Computers, Materials & Continua

Tech Science Press

DOI: 10.32604/cmc.2024.051944

ARTICLE

Two-Stage IoT Computational Task Offloading Decision-Making in MEC with Request Holding and Dynamic Eviction

Dayong Wang[1,*], Kamalrulnizam Bin Abu Bakar[1] and Babangida Isyaku[2]

[1]Department of Computer Science, Faculty of Computing, Universiti Teknologi Malaysia, Johor Bahru, Johor, 81310, Malaysia

[2]Department of Computer Science, Faculty of Information Communication Technology, Sule Lamido University, Jigawa, 741103, Nigeria

*Corresponding Author: Dayong Wang. Email: wangdayong@graduate.utm.my

Received: 19 March 2024 Accepted: 20 June 2024 Published: 15 August 2024

ABSTRACT

The rapid development of Internet of Things (IoT) technology has led to a significant increase in the computational task load of Terminal Devices (TDs). TDs reduce response latency and energy consumption with the support of task-offloading in Multi-access Edge Computing (MEC). However, existing task-offloading optimization methods typically assume that MEC's computing resources are unlimited, and there is a lack of research on the optimization of task-offloading when MEC resources are exhausted. In addition, existing solutions only decide whether to accept the offloaded task request based on the single decision result of the current time slot, but lack support for multiple retry in subsequent time slots. It is resulting in TD missing potential offloading opportunities in the future. To fill this gap, we propose a Two-Stage Offloading Decision-making Framework (TSODF) with request holding and dynamic eviction. Long Short-Term Memory (LSTM)-based task-offloading request prediction and MEC resource release estimation are integrated to infer the probability of a request being accepted in the subsequent time slot. The framework learns optimized decision-making experiences continuously to increase the success rate of task offloading based on deep learning technology. Simulation results show that TSODF reduces total TD's energy consumption and delay for task execution and improves task offloading rate and system resource utilization compared to the benchmark method.

KEYWORDS

Decision making; internet of things; load prediction; task offloading; multi-access edge computing

1 Introduction

The rapid development of the Internet of Things (IoT) in recent years has resulted in insufficient Terminal Device (TD) computing capabilities [1]. To solve this issue, computing task offloading technology moves tasks to other servers for execution [2]. With the support of task offloading technology, TDs can delegate computing-intensive tasks to cloud computing platforms with sufficient resources, clout-lets in local area networks, and various edge computing nodes [3]. However, resource-rich cloud computing platforms are usually far away from TD [4], which will generate more network transmission delays [5]. In addition, traditional edge computing nodes and cloud-lets cannot support

TD mobility well. Although Mobile Cloud Computing (MCC) supports the mobility of terminal devices, it still cannot solve the problem of high network communication latency [6].

The latest development trend is to offload the computing tasks of TDs to the Multi-access Edge Computing (MEC) network. Since the network communication distance between MEC and TDs is usually only one hop, and MEC supports the mobility of TDs very well [7]. However, MEC is different from MCC, and MEC has relatively limited computing resources. Therefore, task offloading decisions and resource allocation need to be optimized [8].

A large number of existing solutions to support MEC computing decision-making and resource allocation introduce traditional mathematical algorithms, heuristic algorithms, AI-based methods, etc. [9]. However, they only directly decide whether the computing task is run locally on the TD or on the MEC based on the decision result of the current time slot. Although some algorithms consider the problem of task offloading for multiple time slots in the future, they do not consider the problem of maintaining task offloading requests to strive for more offloading opportunities in multiple time slots in the future and notify TD as early as possible to start task execution locally. In MEC under conditions of relatively insufficient resources. This way, TDs may lose task offloading opportunities in the next few time slots and cause an increase in the overall task execution delay.

In this work, we focus on optimizing the task offloading decision optimization problem of IoT computing tasks under insufficient MEC resources. To overcome the limitations of single timeslot optimization, this study adopts a combined approach utilizing task offloading load prediction and virtual decision-making for subsequent timeslots to seek additional available resources for task offloading requests. In the proposed two-stage task offloading decision framework, the Deep Q-Network (DQN)-based decision generation algorithm and the Long Short-Term Memory (LSTM)-based task request time series prediction algorithm jointly predict the possibility of each offloading request being approved in future time slots, thereby determining the optimal offloading of the offloading task. strategies and reduce overall task execution time.

- This study provides an in-depth analysis of the performance limitations of existing IoT task offloading methods in resource-constrained MEC networks and elucidates that the cause of this problem is the lack of ability to maximize the potential acceptance opportunities for offloading requests.
- Combining task offloading request prediction and online offloading decision generation to infer the probability of offloading requests being accepted.
- Jointly consider the delay-sensitivity of computational tasks and the predicted completion time to adjust strategies of offloading requests entails holding and eviction for optimizing decision-making.
- Simulation experiment results show that the proposed framework reduces the task completion time and energy consumption and improves the MEC utilization of the system.

The rest of this study is organized as follows. Section 2 discussed the related work of the IoT task offloading method in MEC. Section 3 illustrated the proposed two-stage task offloading decision framework. Section 4 presented the performance evaluation and result discussion. The conclusion was given in Section 5.

2 Related Works

As a key technique in IoT task offloading, the optimization of offloading decisions and resource allocation has been extensively studied in the past decades [10–13]. Classic task decision-making

methods based on mathematical optimization are relatively mature. Such methods can usually find the global optimal solution in a strict mathematical sense [14–16]. However, this method requires mathematical modeling for specific application scenarios, so it has poor dynamic adaptability. In addition, this type of optimization method performs poorly when dealing with complex constrained problems with high-dimensional nonlinearity. The task offloading decision-making method based on game theory allows multiple participants to negotiate, and this method is suitable for adversarial task offloading and resource allocation scenarios [17,18]. This method is usually more conducive to ensuring that all parties involved receive relatively fair benefits. In addition, task offloading solutions based on game theory can better avoid system bottlenecks and single-point failures. However, such methods lead to a rapid increase in computational complexity when the number of participants is large, resulting in solution difficulties. In addition, this type of method may have difficulty converging in a dynamic system environment, and the game equilibrium state may be unstable. The task offloading optimization method based on fuzzy theory can more comprehensively consider multiple factors that affect decision-making effects and can use simpler models to describe complex real-life problems, thereby reducing the complexity of problem modeling. However, the performance of such methods is usually poor and the system output results are not intuitive. Based on heuristic methods, we do not seek the absolute optimal solution but seek relatively better solutions [19,20]. This method is cost-effective and suitable for solving large-scale problems. However, such methods are prone to falling into local optimal problems.

