Hindawi
Security and Communication Networks
Volume 2023, Article ID 9842763, 1 page
https://doi.org/10.1155/2023/9842763

Retraction

Retracted: Mitigation Impact of Energy and Time Delay for Computation Offloading in an Industrial IoT Environment Using Levenshtein Distance Algorithm

Security and Communication Networks

Received 5 December 2023; Accepted 5 December 2023; Published 6 December 2023

This article has been retracted by Hindawi, as publisher, following an investigation undertaken by the publisher [1]. This investigation has uncovered evidence of systematic manipulation of the publication and peer-review process. We cannot, therefore, vouch for the reliability or integrity of this article.

Please note that this notice is intended solely to alert readers that the peer-review process of this article has been compromised.

Wiley and Hindawi regret that the usual quality checks did not identify these issues before publication and have since put additional measures in place to safeguard research integrity.

We wish to credit our Research Integrity and Research Publishing teams and anonymous and named external researchers and research integrity experts for contributing to this investigation.

The corresponding author, as the representative of all authors, has been given the opportunity to register their agreement or disagreement to this retraction. We have kept a record of any response received.

References

[1] A. Rafiq, P. Wang, M. Wei, M. S. Ali Muthanna, and N. N. Josbert, "Mitigation Impact of Energy and Time Delay for Computation Offloading in an Industrial IoT Environment Using Levenshtein Distance Algorithm," *Security and Communication Networks*, vol. 2022, Article ID 6469380, 12 pages, 2022.

Hindawi
Security and Communication Networks
Volume 2022, Article ID 6469380, 12 pages
https://doi.org/10.1155/2022/6469380

Research Article

Mitigation Impact of Energy and Time Delay for Computation Offloading in an Industrial IoT Environment Using Levenshtein Distance Algorithm

Ahsan Rafiq,[1] **Ping Wang**,[2] **Min Wei**,[2] **Muhammad Saleh Ali Muthanna**,[1] **and Nteziriza Nkerabahizi Josbert**[1]

[1]*Department of Computer Science, Chongqing University of Posts and Telecommunication, Chongqing 400065, China*
[2]*Department of Automation, Chongqing University of Posts and Telecommunication, Chongqing 400065, China*

Correspondence should be addressed to Min Wei; weimin@cqupt.edu.cn

Received 24 November 2021; Accepted 12 January 2022; Published 12 February 2022

Academic Editor: Muhammad Arif

Due to the explosive growth of the Internet of things (IoT) devices and the emergence of diverse new applications, network traffic volume is growing exponentially. The traditional centralized network architecture cannot fulfill IoT devices demand because of the heavy network traffic in industrial IoT. Moreover, IoT devices have limited computational ability and battery power. Energy consumption and time delay problems during computation offloading are fundamental issues. A new architecture known as mobile edge computing (MEC) was introduced to overcome these issues, which brings cloud services and its contents to the edge of the network. IoT devices can offload the data for computation to the cloud server or edge nodes. Different schemes have been proposed to overcome this problem under many scenarios (i.e., single-user, multiuser, and vehicular networks). In this paper, we proposed a modified delay mitigation Levenshtein distance algorithm (MDML). We consider an industrial scenario with multiple IoT devices and multiple servers (edge nodes). Each edge node consists of one MEC server. The proposed algorithm solves the offloading optimization problem of energy and mitigation of time delay with much lower complexity while significantly reducing offloading tasks' execution time. It works on the basis of dynamic programming, where we break down a complex problem into subproblems. Performance evaluation of our proposed algorithm shows that it can achieve satisfactory energy efficiency and mitigate time delay in the industrial IoT environment.

1. Introduction

Smart industries called industry 4.0 are the future of the industrial revolution that will enable humanity to fulfill its manufacturing conditions with accuracy and efficiency in the industrial Internet of things (IoT) [1, 2]. The use of actuators, sensors, and IoT devices in the industrial IoT is increasing rapidly; devices communicate with each other to work efficiently [3]. Industrial IoT applications that consume high computation need powerful computing capacity and deserve extraordinary energy consumption to process them locally on the devices [4]. The battery and computation capacity is limited for IoT devices.

New emerging network architecture is mobile edge computing (MEC); the concept of MEC is one that provides user equipment (UE) application at the edge of the network with cloud computing aptitude and Information technology (IT) service environment [5, 6]. It can aggravate resource constraints at UEs by offloading computation tasks from UEs to MEC and can help them to secure the use of different IoT applications. In this revolutionary world of technology, IoT devices need to provide high performance within a short time and less energy with wireless networks to share data and information. Therefore, to resolve this problem, the best way is to offload all the device tasks on the cloud or edge devices. The mobile edge computing application becomes

free after offloading and results in high performance. While using advanced MEC servers at wireless access point and edge nodes leveraging MEC resources, IoT devices can offload their computation tasks partially to access point and edge nodes; these tasks can be computed remotely by the MEC servers and locally.

Moreover, the computation performance rate is low without MEC due to the user's distance from cloud central servers. MEC brings computing and storage resources to the network edge. The modern progress of IoT has motivated different latest applications, and motivation is to enhance their communication skills [7]. A complicated task to be achieved is how to deal with these devices with enhanced computation ability, especially when those are of a minor size and have less power [8]. The computation offloading framework is the decision engine. The devices with low computation or storage power can get help from computational offloading performing heavy tasks on MEC servers [9]. The task will be affected directly in terms of time, processing, and energy. Therefore, devices should offload the task between other conceivable metrics after making an offloading decision based on necessary predictions like energy and time. Computation solutions for offloading were proposed to conquer these restrictions.

In particular, for all system individuals, we target to achieve energy-efficient task offloading. An energy and time cost minimization issue is indicated. The indicated optimization issue is an edit distance problem, which is difficult to decrypt; we centered an algorithm based on the Levenshtein distance. More specifically, we demonstrate that our algorithm accomplishes energy optimization and mitigation of time delay. Finally, simulations are conducted to validate the performance of our proposed algorithm. Dynamic programming (DP) is an algorithmic technique, in which a complex problem is divided into simple and easily solvable problems. Then, every solvable problem has its solution, which is built from the previous subproblem. When we find the final solution, we take the already found results and compute them [10]. DP is a beneficial repetitive process, and DP's effectiveness has been substantiated in different areas of wireless communication. It is a useful technique to solve the problems of computation time, optimization, load regulation [11], vehicle routing [12], power systems [13], and so on can be solved with this. Researchers have also used DP for energy management in long-range electric vehicles. DP is used to recompute optimal constraints by monitoring the energy in real time [14].

In this work, we have proposed the modified delay mitigation Levenshtein distance algorithm (MDML) to optimize the energy efficiency and time delay. This algorithm solves the optimization problem, which is proved by the simulations using MATLAB. We aim to get the possible way to improve the efficiency of energy and time delay mitigation. The main contributions of this article are the following:

(i) We formulate the energy and time as an optimization problem in an industrial scenario where IoT devices offload their tasks to a nearby MEC server.

