ISSN 2409-2665
Journal of Logistics, Informatics and Service Science
Vol. 12 (2025) No. 4, pp. 295-315
DOI:10.33168/JLISS.2025.0416

Construction of MEC Task Offloading Strategy and Risk Assessment Model Based on Multi-objective Optimization

Yong Wang[1,2*], William P. Rey[1*]

[1]School of Information Technology, Mapúa University, Manila, Philippines

[2]School of Information Engineering, Yulin University, Yulin 719000, Shaanxi, China

wyong@mymail.mapua.edu.ph; wprey@mapua.edu.ph (Co-Corresponding author)

Abstract: Traditional mobile edge computing (MEC) task offloading often faces the challenge of balancing latency, energy consumption, and reliability. This study proposes a novel multi-objective optimization framework that combines a long short-term memory (LSTM) network with a Markov decision process (MDP) to achieve intelligent task offloading decisions. The LSTM component processes historical network and resource data to predict the optimal offloading strategy, while the MDP models the decision-making process under uncertainty. In addition, we develop a Bayesian network-based risk assessment model to identify potential system risks and classify them into five levels (0-IV). Experimental evaluation using three task arrival rates (0.3, 0.6, 0.9) shows that our approach achieves 6.9-56% latency reduction and a wide range of energy consumption reduction compared to four state-of-the-art algorithms (GTO, WOA, NGO, DBO), while maintaining more than 98.6% reliability. The risk assessment consistently achieves zero risk classification in all test scenarios, indicating that the system performance is robust. This study provides a comprehensive framework for MEC task offloading. Multiple objectives are optimized simultaneously while ensuring system reliability.

Keywords: task offloading strategy, mobile edge computing, task arrival rate, minimizing latency, risk assessment

1. Introduction

With the rapid growth of the number of smart terminals and the increasing complexity of mobile applications, the demand for real-time processing of massive data has prompted Mobile Edge Computing (MEC) to become a key technology to alleviate the load of cloud computing centers, reduce service delays, and improve system response speed. MEC supports local computing offload of terminal devices close to data sources by sinking computing resources from centralized cloud platforms to the edge of the network, thereby improving user experience and system robustness. It is widely used in scenarios with extremely high timeliness requirements such as smart transportation, augmented reality, and industrial Internet of Things.

In the MEC architecture, task offloading refers to migrating some or all computing-intensive tasks from terminal devices to edge servers or cloud servers for processing (Zhang et al., 2021) to make up for the lack of terminal computing power. However, existing task offloading strategies still face many challenges: on the one hand, data transmission in centralized computing mode often leads to high latency, high energy consumption and bandwidth congestion, which seriously affects the stability and responsiveness of the system (Sriram, 2022); on the other hand, the dynamic decision of task offloading depends on complex environmental state information, and is easily affected by factors such as network instability and uneven distribution of computing resources, making it difficult to ensure execution reliability (Sabireen & Neelanarayanan, 2021; Ramesh et al., 2022). Especially in MEC scenarios where multiple users and multiple tasks are online at the same time, traditional static offloading mechanisms are difficult to adapt to complex and changing scheduling requirements, and a more intelligent, efficient and robust task decision mechanism is urgently needed.

Current research focuses on single-objective optimization of latency or energy consumption, ignoring the multi-objective conflicts and risk factors in the task offloading process, which makes it difficult for the strategy to run stably in actual deployment, or difficult to adapt to sudden network anomalies and edge resource changes. In this regard, this paper focuses on the multi-objective optimization task offloading problem in the MEC environment, attempts to construct an offloading strategy that takes into account latency, energy consumption and task success rate, and introduces risk modeling to improve system robustness.

Specifically, the research work of this paper includes the following three aspects:

(1) Construct a joint model based on long short-term memory network (LSTM) and Markov decision process (MDP), design an optimization task offloading strategy for dynamic environment, covering modules such as node state perception, task segmentation, offloading path decision, data upload and result feedback, and strive to improve the task execution success rate while minimizing latency and energy consumption;

(2) Design a risk assessment model for task offloading based on Bayesian network, identify key risk nodes in the task offloading process, and optimize parameter configuration through training to improve the system's adaptive ability under abnormal conditions;

(3) Verify the performance and risk response capabilities of the proposed model through simulation experiments, and analyze its applicability and promotion value in multi-task and multi-node dynamic environments.