In order to cope with large-scale and dynamic complex computing task offloading application scenarios, various decision-making optimization methods based on AI technology have emerged in large numbers in the past decade [21,22]. This method has self-learning capabilities and can continuously optimize the model itself based on historical data and experience. In addition, AI-based task offloading decision-making methods can usually discover complex relationships hidden in high-dimensional data, so it is easy to find the global optimal solution. Moreover, this type of method can better adapt to dynamic changes in the network environment [23,24]. However, the methods discussed above lack the prediction of the offloading environment status for multiple time slots.

Besides, some researchers consider prediction with the computational load on the edge server. However, such studies mainly focus on the prediction of load on computing nodes but ignore the prediction of characteristics of task offloading requests from TD [25,26]. In addition, the AI models selected in a small number of studies can generate predictions of task scheduling decisions for multiple time slots in the future. However, no consideration is given to finding as many offloading opportunities as possible for offloading tasks in multiple time slots. We selected representative research works of different technical classifications for analysis, which cover common binary and partial task offloading optimization methods. The focus of the study is to analyze the start time of local execution tasks after the offload request is rejected. Table 1 shows the characteristics of representative task offloading algorithms to initiate local task execution.

Table 1: Comparison of task offloading optimization algorithm characteristics

Ref.	Year	Type	Method	Mode	Paradigm	Response to offloading decisions
[27]	2024	AI	Meta reinforcement learning	PO	MEC/IOT	Starting local task execution after an offloading request has been rejected

(Continued)

Table 1 (continued)

Ref.	Year	Type	Method	Mode	Paradigm	Response to offloading decisions
[28]	2023	AI	Q-learning	PO	MEC/IIOT	Starting local task execution after an offloading request has been rejected
[29]	2023	AI	DDPG	PO	MEC/IIOT	Starting local task execution after an offloading request has been rejected
[30]	2021	Heuristic	GA	BO	MEC/IOT	Starting local task execution after an offloading request has been rejected
[31]	2023	Heuristic	NSGA-III	PO	MEC/IOT	Starting local task execution after an offloading request has been rejected
[32]	2021	Lyapunov	Lyapunov-guided DRL	BO	Multi-user MEC	Starting local task execution after an offloading request has been rejected
[33]	2022	Lyapunov	Classic lyapunov	PO	MEC/IOT	Starting local task execution after an offloading request has been rejected
[34]	2021	Game theory	Coalitional game-based	BO	MEC/IOT	Retry in the next decision cycle
[35]	2023	Classic	Mathematical optimization	BO	MEC/UAV	Starting local task execution after an offloading request has been rejected

Note: Abbreviations: BO, Binary offloading; PO, Partial offloading.

The available resources of the MEC change dynamically due to the dynamics of task offloading requests and the completion of the running of computing tasks on the MEC. According to the review of literature, the majority of decision mechanisms only rely on the current slot's task offloading network condition to make the final offloading decision. This results in rejected tasks immediately starting execution locally on the TD, missing the opportunity to seek edge computing resources in more time slots. We illustrate this issue with Fig. 1, which represents a common problem widely observed in similar studies [28,33,35,36].

It can be found in Fig. 1 that appropriately retrying to obtain the opportunity of task offloading in several adjacent decision cycles will be beneficial to reducing the completion time of the task after the first task offloading application is rejected. However, starting local task execution prematurely and excessive offload retries will increase task completion time.

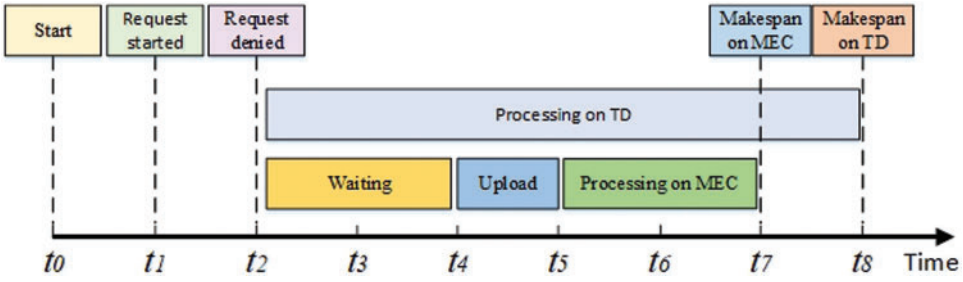

Figure 1: Completion time of different task execution locations

3 Materials and Methods

In this section, the Two-Stage Task Offloading Decision Framework (TSODF) is proposed to allow offloading decision attempts across multiple time slots. In this method, best-effort decision optimization is performed on offloading requests and task allocation in each slot based on Deep Q-learning. In addition, the historical TD task offloading requests are input into the prediction model based on SLTM in time series to predict the task request load of multiple time slots in the future. Furthermore, the offloading decision for pending requests in future time slots is reasoned by combining the inferred MEC available computing resources and predicted TD task offloading requests. This enables the assessment of the cost and benefits of pending task requests. Subsequently, optimized decisions with higher request acceptance rates are output to reduce task execution latency and energy consumption.

We consider a MEC system, where M IoT TDs and an EMC server. Let $D = \{d_1, d_2, \ldots, d_m\}$, denote the sets of the TDs. The MEC server S has powerful computing power C_s and unlimited battery life. We assume that every computational task is indivisible and let $T = \{t_1, t_2, \ldots, t_n\}$. Each TD can choose to execute tasks locally or offload computing tasks to MEC for execution. Fig. 2 depicts the task offloading system model targeted by TSODF, referencing a typical MEC network architecture [37].