(ii) Energy and latency are related to task transmission. We break down a complex problem into subproblems to minimize energy cost and secure the user task's latency requirements.

(iii) We proposed a modified delay mitigation Levenshtein distance algorithm (MDML) to solve the energy optimization and mitigation of time delay.

The remainder of this paper is as follows: the state-of-the-art-related work is outlined in Section 2. The system model and issue preparation are summarized in Section 3. The planned algorithm built on the Levenshtein distance is explained in Section 4. Section 5 examines the performance evaluation and reproduction outcomes. Finally, in Section 6, we give the result of this paper.

2. Related Work

In recent studies, the task offloading issue has been investigated. An energy-efficient task offloading algorithm that aims to enhance the decision of offloading through and the requirement of the quality of service (QoS) in a heterogeneous network was proposed in [15]. Optimizing the latency via implementing a collaborative call graph method was presented by the authors in [16]. A game theory method for distributed task offloading was utilized in [17] to enhance the decision-making for task offloading. The authors in [18] aimed to improve the weighted sum computation rate by studying the joint problem of transmitting time allocation and offloading decision-making. To improve tasks' accomplishment performance and utilize MEC resources effectively, IoT devices can partially execute their tasks. The computation task can be cut into small parts; various parts of the main task can be performed remotely via the MEC server and locally in the device itself. By utilizing the approach of partial offloading, the latency required to finish the whole task was reduced efficiently in [19].

A computational offloading problem was proposed to improve energy efficiency and latency with the 6TiSCH network in smart industries. For practical scenarios using rough set-based root selection (R2S) algorithm for network layer energy and Q-learning algorithm for offloading to manage the energy consumption among edge nodes [20]. The computation offloading can be applied in dual ways that are binary and partial offloading. In the binary offloading situation, the computation task is not partitioned and must be offloaded as an entire. In the one-sided offloading case, the job can be divided into subslices, and individually single task is offloaded. The partial offloading mode was proposed in [21], where the problem was formulated as a cost execution problem (total sum of energy consumption and latency). Three different computing modes (full local, full MEC, and partial) were used for task execution, where the end-user can choose one of them. To achieve the optimal solution, Kuhn–Munkres algorithm and extensive search method were deployed. It has fascinated rising research benefits in mutually academia and industry.

In the larger timescale, whether a task should execute locally or offload to the MEC server, a Markov decision

process approach was proposed in [22], where the computation tasks are scheduled based on the queueing state of the task buffer. The authors formulated a power-constrained delay minimization problem and proposed an efficient one-dimensional search algorithm to find the optimal task scheduling policy. This method can offload a single task at a time to the edge server. A novel offloading algorithm named Dynamic Programming with Hamming Distance Termination (DPH) was presented in [23]. To find a nearly optimal offloading solution quickly, it uses randomization and a hamming distance termination criterion. DPH can offload multiple tasks to the cloud when the network transmission bandwidth is high and minimize the energy use of the mobile device. However, the time delay mitigation problem still exists in this scenario. Authors developed a prototype to explore IoT devices that communicate with a cloud-based controller in [24]; the controller is offloaded to fog and cloud.

The experiments were performed while considering arbitrary jitter and delays. This prototype applies mitigation mechanisms to deal with the delays and jitter caused by the networks when the controller is offloaded to the fog. Still, the performance of this prototype is not sufficient to deal with energy efficiency. The authors formulated the offloading ratio optimization problem based on the Markov process in [25]. They proposed a novel Congestion Optimal WiFi Offloading (COWO) algorithm based on the subgradient method, which aims to obtain the optimal offloading ratio for each access point (AP) to maximize the throughput and minimize the network congestion. This research investigates the impact of user mobility on WiFi offloading performance. Simulation results show that the COWO algorithm could achieve higher throughput compared with other offloading schemes. Still, this scheme supports the single task that can be offloaded to multiple APs and is not suitable in an industrial environment where various tasks need to be offloaded to achieve energy efficiency and adequate mitigation of time delay. Authors apply a novel deep reinforcement learning approach in [26] to automatically decide the optimal allocation of the network resources. In this approach, mobile edge computing was considered in network caching and device-to-device (D2D) communications.

Google TensorFlow was used to implement the deep Q-learning approach. The simulation results of this approach with different network parameters were effective for offloading multiple tasks, but it consumes more energy that is not suitable for the Industrial environment. A reinforcement learning- (RL-) based offloading scheme was proposed in [27] for an IoT device with energy harvesting (EH) to select the edge device and the offloading rate according to the current battery level. This scheme allows IoT devices to optimize the offloading policy without knowing the computation latency, MEC, and the energy consumption models, performance constraints in terms of the energy consumption, and computation latency for an IoT device that uses wireless power transfer for energy harvesting. Simulation results show that the RL-based offloading scheme reduces the computation latency and energy consumption. Still, this scheme supports the single user only, which is not suitable for industrial IoT scenarios. Authors developed an efficient, low-complexity algorithm in [28] using particle swarm optimization, with a heuristic (i.e., metaheuristic) that performed well at solving a complicated optimization problem. This research investigated a two-tier computation offloading strategy for multiuser multiservers in heterogeneous networks to minimize the energy consumption and completion time. Simulation results revealed that the proposed algorithm reduced the total computing overhead of mobile devices, but this strategy was not performed in the industrial scenario.

An energy transmitter scheduling scheme was proposed in [29], for IoT nodes to minimize the cost of charging while keeping IoT nodes sufficiently charged and an energy-aware mode switching strategy was proposed to enable IoT nodes to perform either on-demand sensing or dedicated wireless power transfer (WPT). The simulation results for IoT node scheduling demonstrate that less than 50% of IoT nodes need to be activated in all scenarios to complete the tasks. This scheme shows significant energy reduction in the overall Industrial environment, but it can offload a single task at a time from one IoT node. The authors presented an efficient multiplayer with multitask computation offloading model in [30], with definite performance in network latency and energy consumption for virtual reality (VR) applications based on mobile edge computing. This model was formulated as an integer optimization problem; the objective is to minimize the sum cost in terms of network latency and energy consumption. Simulation results prove that the network latency and energy consumption can be reduced by using this model, but this model was not designed for the industrial environment.

Mobile edge computing uses a capable method to offload computation tasks. Meanwhile, amusing computing resources can meaningfully decrease application processing potential. MEC is a very familiar edge computing that occurs nearby data sources. Moreover, it brings the closeness of user, computing, and storing resources of significant servers to the network edge. The device that has computing ability is also distinct, although that device can be an edge device. However, the computational capability of cloud server resources is better than that from the edge node or server within the particular area for MEC computations [31].

