doi.org/10.48550/ARXIV.2104.13974

29. Liu T, Fang L, Zhu Y, Tong W, Yang Y (2022) A near-optimal approach for online task offloading and resource allocation in edge-cloud orchestrated computing. IEEE Trans Mob Comput 21(8):2687–2700. https://doi.org/10.1109/TMC.2020.3045471
30. Li J et al (2022) Maximizing user service satisfaction for delay-sensitive iot applications in edge computing. IEEE Trans Parallel Distrib Syst 33(5):1199–1212. https://doi.org/10.1109/TPDS.2021.3107137
31. Dai F, Liu G, Mo Q, Xu W, Huang B (2023) Correction to: task offloading for vehicular edge computing with edge-cloud cooperation. World Wide Web 26(2):633–633. https://doi.org/10.1007/s11280-022-01064-9
32. Long S, Zhang Y, Deng Q, Pei T, Ouyang J, Xia Z (2023) An efficient task offloading approach based on multi-objective evolutionary algorithm in cloud- edge collaborative environment. IEEE Trans Netw Sci Eng 10(2):645–657. https://doi.org/10.1109/TNSE.2022.3217085
33. Tang T, Li C, Liu F (2023) Collaborative cloud-edge-end task offloading with task dependency based on deep reinforcement learning. Comput Commun 209:78–90. https://doi.org/10.1016/j.comcom.2023.06.021
34. Laili Y, Guo F, Ren L, Li X, Li Y, Zhang L (2023) Parallel scheduling of large-scale tasks for industrial cloud-edge collaboration. IEEE Internet Things J 10(4):3231–3242. https://doi.org/10.1109/JIOT.2021.3139689
35. Siyadatzadeh R et al (2023) ReLIEF: a reinforcement-learning-based real-time task assignment strategy in emerging fault-tolerant fog computing. IEEE Internet Things J 10(12):1075210763. https://doi.org/10.1109/JIOT.2023.3240007
36. Ebrahim Pourian R, Fartash M, Akbari Torkestani J (2022) A new approach to the resource allocation problem in fog computing based on learning automata. Cybern Syst. https://doi.org/10.1080/01969722.2022.2145653
37. Du Z, Peng C, Yoshinaga T, Wu C (2023) A Q-learning-based load balancing method for real-time task processing in edge-cloud networks. Electronics. https://doi.org/10.3390/electronics12153254
38. Liu L, Zhu H, Wang T, Tang M (2024) A fast and efficient task offloading approach in edge-cloud collaboration environment. Electronics 13(2):2. https://doi.org/10.3390/electronics13020313
39. Ullah I, Lim HK, Seok YJ et al (2023) Optimizing task offloading and resource allocation in edge-cloud networks: a DRL approach. J Cloud Comp 12:112. https://doi.org/10.1186/s13677-023-00461-3
40. Rahmani TA, Belalem G, Mahmoudi SA et al (2024) Machine learning-driven energy-efficient load balancing for real-time heterogeneous systems. Clust Comput 27:4883–4908. https://doi.org/10.1007/s10586-023-04215-3
41. Wehbi O, Arisdakessian S, Wahab OA et al (2023) Fedmint: Intelligent bilateral client selection in federated learning with newcomer IoT devices. IEEE Internet Things J 10(23):20884–20898
42. Bai J, Chen Y (2023) The node selection strategy for federated learning in UAV-assisted edge computing environment. IEEE Internet Things J 10(15):13908–13919
43. Sonmez C, Ozgovde A, Ersoy C (2018) EdgeCloudSim: an environment for performance evaluation of edge computing systems. Trans Emerg Telecommun Technol 29(11):e3493. https://doi.org/10.1002/ett.3493
44. Goyal T, Singh A, Agrawal A (2012) Cloudsim: a simulator for cloud computing infrastructure and modelling. Int Conf Model Optim Comput 38:3566–3572. https://doi.org/10.1016/j.proeng.2012.06.412
45. Hensen B (2023) A systematic literature review of mixed reality learning approaches. In: De Paolis LT, Arpaia P, Sacco M (eds) Extended reality. Springer Nature Switzerland, Cham, pp 15–34
46. Ahmed S, Irfan S, Kiran N, Masood N, Anjum N, Ramzan N (2023) Remote health monitoring systems for elderly people: a survey. Sensors. https://doi.org/10.3390/s23167095
47. Jaseena KU, Kovoor BC (2022) Deterministic weather forecasting models based on intelligent predictors: a survey. J King Saud Univ Comput Inf Sci 34(6):3393–3412. https://doi.org/10.1016/j.jksuci.2020.09.009
48. Aazam M, Huh EN (2014) Broker as a service (baas) pricing and resource estimation model. In: 2014 IEEE 6th international conference on cloud computing technology and science, pp 463–468. IEEE
49. Dinh TQ, Tang J, La QD, Quek TQS (2017) Offloading in mobile edge computing: task allocation and computational frequency scaling. IEEE Trans Commun 65(8):3571–3584
50. Mao Y, Zhang J, Letaief KB (2017) Dynamic computation offloading for mobile-edge computing with energy harvesting devices. IEEE J Sel Areas Commun 34(12):3590–3605

51. Wang S, Zhao Y, Xu J, Yuan J, Hsu CH (2019) Edge server placement in mobile edge computing. J Parallel Distrib Comput 127:160–168
52. Zhou Z, Shojafar M, Alazab M, Abawajy J, Li F (2021) AFED-EF: an energy-efficient VM allocation algorithm for IoT applications in a cloud data center. IEEE Trans Green Commun Netw 5(2):658–669. https://doi.org/10.1109/TGCN.2021.3067309
53. Zhou Z, Abawajy J, Chowdhury M, Hu Z, Li K, Cheng H, Li F (2018) Minimizing SLA violation and power consumption in Cloud data centres using adaptive energy-aware algorithms. Future Gener Comput Syst 86:836–850
54. Zhou Z, Shojafar M, Alazab M, Li F (2022) IECL: an intelligent energy consumption model for cloud manufacturing. IEEE Trans Industr Inf 18(12):8967–8976

Authors and Affiliations

Mehdi Ghorbani[1] · Navid Khaledian[2] · Setareh Moazzami[3]

✉ Mehdi Ghorbani
mehdi.ghorbani@ssaa.ir

Navid Khaledian
navid.khaledian@uni.lu

Setareh Moazzami
s.moazzami@iau-tnb.ac.ir

[1] Department of Computer Engineering and Information Technology, Qazvin Branch, Islamic Azad University, Qazvin, Iran

[2] Interdisciplinary Centre for Security, Reliability and Trust (SnT), University of Luxembourg, Esch-Sur-Alzette, Luxembourg

[3] Department of Computer Engineering, Tehran North Branch, Islamic Azad University, Tehran, Iran