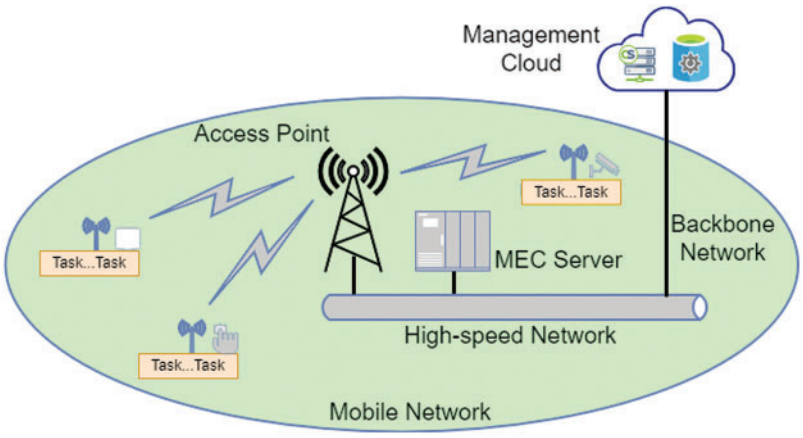

Figure 2: The system architecture of task offloading in MEC

In the proposed framework, the DQN-based decision-making model performs best-effort optimized task offloading decisions for each time slot. Based on the allocated MEC server resource

records, TSODF estimates the CPU resources available in each future time slot. The LSTM-based prediction module predicts task offloading requests in several future time slots based on historical TDs task offloading request records. TSODF jointly analyzes the above information to determine whether to continue to look for offloading opportunities for waiting task offloading requests, and outputs the optimized final decision result. Fig. 3 shows the proposed system framework of TSODF.

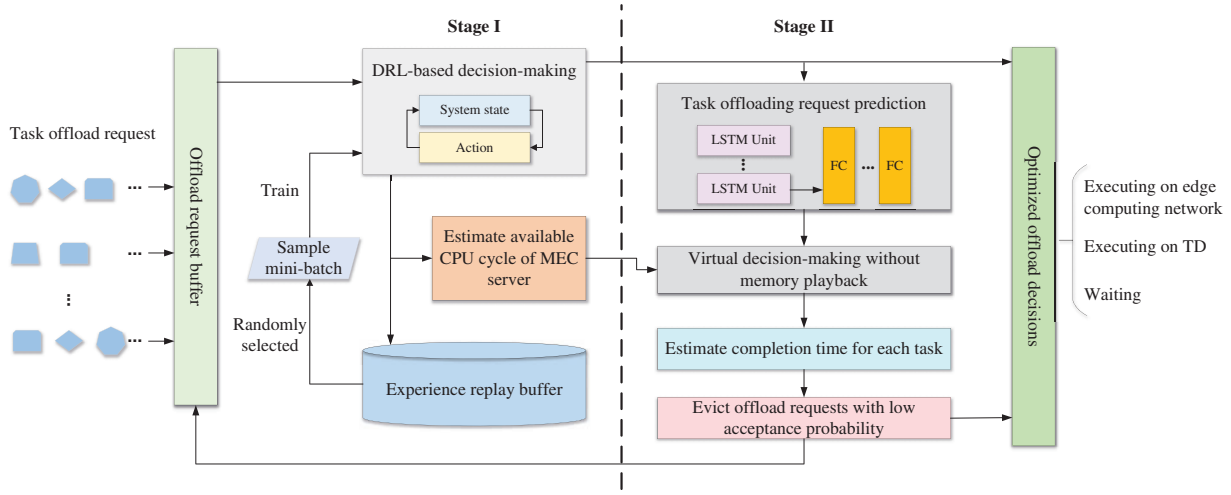

Figure 3: The system model of TSODF

TSODF contains two main work stages, in which stage 1 performs the generation of optimal offloading decisions in each decision cycle based on deep reinforcement learning; stage 2 performs task offloading request load prediction based on LSTM. In addition, TSODF estimates the available resources of the future MEC server based on the decision results that have been generated. Thus, TSODF infers the probability that a task offloading request will be accepted in the future for offloading tasks that have not yet been accepted. Based on this mechanism, task offloading requests will be suspended waiting for an upcoming acceptance opportunity or starting local execution immediately.

The DQN-based but time-slot offloading decision generation method is discussed in Section 3.1. Section 3.2 introduces the task offloading request prediction method based on LSTM. The complete TSODF framework workflow is given in Section 3.3.

3.1 Single Time-Slot Offloading Decision Based on DRL

In order to make task offloading request decisions for each time slot based on available MEC resources, we build an adaptation algorithm based on DRL technology. Different from previous research, the DRL-based decision-making algorithm constructed needs to run synchronously with other modules on a global scale.

The system state in each time slots t can be presented as follows:

$$s_t = \{C_S(t), Load_D(t), QueueLength_T(t), T_D(t)\}, \tag{1}$$

where $C_S(t)$, $Load_D(t)$, $QueueLength_T(t)$ and $T_D(t)$ symbolize the current load of the server, the load status of the TD, the length of the task queue, and the type of task, respectively. In addition, more content can be added to the state space in different scenarios without affecting the working logic of the

proposed framework. The task offloading decision and the computing resource allocation constitute the action space at epoch time slot t can be described as:

$$a_t = \left\{X_{i,j}(t)\right\}, \tag{2}$$

where a_t is the task offloading decisions. In addition, the object is to minimize the joint cost and the system reword. Consider system states as follows:

$$E_{exec,i,j} = \frac{C_i}{C_j} \times P_j, \tag{3}$$

where $E_{exec,i,j}$ represents the energy consumption of task execution, C_j is the computing power of device d_j. The transfer time of a task is calculated as follows:

$$T_{i,j} = \frac{W_i}{B}, \tag{4}$$

where W_i and B represent the data size of the task and network bandwidth, respectively. The energy consumption of TD for network transmission task data is calculated as follows:

$$E_{trans,i,j} = \left(T_{up,i,j} + T_{down,i,j}\right) \times P_{trans}, \tag{5}$$

where T and P represent transmission time and RF power consumption, respectively. This way, the total energy consumption and latency can be calculated as follows:

$$E_{total} = \sum\nolimits_{i \in T} \sum\nolimits_{j \in D} \left(E_{exec,i,j} + E_{trans,i,j}\right) \times X_{i,j}, \tag{6}$$

$$L_i = E_{i,j} + T_{up,i,j} + T_{down,i,j}, \tag{7}$$

We consider energy consumption and latency together to form the reward as follow of reinforcement learning:

$$R = \alpha \cdot E_{total} + \beta \cdot L_i, \tag{8}$$

where α and β are weight factors used to balance the importance of different rewards.