Such as a capable skill, mobile edge computing (MEC) is introduced to supply computing services at the network edge. Dissimilar to the predictable cloud computing, which is inaccessible from the IoT devices, MEC can be installed at the radio access point, such as a base place. MEC can support decreasing the service latency and movement of the central network. The IoT devices can obtain enhanced computing service and extend the battery timing by offloading the computation tasks. That is why the task offloading in MEC for IoT has fascinated important consideration from both industry and academia. Mobile Edge Computing Offloading (MECO) has been getting further attention from researchers, and piles of analysis and research on MECO have been approved in recent years [32]. Energy and wireless channel results are time-varying for mobile users and as a stochastic game formulated. MECO problem expressed the

computation offloading decision production mutually with interference management. Besides, substantial resource block portion and computation resource allocation are described as optimization problem in mobile-edge computing (MEC). Several IoT devices connect to a mutual cloud/edge server where multicell networks are individually focused on the situations. In recent times, there have progressed particular works focusing on MECO. Deep reinforcement learning is used by [33], for adaptive real-time offloading decisions in MEC.

3. System Model and Problem Formulation

We consider an application that comprises two types of assignments (i.e., offloadable tasks and unoffloadable tasks). We can allude unoffloadable as local tasks. These tasks are processed locally, and they handle the collaboration of clients and access local input and output devices. They can also access specific data on a mobile device for mobile users. The decision of which task is to be processed locally and which is to be processed on the server takes energy and time. Figure 1 shows the mobile edge computing system model for offloading a task from a client node or IoT mobile device to a peripheral (MEC) server.

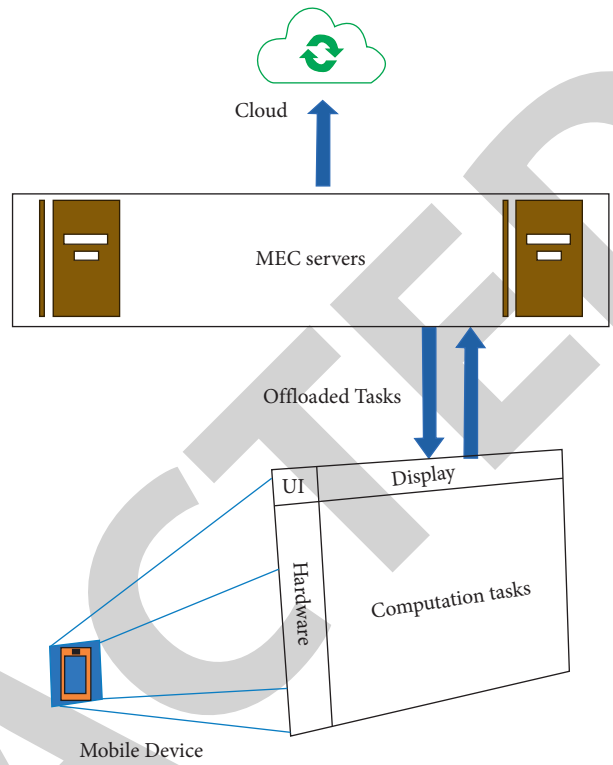

Figure 1: Mobile edge computing system model.

3.1. Network Model. We consider an industrial scenario where the MEC framework comprises one server and different smart devices and client nodes. Mobile edge computing architecture is shown in Figure 2. At the system level, it has user equipment (UE) applications and third-party software. UE will send his data to MEC Host, where traffic rules, virtualization infrastructure, and data plane are defined along with MEC applications. At the end of the architecture, we have a network layer to control the network traffic flow.

Different IoT and mobile devices are processing specific tasks in Industrial IoT. The mobile device runs N number of independent tasks $i \in \{0, 1 \ldots, N\}$; every application has two buffer queues: one is the task queue, and the other is the return queue. These tasks can be executed locally or exchanged to the cloud for computation [34]. We assume that a WiFi network is accessible for our network model's IoT devices, but network transmission bandwidth still can change dynamically. The time taken to exchange an assignment through a wireless link among a cloud and IoT device is crucial. For all tasks, a complete execution time constraint exists [35]. Dynamic programming (DP) is an algorithmic technique, DP is a beneficial repetitive process, and DP's effectiveness has been substantiated in different areas of wireless communication [10]. It is a useful technique to solve the problems of computation time and energy optimization. Our proposed algorithm works based on DP, so while figuring a decision, our proposed Levenshtein distance-based algorithm considers the present wireless network bandwidth.

The IoT devices make the task assignment recommendation at each schedule vacancy according to the proposed algorithm and wireless network transmission bandwidth availability. The assignments can be executed by the local CPU or cloud server, or they can also be declined. If the battery's energy is enough to be computed, then the task can be computed locally to decrease transmission delay. The job can be offloaded to the mobile edge server, and it will spare the use of energy.

Figure 3 shows the sequence diagram for MEC offloading. First of all, the client node initiates a request to the access network, which passes to edge infrastructure, where the service orderer initiates a session. The access network sends the session contest back to the client node. After creating the session, the client node sends an offloading request to the MEC provider through the service orderer, passing this task to the MEC application. MEC app processes the request and sends back the results to the MEC provider, which forwards these results back to the client through the service order.

3.2. Problem Formulation. We consider a network with different devices and each device executing an N number of tasks. For task i, let $Wi \in \{0, 1\}$ be an indicator of execution. The mobile client first chooses whether its task i ought to be handled locally or offloaded to the cloud. We can denote this choice by

$$Wi = \begin{cases} 0, & \text{task } i \text{ executed locally,} \\ 1, & \text{task } i \text{ exected on cloud.} \end{cases} \quad (1)$$

In case the task i is executed from the MEC server, the value of Wi is taken as 1; on the other hand, when task i is executed locally, the value of Wi is 0. For mobile devices, the

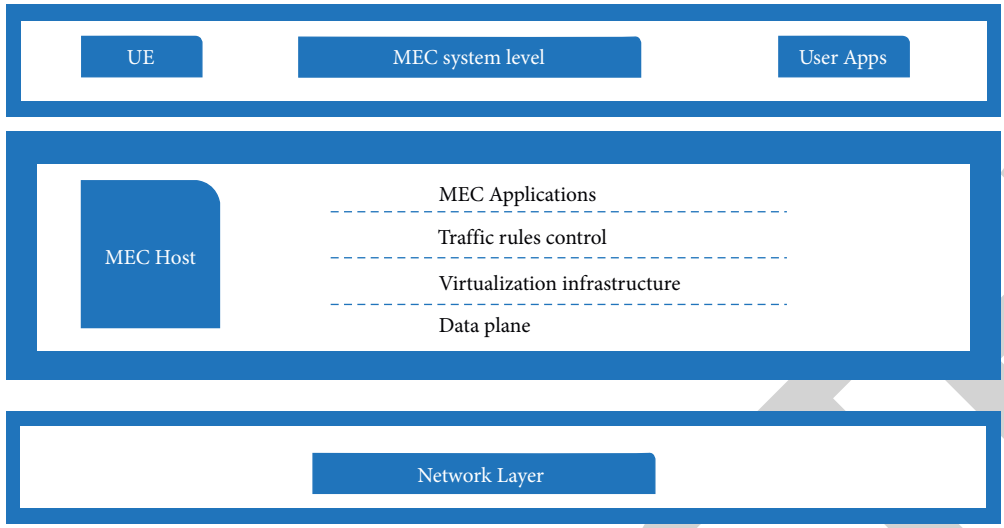

Figure 2: MEC architecture.