2. Related Work

The current academic community has proposed certain research methods in task offloading strategies, which are mainly divided into two categories:

The first category is about energy consumption. Xie et al. (2024) considered energy consumption and server rental energy consumption, first divided application tasks and designed sub task priorities, and then proposed a multi-user sub-task scheduling scheme. He designed an improved Simulated

Annealing Particle Swarm Optimization (SAPSO) (You & Tang, 2021) to solve the minimum system total energy consumption, thereby achieving the best offloading decision. Liu et al. (2023) proposed a heterogeneous task offloading method that combined service updates and cloud edge collaboration. Firstly, by using an improved Page Replacement Algorithm (Gorawski & & Gorawska, 2023), potential user service demands were predicted and edge server services were updated in a timely manner. Secondly, based on cloud edge collaboration, task offloading was achieved through an improved Greedy Algorithm (Chen et al., 2021), while minimizing energy consumption. Li et al. (20204) proposed a task offloading method based on multi-agent deep reinforcement learning. Firstly, the task volume, service resources, and queue load were comprehensively considered with the goal of minimizing the total energy consumption of the system. The priority experience replay mechanism was applied to improve the multi-agent deep deterministic strategy gradient algorithm, successfully reducing the energy consumption of task offloading.

The second type is to consider the aspect of latency. To obtain the optimal task offloading decision and resource allocation scheme, Jiang et al. (2024) proposed a computational offloading strategy based on improved particle swarm optimization, which considered latency and server task balance, and adaptively adjusts latency. Zhou et al. (2024) focused on video detection and optimized system latency, and established computational offloading models and Markov decision models. Considering the complex dynamic factors of computational offloading scenarios, a computational offloading strategy based on deep reinforcement learning (Ladosz et al. 2022) was proposed for solution, ultimately reducing system latency. These methods have effectively solved some issues related to task offloading, but it should be noted that these methods only consider a certain point, such as single type offloading tasks, without considering the heterogeneity of offloading tasks, only considering energy consumption and latency, without considering other issues such as reliability, and without evaluating risks. Overall, the balance between multiple objectives is neglected, making it difficult to meet the diverse needs of different application scenarios.

Therefore, this article proposed a multi-objective optimization-based MEC (Siriwardhana et al. 2021) task offloading strategy and its risk assessment model construction. MEC sinks computing resources to the edge of the network, allowing tasks to be processed closer to the data source (Baas et al. 2020), reducing the distance and time of data transmission, lowering latency, and improving the real-time response capability of the system. Task processing on the edge computing node (Smith et al. 2020) can make full use of the computing resources of the edge node, reduce the dependence on the cloud server, and reduce the pressure on network bandwidth (Alikhan et al. 2023) and data transmission energy consumption. By adopting the MEC strategy, tasks can be distributed and executed on multiple edge nodes, forming a distributed architecture (Moustafa, 2021). Its built-in automatic fault tolerance and self-healing capabilities enable seamless service switching and continuous data processing. When a node fails, the system can automatically switch to a backup node, effectively reducing system latency (Yu et al. 2020), reducing energy consumption, and improving overall system reliability. After building a risk assessment model, operators can better understand and predict the performance of the system under different conditions, quantify the reliability of task offloading, and formulate corresponding offloading strategies, laying the foundation for building an efficient and reliable MEC system.

3. Methods

3.1. Design of MEC Task Offloading Strategy

3.1.1. MEC

Mobile edge computing aims to move computing resources and storage functions to the edge of the network, close to users and data sources, and meet the growing demand of mobile applications and services for low latency, high bandwidth, high reliability and security. Edge nodes are located at the edge of mobile networks, including base stations, WiFi access points, edge servers, and other locations. The core principle is to sink computing, storage, and network functions to the edge of the network, allowing mobile terminal devices to access computing resources and services more quickly. These functions make mobile edge computing an important infrastructure supporting various mobile applications and services, and are widely used in various fields. The MEC architecture includes cloud layer, edge layer, and terminal layer, each playing a different role in the entire computing architecture, providing different functions and services, as shown in Figure 1:

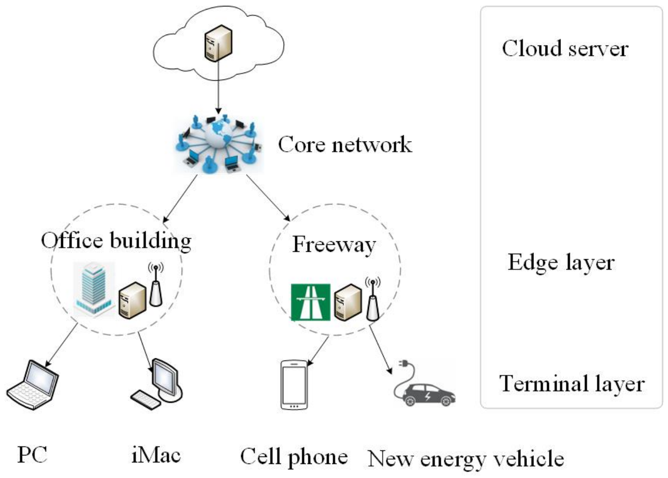

Fig 1: MEC architecture

Figure 1 shows the layered architecture of mobile edge computing (MEC), which consists of the cloud, edge, and terminal layers. In the figure, the terminal layer covers a variety of devices such as PCs and iMacs in office buildings and mobile phones and new energy vehicles on highways. These terminal devices can offload tasks to the edge layer closer to the user, which can reduce the distance and time of data transmission, thereby reducing latency and improving system response speed. The edge layer is connected to the core network, which is further connected to the cloud server. The cloud server is mainly responsible for performing large-scale computing tasks, while the edge layer uses infrastructure such as base stations to provide low-latency computing services to terminal devices. Through this architecture, MEC gives full play to the advantages of each layer to achieve efficient task processing and resource allocation.

3.1.2. Task Offloading in Mobile Edge Computing

One of the core concepts of mobile edge computing is task offloading. Mobile terminal devices can delegate some computing tasks to edge nodes to process, reducing the computing pressure of terminal devices. In the MEC system, the task offloading mechanism operates as follows: tasks on mobile devices are transferred to MEC servers within the communication coverage. This process involves transmitting computing tasks to MEC servers through uplink wireless links, aiming to accelerate task processing speed. This task offloading strategy effectively reduces the energy consumption of mobile devices when running applications. Afterwards, it is considered that whether the task needs to be uninstalled and to which server, and the partition situation of the task is determined. However, the limited load makes it difficult for the server to continuously provide high-quality services, so achieving reasonable resource allocation is also a challenge. The offloading of MEC tasks also requires a comprehensive consideration of task offloading and resource allocation issues. Therefore, whether the allocation of computing resources during the task offloading process has achieved the expected optimization goals is also a problem that needs to be solved. Figure 2 shows the uninstallation model of the MEC system.

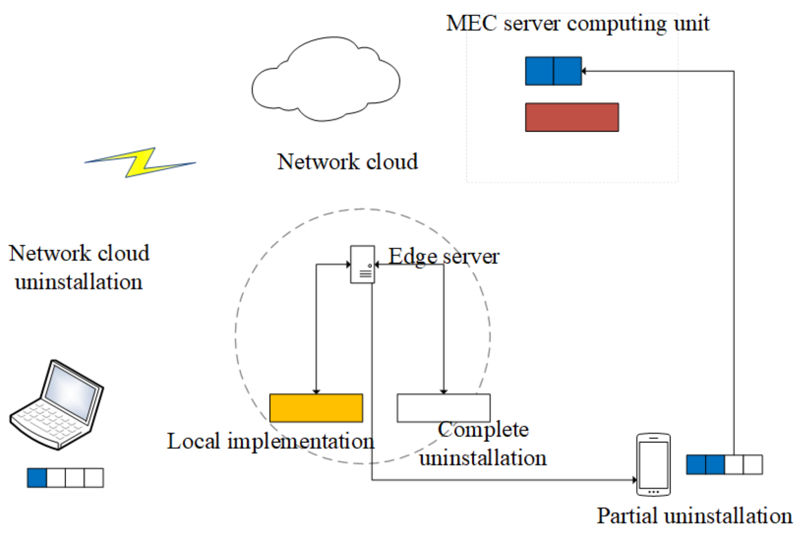

Fig 2: MEC system uninstallation model

Figure 2 shows the task offloading model in the mobile edge computing (MEC) system, covering three strategies: local execution, complete offloading, and partial offloading. Local execution means that the task is completed completely on the terminal device; complete offloading means that all tasks are transferred to the edge server for processing; partial offloading is somewhere in between, where only part of the task is offloaded to the edge server. This model helps optimize the allocation of computing resources, improve computing efficiency, and reduce energy consumption.