Through the reinforcement learning process, the optimal transfer strategy will be found to obtain the relatively optimal solution of the offloading decision for each time slot i. As Algorithm 1 shows, DQN-based decision generation and resource allocation work in cycles for each time slot, which is composed of three main parts:

1) Randomly initialize the deep neural network and generate offloading decisions and resource allocations for continuously incoming task offloading requests (lines 1–10).
2) Execute the offload decisions generated by the preceding steps and collect updated system status. Meanwhile, the rewards for offloading decisions are calculated based on the accumulated delay and energy consumption (lines 11–13).
3) Cache continuously generated decisions, system state, and rewards. Records from the cache are randomly selected to train and update the weights of the neural network. Thus, the decision-making capabilities of algorithms will continue to improve (lines 14–19).

Algorithm 1: DQN Based Task Offloading Decision-Making and Resource Allocation

Input: state $\Phi(i)$, action $\Theta(i)$
Output: offload decision for time slot i

(Continued)

Algorithm 1 (continued)

Initialize: set the evaluation and the target network with random weight θ_e and θ_t
for episode =1, M **do**
  for $i = 1, I$ **do**
    Observe the state $\Phi(i)$
    Random value τ between 0 and 1
    If $\tau < \varepsilon$ **then**
      Choose an action at random
    else
      Select $\Theta(i) = argmax_{\Theta} Q(\Phi(i), \Theta(i), \theta_e)$
    end if
    Perform action $\Theta(i)$
    Observe the next step state $\Phi(i+1)$
    Realize R
    Store $\Phi(i), \Theta(i), r(i), \Phi(i+1)$ in experience memory E
    Randomly select sample $\left(\Phi_j, \Theta_j, r_j, \Phi_{j+1}\right)$ from E and calculate TD target
    Update Q and θ_e
    Execute a gradient decent step on $Loss(\theta_e)$
    if $i == C$ **then**
      Update the target network weight parameter with $\theta_t = \theta_e$
    end if
  end for
end for

3.2 IoT Task Offloading Request Prediction

In order to predict the task offloading request load from TD in each subsequent time slot, we build a task offloading request sequence prediction algorithm based on LSTM. The prediction includes the number, size, CPU requirements and deadline of subsequent tasks that need to be offloaded. This algorithm provides a calculation basis for the decision-making optimization in the second stage. Let H(t) denote the offloading task request time series composed of tasks $\Gamma_{n,k}(i)$, and input it into the LSTM-based prediction model on a rolling basis. In order to output the predicted values of multiple attributes for the offloading task, we added three fully connected layers to the output of the LSTM network. This way, the prediction mode will output $\Gamma_{n,k}(i+1, i+2, \ldots, i+\varpi)$, where ϖ is half the slot required for the longest task execution in history. Fig. 3 presents the structure of the task offloading request prediction model. As shown in Algorithm 2, its implementation includes two steps:

1) Task offloading requests from TDs are continuously collected and saved in order, thereby generating time series data for training the LSTM-based prediction model network. The collected task request characteristics include type, size, deadline, etc. (lines 1–7).
2) Predict task offloading requests for subsequent time slots based on the trained LSTM model. The predicted output value for each round contains the task characteristics of offloading requests for multiple consecutive time slots. The prediction is moved forward continuously through the decision-making cycle (lines 8–12).

Algorithm 2: LSTM Based Task Offloading Request Prediction

Input: Historical request sequence $H(i)$
Output: Predicted TD task offloading request sequence $Q_{i+\omega}$

```
Collect task offloading requests from TDs
Split collected request data into train and test sets
Connect input units, LSTM unit and three fully connected output units
Set up optimizer
    for epochs do
        Train the LSTM network
    end for
    for i = 1, I do
        Input H(i) into the LSTM network
        Make predictions
        Return Q_{i+ω}
    end for
```

3.3 Offload Decision Optimization with Offload Request Retention and Eviction

It is necessary to judge the possibility of each waiting request being accepted in multiple time slots in order to find as many offloading opportunities as possible for tasks from TD within the appropriate range. The completion time of the task will be shortened if the offloaded task can be executed in the MEC after waiting for several time slots. In contrast, TD should be notified as early as possible to start local task execution when no offloading opportunity can be found for the task in the future. This way, it's needed to combine available MEC resources, predicted results of task offloading requests, and the latest offloading decision results to generate optimized offloading decisions $D^*_{n,k}$. Fig. 4 shows the working principle of the TSODF, and Algorithm 3 shows the specific workflow.

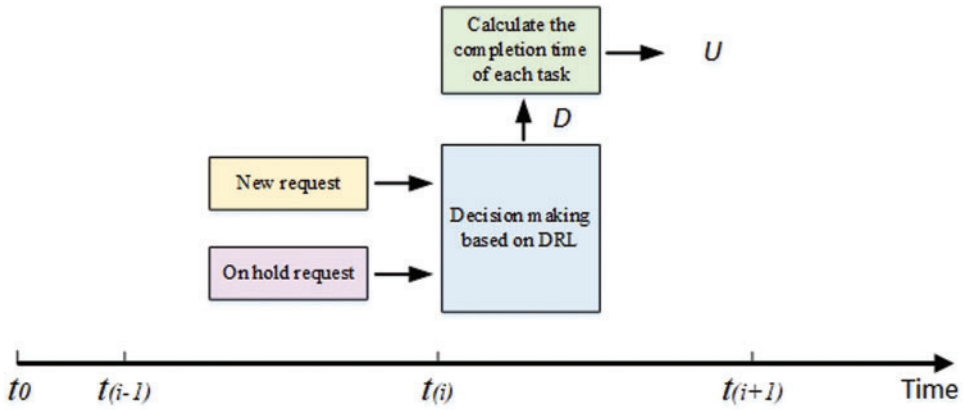

Figure 4: The working principle of TSODF

The main idea of TSODF is to find the possibility of being accepted for a currently unaccepted task offloading request in future time slots based on the estimation of the release of available resources of the MEC server and the prediction of subsequent task offloading requests from TD. The LSTM neural network in the framework is responsible for predicting future task offloading request sequences to understand in advance the load that the MEC server will bear. The DQN-based task offloading decision making module is responsible for generating the optimal offloading decision

for the current time slot. In addition, the DQN module is also responsible for generating future-oriented virtual offloading decisions based on predicted task requests and service resource availability of countermeasures. Thus, TSODF decides whether the task should be wait to be accepted. As shown in Algorithm 3, its implementation includes four steps:

1) Call Algorithm 1 to generate the local optimal offloading decision and resource allocation for the task offloading request in the current time slot (lines 1–5).
2) Call Algorithm 2 to predict the task offloading request characteristics of subsequent time slots, and the expected resource consumptions of the edge server are estimated based on the prediction results (lines 6–7).
3) The joint predicted task offload request load and the current operating state of the edge server invoke Algorithm 1 to generate offload decisions and resource allocations for future virtual time slots (lines 8–11).
4) The possibility of the task offloading request being accepted in subsequent time slots is speculated based on the inference results. Thus, the holding and eviction decision of the task offloading request is generated (lines 12–15).