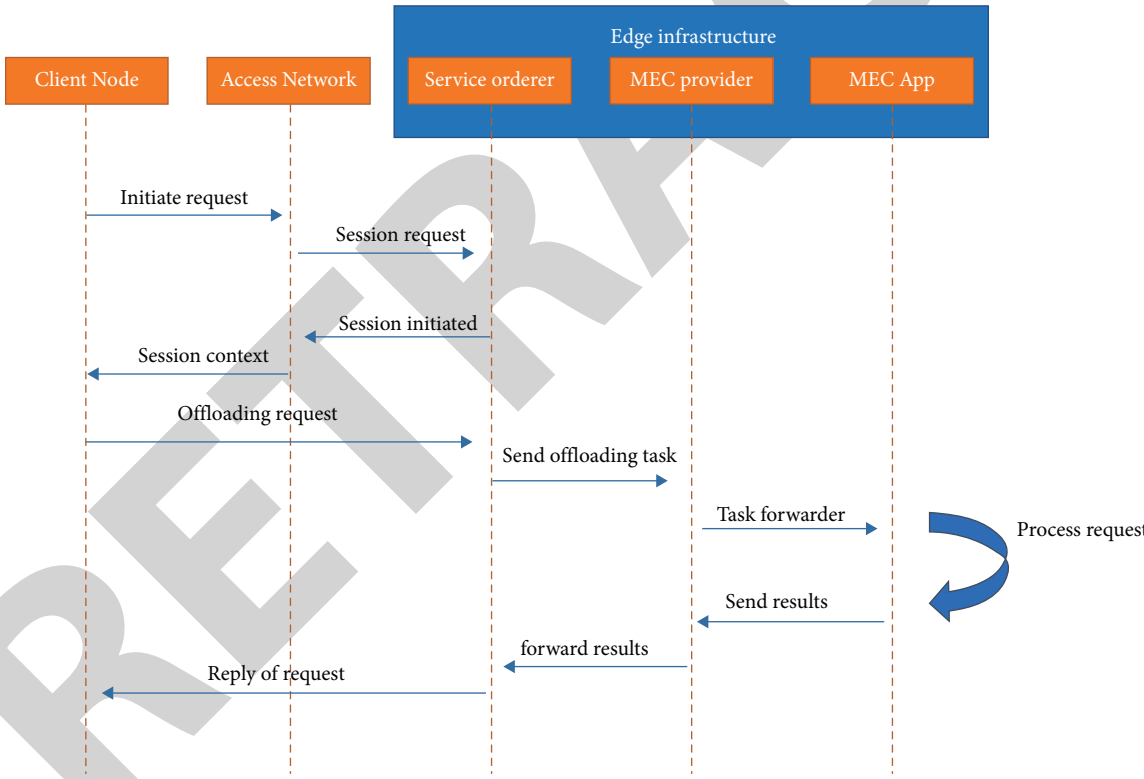

Figure 3: Sequence diagram for MEC offloading.

energy consumption is denoted by *Eri* while for the local execution, the energy consumption is denoted by *ELi*. In case the task *i* is done through the cloud, the cost of energy consumed in transmitting the task to the cloud server is referred to as *EWi*. The execution time for a locally executed task for the task *i* is denoted by *TLi*. When the task is executed remotely through a cloud server, the transmission time is denoted by *Tri*. The bandwidth of the network transmission is solely responsible for the transmission of energy consumed in transferring assignments. Therefore, if there appears any change in the wireless network bandwidth, the offloading selection will be affected [36], if it is assumed that the transmission time is equal to the time taken by each assignment isolated by the network transmission rate. In that case, the decision as to whether the task should be offloaded or not relies on the variation of the transmission rate. Approach execution time and energy consumption functions are described in equations (2) and (3) while

considering the constraint of equation (1), where Er_i is the energy consumption of the IoT device when the task i is accomplished on the MEC server, and Et_i is the energy cost of transmitting task i to the MEC server.

$$E_C = \sum_{i \in N} \left(W_i EL_i + \frac{(1-W)Er_i}{\lambda_k} + (1-W_i)Et_i \right) + \sigma, \quad (2)$$

$$T_E = \sum_{i \in N} \left(W_i TL_i + \frac{(1-W)Tr_i}{\lambda_k} + (1-W_i)Tt_i \right) + \sigma, \quad (3)$$

$$\min_i E_c = T_E \leq T_C. \quad (4)$$

On the execution time, it must be satisfying for all tasks where T_C is the time constraint. We can get the solution to our problem, that is, $\min_i E_c$, if this constraint is fulfilled; we proposed the Levenshtein distance-based algorithm for this purpose. The number of blends of two values Wi to find out the ideal solution develops with the number of tasks to be performed. It will decide the tasks that must be offloaded on the cloud to consume less energy and optimize the imperative execution time. For our simulation, we used the parameters in [23]. The notations and terminologies mentioned in the present paper are tabulated in Table 1.

4. Proposed Algorithm Based on the Levenshtein Distance

This section explained our proposed computation offloading scheme and our modified delay mitigation Levenshtein distance (MDML) algorithm in detail. Levenshtein algorithm is one of the most common algorithms to calculate the edit distance or the similarity of two equal-length strings [37]. It can be explained as the minimum number of single-character edits required to convert one word into another or make the two input strings similar. The bigger the return value is, the less related the two texts are. Three basic operations, insertions, deletions, or substitutions, are performed to achieve the goal. Each operation costs one edit. It is used for many purposes, such as language strings, DNA strings matching, clone detection, and plagiarism detection. We initialize the first row and the first column in the matrix with zeros in the first step. The rest of the matrix will be filled using equation (1). In the matrix, i represents the row number, and K represents the column. The binary values depend on the number of characters represented by row i and column K. If the characters are the same, 0 value will be inserted in the memoization table. If the characters are not the same, then 0 will be added.

To achieve energy and time efficiency, we have used this Levenshtein algorithm. Our proposed algorithm is called a modified delay mitigation Levenshtein distance (MDML). Figure 4 presents the flow diagram of computational offloading. It starts with the availability of the task to be offloaded. If the job is offloadable, then it will be computed locally; otherwise, it will check whether the MEC server is available or not. If it is not possible, then the task should be sent to the queue. Otherwise, it will be offloaded.

Table 1: Parameter settings in the simulation.

Notation	Description	Values
Σ	Sigma	$1e-9$
Λ	Lambda	Ones (1, N)
M	Mutation probability	0.5
TC	T_constraint	700
VL	CPU frequency cycle/s in local	500×10^{-6} cycle/s
VC	CPU frequency cycle/s in cloud	10×109 cycle/s
W	Tasks	15
D_{in}	Input data sizes	Between 1 and 3 M
D_{out}	Output data sizes	Between 10 and 30 MB

In this algorithm, to store the bitstreams that show which assignments must be offloaded, we utilize a 15×15 memoization matrix. Here, we take 15 because, for our simulations, we are considering tasks $W = 15$. To obtain the first solution, the random bitstream is produced. First, we check whether all the randomly generated bits are 1 s or 0 s. If so, then we will again create the bitstream. If the character matches, then 0 will be added; otherwise, 1. An 8×8 two-dimensional memoization matrix is shown in Table 2. To further explain, assume that the number of tasks W is eight, and the first random stream is 00110110, which is placed in the first row of the table, and random stream is 11000111 in the first column of the table. The second bitstream's beginning cell is (1, 2). Table 2 is completed by following the previously mentioned rules.

Whenever a bitstream is generated arbitrarily, equation (2) is used to compute the self-energy accordingly and time according to equation (3) for each task. In the meantime, it evaluates the total energy consumption as well as the time taken for execution. Initially, a random stream of a bit is produced. The energy optimization is checked for a newly created string until the 1st conventional cell; after that, total energy is calculated compared with present total energy at cell. If this precise cell in the table is visited before, we examine the new total energy and time of this cell with the previous one. If the cell's new total energy is less than the earlier one, it will be replaced with the new calculated energy and new computed time. The energy and execution time are updated on the remaining cells. It depends on the new values at this regular cell. This procedure will be repeated for the current stream until the total energy is higher than that of the new bitstream. If total energy is less than or equal to minimum energy and its time is less than or equal to the time constraint and meets the Levenshtein distance, total energy and total time will return each time a new stream is produced. We continue following the course of action of the flow in the memorization matrix. We conclude and recognize a solution that has modified delay mitigation Levenshtein distance (MDML) more significantly than a provided limit from an all 1s stream. If all bitstreams are 0 s, it indicates that all components are executed on the cloud, and if they all are 1 s, it means they all are going to execute locally.

A stepwise algorithm is presented in Algorithm 1. This algorithm initializes with the assigning of the random bitstream to act as decision-makers for offloading. After the

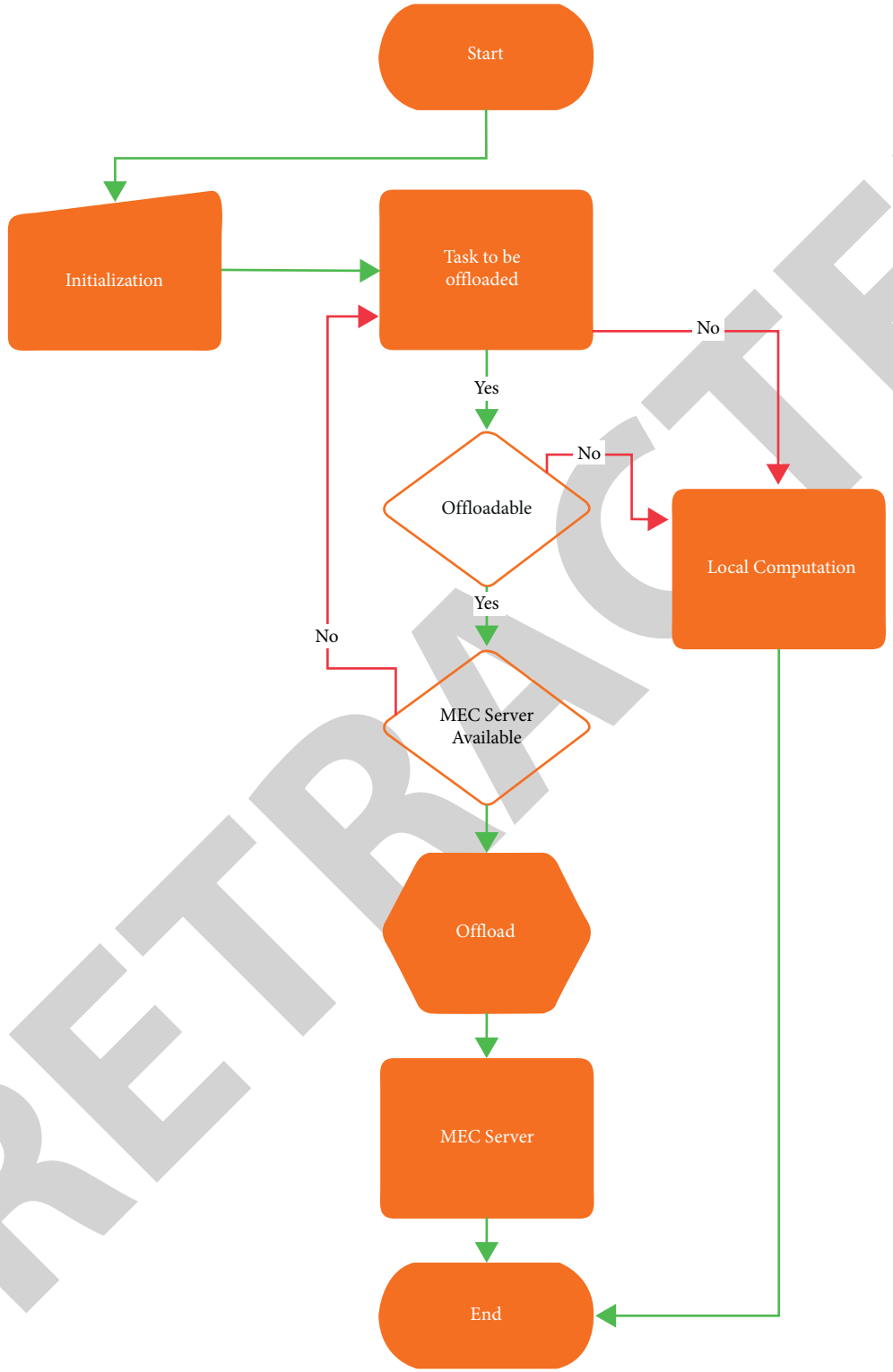

Figure 4: Flowchart of computation offloading.

initialization of the memoization table, it will calculate self-energy and time. If it keeps updating the visited table with every iteration, it will compute the total energy and time at the end of the loop. After finishing the iteration, there is a check whether absolute energy is less than or equal to minimum energy optimization and the same with time. It will also check whether it meets the Levenshtein distance or not.

Table 2: Bitstream using Levenshtein distance.

	0	0	1	1	1	0	0	0
1	1	1	0	0	0	1	1	1
1	1	1	0	0	0	1	1	1
0	0	0	1	1	1	0	0	0
0	0	0	1	1	1	0	0	0
1	1	1	0	0	0	1	1	1
0	0	0	1	1	1	0	0	0
0	0	0	1	1	1	0	0	0
1	1	1	0	0	0	1	1	1

```
for iter = 1 to Total Ite
  initialize random bitstream M
    if all bits are 0s
    go to step 2
    end if
    if all bits are 1s
    go to step 2
    end if
  for k = 1 to k = N
    Insert correct bitstream in the table
      if cell K in table is not visited before
        Calculate self-energy according to equation (2)
        Calculate time according to equation (3)
        Calculate total energy
        Calculate total time
        Update the visited table for that cell K
        Save energy for every tasks E (k) = E(i, j)
        Save time for every tasks T (k) = T(i, j)
      else if this cell K in table is visited before
        Compare the new total time duration and energy of this cell with previous one
      if value of new energy cell is less than the existing value
        Energy of cell K = new calculated energy;
        Time of cell K = new calculated time;
      end if
        Calculate total energy
        Calculate total time
      end if
      if Total Energy is less than or equal to min energy and time less than or equal to time constraint and meet the Levenshtein
distance
      return Total Energy
      return Total time
      end if
  end for
end for
```

Algorithm 1: Modified delay mitigation Levenshtein distance (MDML).