Algorithm 3: Offloading Request Holding and Eviction

Input: MEC CPU cycle per time slot F^{total}, bandwidth B, task required CPU cycle $c_{n,k}$, task types K, task offloading request $\Gamma_{n,k}(i)$

Output: optimized offloading decision $\mathrm{D}^{*}_{n,k}$

1. **for** timeslot i in I **do**
2. Calculate the number of timeslots ϖ for the average task processing time
3. Call algorithm 1 to generate offloading decisions $\mathrm{D}_{n,k}$ for the time slot i
4. **if** $D_t \neq 1$ **do**
5. $\omega = \varpi/2$
6. Call algorithm 2 to predict task offloading requests from $\Gamma^{i}_{n,k}$ to $\Gamma^{i+\omega}_{n,k}$
7. $F^{free}_{i} = F^{total} - \sum_{i}^{i+\omega} f^{mec}_{n,k}$
8. Call algorithm 1 to generate offloading decisions $\mathrm{D}_{n,k}$ for the time slot $i+\omega$
9. **for** t =1, ω **do**
10. **if** $d^{t}_{n,k}$ == true **do**
11. Calculate the completion time for $\Gamma_{n,k}$
12. **if** $\tau_{n,k} \geq L_{n,k}$ **do**
13. Evict task offloading requests from buffer
14. **else**
15. Keep holding
16. **end if**
17. **end if**
18. **end for**
19. **end if**
20. **end for**

In Algorithm 3, line 1 constructs a time slot-based algorithm to continuously run the main loop (line 1). Calls Algorithm 1 to generate the optimal task offloading decision for the current time slot (lines 2 and 3). Temporarily holds currently unaccepted task offloading requests and calls Algorithm 2 to predict future task offloading request trends (lines 4–6). Estimates the available resources of the

future MEC server based on known task offloading decisions (line 7). Calls Algorithm 1 to virtually generate offloading decisions for the next decision cycle (line 8). Determines the possibility that the offload request in the holding state will be accepted in multiple time slots in the future based on the prediction results of the previous steps (lines 9–15). Unloading requests for tasks that are about to get an unloading opportunity continue to be maintained. On the contrary, local execution is started for tasks that have no chance of offloading in the short term.

4 Results and Discussion

In this section, numerical results are presented to evaluate the performance of the proposed TSODF framework. Simulation environment information is given in Section 4.1. The results of the experiments and data analysis are discussed in Section 4.2.

4.1 Simulation Settings

To assess the evaluation of IoT task offloading in MEC, we adopt an MEC computing network, where there is an MEC server with an access point (AP) which covers a range of 50 m. The wireless channel bandwidth is set to 40 MHz, and N = 10 TDs are randomly distributed in the coverage of the wireless AP. The computing capacity of a TD is set to 0.8 GHz. The total CPU cycle of MEC server is set to 4 GHz. The transmit power of a TD is set as 100 mW and the background noise SR_n is −100 dBm. The range of task size is set from 300 to 1024 KB. The deadline for each task is randomly generated from 0.1 to 2 s. The simulation environment was built on the Windows 10 operating system, and Python 3.10 was selected as the program running environment. The computer hardware comes with inter I7 CPU, and 16 GB RAM. Table 2 lists the key parameters in the simulation.

Table 2: Parameter settings in the simulation

Parameters	Value and unit	Parameters	Value and unit
Number of TDs	10	Transmission power of TDn	100 mW
Total number of task types	5	Background noise power	−100 dBm
Probability of type-k task requested	$0 < \mathcal{P}_{i,j} < 1$	Data size of task	300–1024 KB
Available CPU cycle of TDn	0.8 GHz	CPU requirement to process	100–10000 Megacycles
Computing capability of MEC	4 GHz	Deadline of task	0.1–2 s

4.2 Analysis of Results

The task offloading rate is one of the main indicators used to measure the task offloading mechanism. It is the ratio of offloaded tasks to the total task volume [38]. The average task completion time is an important indicator for evaluating the task offloading mechanism and is used to demonstrate the time saving benefits of task offloading [39]. The average energy consumption of task completion is used to measure the energy saving benefit of task offloading [40].

In order to verify the TSODA, we compare the task offloading rate with classical DQN-based decision making method.

- TD only (TO). The TO solution denotes that all IoT tasks will be processed on TDs.
- MEC only (MO). The MO solution denotes that all IoT tasks will be processed on the MEC server. In addition, the computing resources of the MEC server will be evenly allocated to each IoT task.
- Classic DQN (Classic-DQN). IoT tasks will be distributed and processed between the MEC server and TDs based on a pure DQN-based offloading decision-making method.
- Two-Stage Task Offloading Decision Framework (TSODF). The TSODF solution denotes that all IoT tasks will be processed according to the decisions given by the two-stage task offloading decision framework.

Fig. 5a shows the comparison between 4 schemes. It can be observed that the convergence time of TSODF with added task offloading prediction is not much different from the classic DQN scheme. However, the delay of tasks scheduled by the TSODF algorithm is significantly improved compared to the classic DQN method. Fig. 5b shows the latency differences between various offloading decision algorithms for task offloading requests with different task sizes. It can be seen that when the total amount of offloading task requests is greater than the upper limit of the computing power of the MEC server, the delay of the classic DQN method is close to that of the MEC-only method. However, the delay of the TSODF method stabilizes at a relatively low level for a long time.