5. Evaluation and Simulation Results

This section introduces the simulation environment and the results obtained by applying the proposed algorithm. To verify the accuracy of the recommended method, we used MATLAB R2019b to simulate and evaluate the system's performance. Charts and graphics are also designed using MATLAB functions. For graphs, we have used the MATLAB plot toolbox [38]. MATLAB needs comprehensive execution time to solve complex problems. In short, we consider only the right amount of power harvesting equipment and servers. To run this method stably, we used a Core i5-8500 processor with 16 GB RAM capacity. Table 3 describes the components used in experimental simulations.

We initialize the characteristics of the mobile device and choose $W = 15$ number of tasks. The computation time for local execution is 4.13×10^{-7} s/bit, and the energy consumption used in the process is 3.12×10^{-7} J/bit. Its CPU frequency 500×10^{-6} cycle/s in local and 10×10^{9} cycles per second in the cloud is considered as task i in our simulations.

Table 3: Simulation environment.

Sr. no.	Component	Description
1	Central processing unit	Intel Core i5-8500
2	Memory	16 GB
3	Operating system	MS Windows 10
4	Integrated development environment	MATLAB-MathWorks
5	Scripting language	MATLAB
6	Graph tool	MATLAB plot gallery

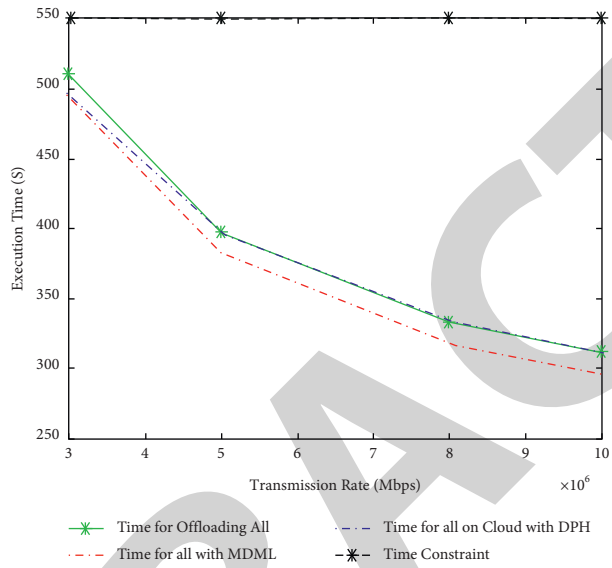

Figure 5: Comparison of execution time.

The input data size is 1 to 3 MB, and the output data size is 10 to 30 MB. When it comes to the size of each output task, it is assumed as uniformly distributed. The consumption of energy is an essential factor in our simulation. To transmit and receive data of the mobile user, energy consumption is 1.42×10^{-7} J/bit as in [15]. The highest transmission rate at the time of task offloading is 15 Mbps.

In the results of our simulations, we expressed total energy cost as a combination of three factors, which include total consumption of energy, cost of offloading, and processing delay. We represent this cost in joules. The energy consumption and execution time of our proposed MDML algorithm are computed using equations (2) and (3), respectively.

Figure 5 illustrates the accomplished execution time obtained as a result of the proposed MDML algorithm and existing approaches. It demonstrates the transmission rate in opposition to the execution time. The outcomes we attained from our proposed MDML algorithm are compared with three scenarios. The first scenario is time for offloading, the second is time for all on the cloud with Dynamic Programming with Hamming Distance Termination (DPH), and the third is with the time constraint of 650. The wireless transmission rate varies between three and ten Mbps while simulations. Since the MDML algorithm is dynamic, it decides which task should be offloaded in accordance with the wireless transmission rate. The proposed algorithm

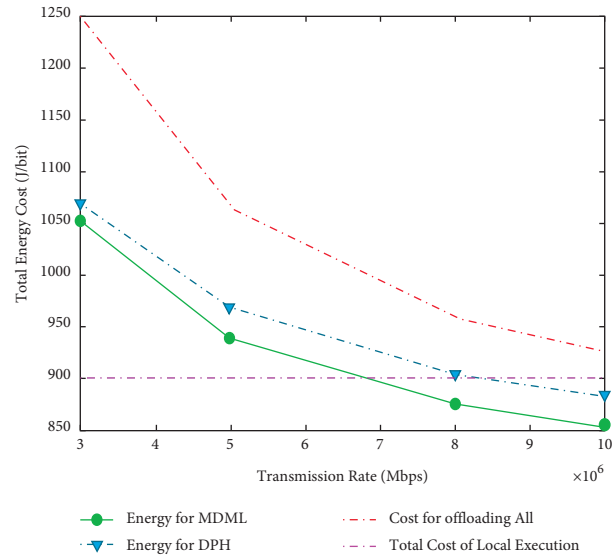

Figure 6: Comparison of total energy cost.

minimizes the execution time throughout the network, and it can be observed that it outperforms the other schemes.

Figure 6 shows the total energy cost accomplished by the proposed MDML algorithm and existing approaches. It shows the total energy cost against the transmission rate

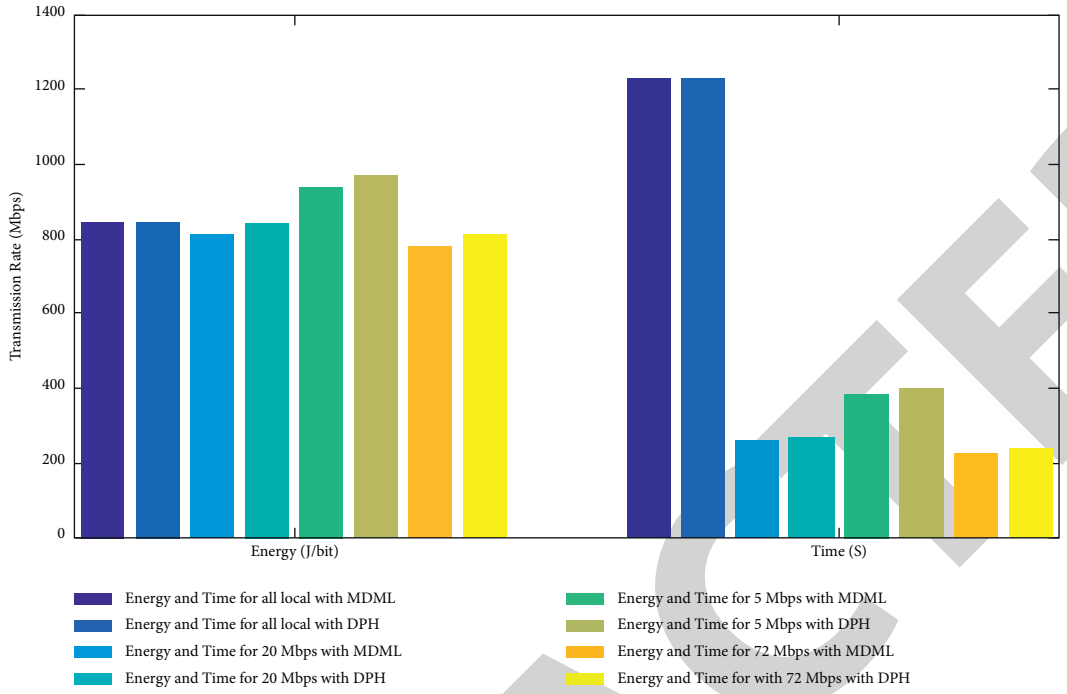

Figure 7: Comparison of energy and time with different transmission rates.

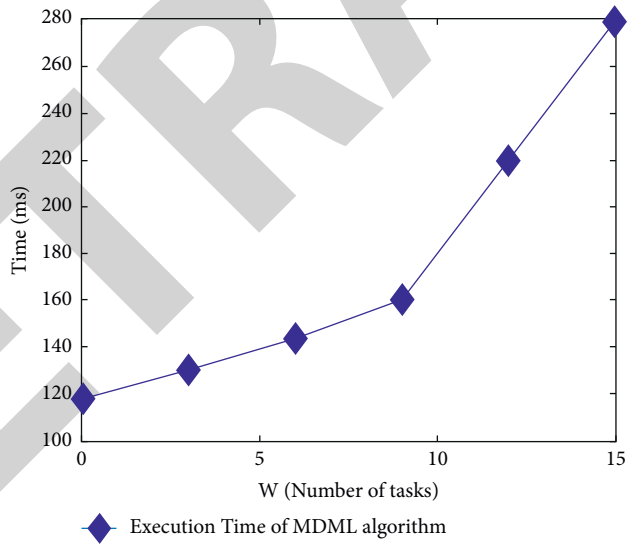

Figure 8: Execution time of MDML for different number of components.

using the mutation probability 5×10^{-7} J/bit. Our acquired results with the MDML algorithm are compared with three scenarios. The first one is energy for DPH, the second is energy cost for offloading all, and the third is the total energy cost of local execution. The proposed method minimizes the total energy cost concerning transmission rate across the network. The obtained results justified that our proposed scheme is most cost-effective as compared to the other methods.

Figure 7 presents the performance of both energy and time through our proposed MDML scheme with other schemes. It displays energy and time performances against various transmission rates.

The outcomes for energy and time using our proposed MDML algorithm are compared with three transmission rates scenarios. The first is energy and time for 20 Mbps with DPH, the second is energy and time for 5 Mbps with DPH, and the third is energy and time for 72 Mbps with DPH. In addition, we use energy and time for all local DPH to compare with our proposed MDML algorithm. The simulation results indicate that our MDML algorithm outperforms the other existing approaches in different

transmission rates. In Figure 8, we plot the total execution time and the transmission rate. It shows that the benefits of offloading increase with the increment in transmission rate and decrement in total execution.

6. Conclusion

Industrial IoT devices have limited computational and battery power. A new emerging paradigm known as mobile edge computing (MEC) is introduced to overcome these issues bringing cloud services and contents to the network edge. IoT devices can offload the data for computation to the cloud server or edge nodes. However, with the increment in the number of IoT devices, energy and time delay issues occur during the computational offloading. A smart device should be able to choose between offloadable and unoffloadable tasks. In this way, it can minimize the consumption of energy and also meet the time constraint. We proposed our optimal Modified Delay Mitigation Levenshtein distance algorithm based on an algorithmic technique called Dynamic programming (DP) to achieve these goals. The simulation results proved that our proposed MDML algorithm could help find optimal energy and time delay mitigation solutions. This algorithm is based on DP so that it can adapt to the changes in the industrial network. Currently, this algorithm works well when the transmission rate is low or high but does not provide excellent results for average transmission rates. In the future, we plan to use this algorithm to improve average transmission rates and explore the security aspects of edge computing from the perspective of energy efficiency.

Data Availability

Any data will be made available from the corresponding author upon request.

Conflicts of Interest

The authors declare that they have no conflicts of interest regarding the present study.

Acknowledgments

This work was supported by the National Key Research and Development Program of China (2020YFB1708800) and the Chongqing Talent Plan Project (cstc2021ycjh-bgzxm0206).

References

[1] G. Salierno, L. Leonardi, and G. Cabri, "The future of factories: different trends," *Applied Sciences*, vol. 11, no. 21, p. 9980, 2021.

[2] P. W. Khan, Y.-C. Byun, and N. Park, "IoT-blockchain enabled optimized provenance system for food industry 4.0 using advanced deep learning," *Sensors*, vol. 20, no. 10, p. 2990, 2020.

[3] P. W. Khan and Y. Byun, "A blockchain-based secure image encryption scheme for the industrial internet of things," *Entropy*, vol. 22, no. 2, p. 175, 2020.

[4] B. Seok, J. Park, and J. H. Park, "A lightweight hash-based blockchain architecture for industrial IoT," *Applied Sciences*, vol. 9, no. 18, p. 3740, 2019.

[5] S.-C. Wang, W.-S. Hsiung, C.-F. Hsieh, and Y.-T. Tsai, "Reliability enhancement of edge computing paradigm using agreement," *Symmetry*, vol. 11, no. 2, p. 167, 2019.

[6] K. Abbas, T. A. Khan, M. Afaq, and W. C. Song, "Network slice lifecycle management for 5g mobile networks: an intent-based networking approach," *IEEE Access*, 2021.

[7] S. K. Datta and C. Bonnet, "MEC and IoT based automatic agent reconfiguration in industry 4.0," in *Proceedings of the 2018 IEEE International Conference on Advanced Networks and Telecommunications Systems (ANTS)*, pp. 1–5, IEEE, Indore, India, December.2018.

[8] M. Chen and Y. Hao, "Task offloading for mobile edge computing in software defined ultra-dense network," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 3, pp. 587–597, 2018.

[9] P. W. Khan, K. Abbas, H. Shaiba, A. Muthanna, A. Abuarqoub, and M. Khayyat, "Energy efficient computation offloading mechanism in multi-server mobile edge computing-an integer linear optimization approach," *Electronics*, vol. 9, no. 6, p. 1010, 2020.

[10] R. E. Bellman and S. E. Dreyfus, *Applied dynamic programming*, Princeton university press, Princeton, NJ, USA, 2015.

[11] F. Hafiz, P. Fajri, and I. Husain, "Load regulation of a smart household with PV-storage and electric vehicle by dynamic programming successive algorithm technique," in *Proceedings of the 2016 IEEE Power and Energy Society General Meeting (PESGM)*, pp. 1–5, IEEE, Boston, MA, USA, July 2016.