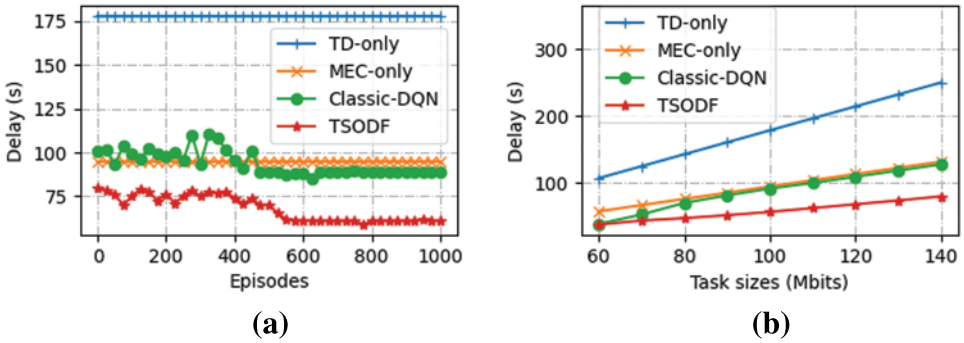

Figure 5: (a) Performance of different algorithms under total task sizes D = 100 Mb. (b) Delay with different total task sizes

In this study, the energy consumption of TD is the most focused indicator and the energy consumption of the MEC server is not calculated due to the limited battery capacity of TD. The TSODF algorithm allocates more computing tasks to the MEC server for execution, thereby reducing the energy consumption of TD.

Fig. 6a presents the differences in TD energy consumption of various task offloading methods. In addition, Fig. 6b shows the difference in utilization of MEC server computing resources by each method. In the MEC network environment, the offloading rate of IoT computing tasks offloading refers to the ratio of offloading computing tasks from the local execution of IoT devices to edge computing nodes. It is one of the important indicators to measure the algorithm since each method always offloads as many tasks as possible from TD to the MEC network. Fig. 7a,b respectively show the difference between the offloading rates of the task offloading decision-making methods under different task sizes and different TD numbers.

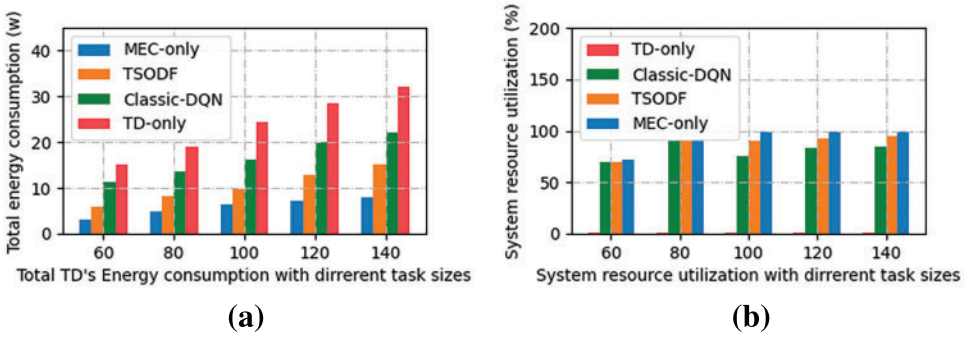

Figure 6: (a) Total TD's energy consumption. (b) System resource utilization

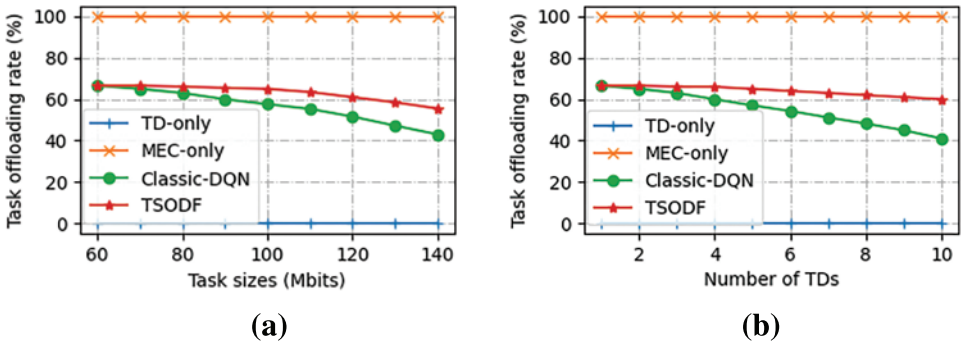

Figure 7: (a) Task offloading rate with different total task sizes. (b) Task offloading rate with different numbers of TDs varies from 1 to 10

5 Conclusion

In this work, the optimization of IoT task offloading decisions in MEC is studied in depth. Different from previous work, the proposed Two-Stage Task Offloading Decision Framework (TSODF) considers the continuous optimization of multiple attempts for unapproved offload requests when MEC resources are insufficient. The TSODF is designed to explore available edge computing network resources to accept task offloading requests from TDs. In this way, the success rate of task offloading and the utilization of edge computing resources are increased, resulting in overall reductions in task completion time and energy consumption. In addition, the request eviction reduces the delay for local task execution in waiting for decision-making. Simulation results show that the performance of the proposed task offloading framework improves the overall task offloading rate and reduces the overall task execution latency and energy consumption compared to the classical DQN-based task offloading decision scheme. Although only the case of DQN-based offloading decision generation techniques is considered in this paper, the proposed framework can be easily extended to the scenarios with the different types of offloading decision generation techniques. For future work, we will investigate optimization methods for IoT task offloading decisions in MEC environments supporting scalable IoT networks.

Acknowledgement: We thank Universiti Teknologi Malaysia (UTM) for supporting us during this work.

Funding Statement: The authors received no specific funding for this study.

Author Contributions: The authors confirm contribution to the paper as follows: Dayong Wang contributed to Conceptualization and writing-original draft preparation, Babangida Isyaku and Dayong Wang contributed to methodology, Babangida Isyaku and Kamalrulnizam Bin Abu Bakar contributed to writing-review and editing, and Kamalrulnizam Bin Abu Bakar supervised the process of the research. All authors reviewed the results and approved the final version of the manuscript.

Availability of Data and Materials: Not applicable.

Conflicts of Interest: The authors declare that they have no conflicts of interest to report regarding the present study.