[12] Y. Xiao and A. Konak, "A genetic algorithm with exact dynamic programming for the green vehicle routing & scheduling problem," *Journal of Cleaner Production*, vol. 167, pp. 1450–1463, 2017.

[13] X. Yang, H. He, and X. Zhong, "Adaptive dynamic programming for robust regulation and its application to power systems," *IEEE Transactions on Industrial Electronics*, vol. 65, pp. 5722–5732, 2017.

[14] A. Kalia and B. Fabien, "On implementing optimal energy management for EREV using distance constrained adaptive real-time dynamic programming," *Electronics*, vol. 9, no. 2, p. 228, 2020.

[15] F. Guo, H. Zhang, H. Ji, X. Li, and V. C. M. Leung, "An efficient computation offloading management scheme in the densely deployed small cell networks with mobile edge computing," *IEEE/ACM Transactions on Networking*, vol. 26, no. 6, pp. 2651–2664, 2018.

[16] S. Yu, R. Langar, X. Fu, L. Wang, and Z. Han, "Computation offloading with data caching enhancement for mobile edge computing," *IEEE Transactions on Vehicular Technology*, vol. 67, no. 11, pp. 098–112, 2018.

[17] X. Chen, L. Jiao, W. Li, and X. Fu, "Efficient multi-user computation offloading for mobile-edge cloud computing," *IEEE/ACM Transactions on Networking*, vol. 24, no. 5, pp. 2795–2808, Oct. 2016.

[18] S. Bi and Y. J. Zhang, "Computation rate maximization for wireless powered mobile-edge computing with binary computation offloading," *IEEE Transactions on Wireless Communications*, vol. 17, no. 6, pp. 4177–4190, 2018.

[19] Y. Wang, M. Sheng, X. Wang, L. Wang, and J. Li, "Mobile-edge com[1]puting: partial computation offloading using dynamic voltage scaling," *IEEE Transactions on Communications*, vol. 64, no. 10, pp. 4268–4282, 2016.

[20] A. Rafiq, W. Ping, W. Min, and M. S. A. Muthanna, "Fog assisted 6TiSCH tri-layer network architecture for adaptive scheduling and energy-efficient offloading using rank-based Q-learning in smart industries," *IEEE Sensors Journal*, vol. 21, pp. 25489–25507, 2021.

[21] A. Rafiq, W. Ping, W. Min, S. H. Hong, and N. N. Josbert, "Optimizing Energy consumption and Latency based on computation offloading and cell association in MEC enabled Industrial IoT environment," in *Proceedings of the 2021 6th International Conference on Intelligent Computing and Signal Processing (ICSP)*, pp. 10–14, IEEE, Xi'an, China, April 2021.

[22] J. Liu, Y. Mao, J. Zhang, and K. B. Letaief, "Delay-optimal computation task scheduling for mobile-edge computing systems," in *Proceedings of the 2016 IEEE International Symposium on Information Theory (ISIT)*, pp. 1451–1455, IEEE, Barcelona, Spain, July 2016.

[23] H. Shahzad and T. H. Szymanski, "A dynamic programming offloading algorithm for mobile cloud computing," in *Proceedings of the 2016 IEEE Canadian Conference on Electrical and Computer Engineering (CCECE)*, pp. 1–5, IEEE, Vancouver, Canada, May 2016.

[24] S. Mubeen, P. Nikolaidis, A. Didic, H. Pei-Breivold, K. Sandstrom, and M. Behnam, "Delay mitigation in offloaded cloud controllers in industrial IoT," *IEEE Access*, vol. 5, pp. 4418–4430, 2017.

[25] B. Liu, Q. Zhu, W. Tan, and H. Zhu, "Congestion-optimal WIFI offloading with user mobility management in smart communications," *Wireless Communications and Mobile Computing*, vol. 2018, pp. 1–15, 2018.

[26] Y. He, C. Liang, R. Yu, and Z. Han, "Trust-based social networks with computing, caching and communications: a deep reinforcement learning approach," *IEEE Transactions on Network Science and Engineering*, vol. 7, no. 1, pp. 66–79, 2018.

[27] M. Min, L. Xiao, Y. Chen, P. Cheng, D. Wu, and W. Zhuang, "Learning-based computation offloading for IoT devices with energy harvesting," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 2, pp. 1930–1941, 2019.

[28] L. N. Huynh, Q. V. Pham, X. Q. Pham, T. D. Nguyen, M. D. Hossain, and E. N. Huh, "Efficient computation offloading in multi-tier multi-access edge computing systems: a particle SwarmOptimization approach," *Applied Sciences*, vol. 10, p. 203, 2020.

[29] W. Ejaz, M. Naeem, and S. Zeadally, "On-demand sensing and wireless power transfer for self-sustainable industrial IoT networks," *IEEE Transactions on Industrial Informatics*, vol. 17, no. 10, pp. 7075–7084, 2020.

[30] A. Alshahrani, I. A. Elgendy, A. Muthanna, A. M. Alghamdi, and A. Alshamrani, "Efficient multi-player computation offloading for VR edge-cloud computing systems," *Applied Sciences*, vol. 10, no. 16, p. 5515, 2020.

[31] L. Park, C. Lee, W. Na, S. Choi, and S. Cho, "Two-stage computation offloading scheduling algorithm for energy-harvesting mobile edge computing," *Energies*, vol. 12, no. 22, p. 4367, 2019.

[32] C. You, K. Huang, and H. Chae, "Energy efficient mobile cloud computing powered by wireless energy transfer," *IEEE Journal on Selected Areas in Communications*, vol. 34, no. 5, pp. 1757–1771, 2016.

[33] S. Park, D. Kwon, J. Kim, Y. K. Lee, and S. Cho, "Adaptive real-time offloading decision-making for mobile edges: deep reinforcement learning framework and simulation results," *Applied Sciences*, vol. 10, no. 5, p. 1663, 2020.

[34] D. de Oliveira, E. Ogasawara, F. Baião, and M. Mattoso, "SciCumulus: a lightweight cloud middleware to explore many task computing paradigm in scientific workflows," in *Proceedings of the 2010 IEEE 3rd International Conference on Cloud Computing*, pp. 378–385, Miami, FL, USA, July 2010.

[35] M. Arif, G. Wang, T. Wang, and T. Peng, "SDN-based secure VANETs communication with fog computing," in *Proceedings of the International Conference on Security, Privacy and Anonymity in Computation, Communication and Storage*, pp. 46–59, Springer, Melbourne, Australia, December 2018.

[36] Z. Hu, X. Wen, Z. Lu, and W. Jing, "AORS: adaptive mobile data offloading based on attractor selection in heterogeneous wireless networks," *Wireless Networks*, vol. 23, no. 3, pp. 831–842, 2017.

[37] A. H. Lubis, A. Ikhwan, and P. L. E. Kan, "Combination of levenshtein distance and rabin-karp to improve the accuracy of document equivalence level," *International Journal of Engineering & Technology*, vol. 7, pp. 17–21, 2018.

[38] M. P. G. Team, *MATLAB Plot Gallery*, MATLAB Central File Exchange, 2020.