References

[1] K. B. A. Bakar, F. T. Zuhra, B. Isyaku, and S. B. Sulaiman, "A review on the immediate advancement of the internet of things in wireless telecommunications," *IEEE Access*, vol. 11, no. 70, pp. 21020–21048, 2023. doi: 10.1109/ACCESS.2023.3250466.

[2] X. Jin, W. Hua, Z. Wang, and Y. Chen, "A survey of research on computation offloading in mobile cloud computing," *Wirel. Netw.*, vol. 28, no. 4, pp. 1563–1585, May 2022. doi: 10.1007/s11276-022-02920-2.

[3] K. Sadatdiynov, L. Cui, L. Zhang, J. Z. Huang, S. Salloum and M. S. Mahmud, "A review of optimization methods for computation offloading in edge computing networks," *Digit. Commun. Netw.*, vol. 9, no. 2, pp. 450–461, Apr. 2023. doi: 10.1016/j.dcan.2022.03.003.

[4] B. Isyaku, K. B. Abu Bakar, F. A. Ghaleb, and S. Sulaiman, "Performance evaluation of flowtable eviction mechanisms for software defined networks considering traffic flows variabilities," in *2022 IEEE 12th Symp. Comput. Appl. Ind. Electron. (ISCAIE)*, May 2022, pp. 71–75. doi: 10.1109/ISCAIE54458.2022.9794547.

[5] S. Zhou, W. Jadoon, and I. A. Khan, "Computing offloading strategy in mobile edge computing environment: A comparison between adopted frameworks, challenges, and future directions," *Electronics*, vol. 12, no. 11, pp. 2452, Jan. 2023. doi: 10.3390/electronics12112452.

[6] S. A. Abdulzahra, A. K. M. Al-Qurabat, and A. K. Idrees, "Compression-based data reduction technique for IoT sensor networks," *Baghdad Sci. J.*, vol. 18, no. 1, pp. 0184, 2021. doi: 10.21123/bsj.2021.18.1.0184.

[7] H. Jin, M. A. Gregory, and S. Li, "A review of intelligent computation offloading in multiaccess edge Computing," *IEEE Access*, vol. 10, no. 3, pp. 71481–71495, 2022. doi: 10.1109/ACCESS.2022.3187701.

[8] M. Maray and J. Shuja, "Computation offloading in mobile cloud computing and mobile edge computing: Survey, taxonomy, and open issues," *Mob. Inf. Syst.*, vol. 2022, no. 3, pp. e1121822, 2022. doi: 10.1155/2022/1121822.

[9] B. Kar, W. Yahya, Y. D. Lin, and A. Ali, "Offloading using traditional optimization and machine learning in federated cloud-edge–fog systems: A survey," *IEEE Commun. Surv. Tutor.*, vol. 25, no. 2, pp. 1199–1226, 2023. doi: 10.1109/COMST.2023.3239579.

[10] K. Sadatdiynov, L. Cui, L. Zhang, J. Z. Huang, N. N. Xiong and C. Luo, "An intelligent hybrid method: Multi-objective optimization for MEC-enabled devices of IoE," *J. Parallel Distr. Comput.*, vol. 171, pp. 1–13, Jan. 2023. doi: 10.1016/j.jpdc.2022.09.008.

[11] M. Maray, E. Mustafa, and J. Shuja, "Wireless power assisted computation offloading in mobile edge computing: A deep reinforcement learning approach," *Hum.-Centric Comput. Inf. Sci.*, vol. 14, pp. 22, 2024. doi: 10.22967/HCIS.2024.14.022.

[12] M. Ahmed *et al.*, "A survey on vehicular task offloading: Classification, issues, and challenges," *J. King Saud Univ.-Comput. Inf. Sci.*, vol. 34, no. 7, pp. 4135–4162, May 2022. doi: 10.1016/j.jksuci.2022.05.016.

[13] M. Maray, E. Mustafa, J. Shuja, and M. Bilal, "Dependent task offloading with deadline-aware scheduling in mobile edge networks," *Internet of Things*, vol. 23, no. 1, pp. 100868, Oct. 2023. doi: 10.1016/j.iot.2023.100868.

[14] Z. Yu, Y. Gong, S. Gong, and Y. Guo, "Joint task offloading and resource allocation in UAV-enabled mobile edge computing," *IEEE Internet Things J.*, vol. 7, no. 4, pp. 3147–3159, Apr. 2020. doi: 10.1109/JIOT.2020.2965898.

[15] I. A. Elgendy, W. Z. Zhang, Y. Zeng, H. He, Y. C. Tian and Y. Yang, "Efficient and secure multi-user multi-task computation offloading for mobile-edge computing in mobile IoT networks," *IEEE Trans. Netw. Serv. Manag.*, vol. 17, no. 4, pp. 2410–2422, Dec. 2020. doi: 10.1109/TNSM.2020.3020249.

[16] L. Ma, X. Wang, X. Wang, L. Wang, Y. Shi and M. Huang, "TCDA: Truthful combinatorial double auctions for mobile edge computing in industrial internet of things," *IEEE Trans. Mob. Comput.*, vol. 21, no. 11, pp. 4125–4138, Nov. 2022. doi: 10.1109/TMC.2021.3064314.

[17] Y. Chen, J. Zhao, X. Zhou, L. Qi, X. Xu and J. Huang, "A distributed game theoretical approach for credibility-guaranteed multimedia data offloading in MEC," *Inf. Sci.*, vol. 644, no. 11, pp. 119306, Oct. 2023. doi: 10.1016/j.ins.2023.119306.

[18] J. Huang, M. Wang, Y. Wu, Y. Chen, and X. Shen, "Distributed offloading in overlapping areas of mobile-edge computing for internet of things," *IEEE Internet Things J.*, vol. 9, no. 15, pp. 13837–13847, Aug. 2022. doi: 10.1109/JIOT.2022.3143539.

[19] H. Hao, J. Zhang, and Q. Gu, "Optimal IoT service offloading with uncertainty in sdn-based mobile edge computing," *Mobile Netw. Appl.*, vol. 27, no. 6, pp. 2318–2327, Dec. 2022. doi: 10.1007/s11036-021-01796-4.

[20] T. Zhou, Y. Yue, D. Qin, X. Nie, X. Li and C. Li, "Joint device association, resource allocation, and computation offloading in ultradense multidevice and multitask IoT networks," *IEEE Internet Things J.*, vol. 9, no. 19, pp. 18695–18709, Oct. 2022. doi: 10.1109/JIOT.2022.3161670.

[21] J. Xu, B. Ai, L. Chen, Y. Cui, and N. Wang, "Deep reinforcement learning for computation and communication resource allocation in multiaccess MEC assisted railway IoT networks," *IEEE Trans. Intell. Transp. Syst.*, vol. 23, no. 12, pp. 23797–23808, Dec. 2022. doi: 10.1109/TITS.2022.3205175.

[22] H. Zhou, M. Li, N. Wang, G. Min, and J. Wu, "Accelerating deep learning inference via model parallelism and partial comSputation offloading," *IEEE Trans. Parallel Distrib. Syst.*, vol. 34, no. 2, pp. 475–488, Feb. 2023. doi: 10.1109/TPDS.2022.3222509.

[23] A. Acheampong, Y. Zhang, and X. Xu, "A parallel computing based model for online binary computation offloading in mobile edge computing," *Comput. Commun.*, vol. 203, no. 4, pp. 248–261, Apr. 2023. doi: 10.1016/j.comcom.2023.03.004.

[24] S. Iftikhar *et al.*, "HunterPlus: AI based energy-efficient task scheduling for cloud-fog computing environments," *Internet of Things*, vol. 21, no. 2, pp. 100667, Apr. 2023. doi: 10.1016/j.iot.2022.100667.

[25] M. Tang and V. W. S. Wong, "Deep reinforcement learning for task offloading in mobile edge computing systems," *IEEE Trans. Mob. Comput.*, vol. 21, no. 6, pp. 1985–1997, Jun. 2022. doi: 10.1109/TMC.2020.3036871.

[26] S. Tuli, S. Ilager, K. Ramamohanarao, and R. Buyya, "Dynamic scheduling for stochastic edge-cloud computing environments using A3C learning and residual recurrent neural networks," *IEEE Trans. Mob. Comput.*, vol. 21, no. 3, pp. 940–954, Mar. 2022. doi: 10.1109/TMC.2020.3017079.

[27] M. A. Hossain, W. Liu, and N. Ansari, "Computation-efficient offloading and power control for MEC in IoT networks by meta reinforcement learning," *IEEE Internet Things J.*, vol. 11, no. 9, pp. 16722–16730, 2024. doi: 10.1109/JIOT.2024.3355023.

[28] S. Wu, H. Xue, and L. Zhang, "Q-learning-aided offloading strategy in edge-assisted federated learning over industrial IoT," *Electronics*, vol. 12, no. 7, pp. 1706, Jan. 2023. doi: 10.3390/electronics12071706.

[29] H. Mai Do, T. P. Tran, and M. Yoo, "Deep reinforcement learning-based task offloading and resource allocation for industrial IoT in MEC federation system," *IEEE Access*, vol. 11, pp. 83150–83170, 2023. doi: 10.1109/ACCESS.2023.3302518.

[30] C. Li, Q. Cai, C. Zhang, B. Ma, and Y. Luo, "Computation offloading and service allocation in mobile edge computing," *J. Supercomput.*, vol. 77, no. 12, pp. 13933–13962, Dec. 2021. doi: 10.1007/s11227-021-03749-w.

[31] K. Sadatdiynov, L. Cui, and J. Z. Huang, "Offloading dependent tasks in MEC-enabled IoT systems: A preference-based hybrid optimization method," *Peer-to-Peer Netw. Appl.*, vol. 16, no. 2, pp. 657–674, Mar. 2023. doi: 10.1007/s12083-022-01435-z.

[32] S. Bi, L. Huang, H. Wang, and Y. J. A. Zhang, "Lyapunov-guided deep reinforcement learning for stable online computation offloading in mobile-edge computing networks," *IEEE Trans. Wirel. Commun.*, vol. 20, no. 11, pp. 7519–7537, Nov. 2021. doi: 10.1109/TWC.2021.3085319.

[33] H. Hu, W. Song, Q. Wang, R. Q. Hu, and H. Zhu, "Energy efficiency and delay tradeoff in an MEC-enabled mobile IoT network," *IEEE Internet Things J.*, vol. 9, no. 17, pp. 15942–15956, Sep. 2022. doi: 10.1109/JIOT.2022.3153847.

[34] X. Yang, H. Luo, Y. Sun, J. Zou, and M. Guizani, "Coalitional game-based cooperative computation offloading in MEC for reusable tasks," *IEEE Internet Things J.*, vol. 8, no. 16, pp. 12968–12982, Aug. 2021. doi: 10.1109/JIOT.2021.3064186.

[35] H. Wang, H. Xu, H. Huang, M. Chen, and S. Chen, "Robust task offloading in dynamic edge computing," *IEEE Trans. Mob. Comput.*, vol. 22, no. 1, pp. 500–514, Jan. 2023. doi: 10.1109/TMC.2021.3068748.

[36] M. Y. Akhlaqi and Z. B. Mohd Hanapi, "Task offloading paradigm in mobile edge computing-current issues, adopted approaches, and future directions," *J. Netw. Comput. Appl.*, vol. 212, no. 10, pp. 103568, Mar. 2023. doi: 10.1016/j.jnca.2022.103568.

[37] B. Liang, M. A. Gregory, and S. Li, "Multi-access edge computing fundamentals, services, enablers and challenges: A complete survey," *J. Netw. Comput. Appl.*, vol. 199, no. 1, pp. 103308, Mar. 2022. doi: 10.1016/j.jnca.2021.103308.

[38] A. Shirke and M. M. Chandane, "Collaborative offloading decision policy framework in IoT using edge computing," *Multimed. Tools Appl.*, Jan. 2023. doi: 10.1007/s11042-023-14383-4.

[39] D. Wang, W. Wang, H. Gao, Z. Zhang, and Z. Han, "Delay-optimal computation offloading in large-Scale Multi-access edge computing using mean field game," *IEEE Trans. Wirel. Commun.*, 2023. doi: 10.1109/TWC.2023.3344229.

[40] L. Cui *et al.*, "Joint optimization of energy consumption and latency in mobile edge computing for internet of things," *IEEE Internet Things J.*, vol. 6, no. 3, pp. 4791–4803, Jun. 2019. doi: 10.1109/JIOT.2018.2869226.