The uninstallation of tasks can be divided into local processing, partial uninstallation, complete uninstallation, and cloud task uninstallation. Different uninstallation methods can be selected according to different needs. In mobile edge computing, there are two methods: local computing and offloading computing. The offloading process is divided into node awareness, task segmentation, offloading decision, task upload, task computing, and result retrieval. The process is shown in Figure 3:

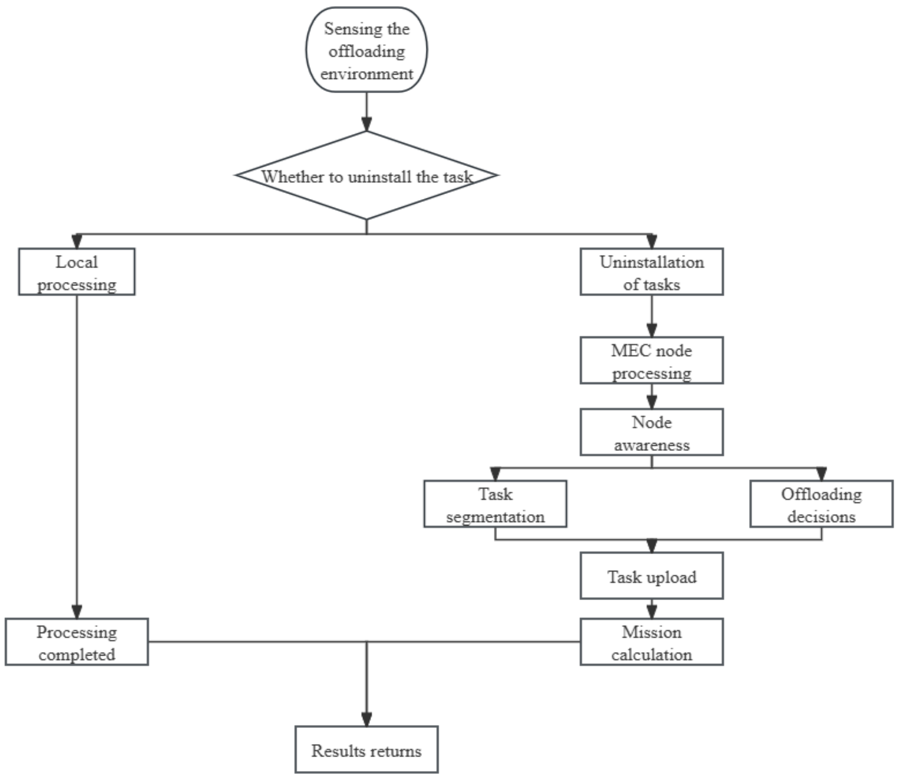

Fig 3: MEC task uninstallation process

Figure 3 presents the task offloading process in the mobile edge computing (MEC) system in the form of a flowchart. First, the system senses the offloading environment and then decides whether to offload the task. If you choose not to offload, the task will be processed locally and the result will be returned directly after completion. If you choose to offload, the process enters the offloading decision stage, which involves three steps: task segmentation, node perception, and offloading decision. Task segmentation breaks down large tasks into small subtasks, node perception collects network and computing resource information, and offloading decision determines which subtasks need to be offloaded to the MEC node. Then, the subtasks are uploaded to the MEC server for task calculation. Finally, the calculation results are returned to the terminal device, completing the entire task offloading process.

Node awareness aims to collect and evaluate the current network environment and computing resources. Computing resource awareness involves monitoring the CPU utilization, memory usage, and available storage space of MEC servers to determine their processing capacity and load conditions, ensuring efficient processing of tasks after offloading. Task segmentation decomposes complex computing tasks into independently processable subtasks, and adopts a segmentation strategy to divide tasks into parallel tasks. The task data is divided into multiple data blocks and compressed to reduce transmission time and bandwidth consumption, ensuring efficient execution of tasks on edge servers. The offloading decision determines which tasks need to be offloaded to the MEC server for execution. The decision-making process adopts the greedy algorithm in heuristic algorithms, and the decision indicators include latency minimization, energy minimization, and reliability maximization. This step ensures that tasks are efficiently executed while meeting performance requirements. The divided subtasks and data are uploaded to the MEC server to ensure that the uninstalled tasks can be smoothly transmitted and processed. In the following text, real-time data is not used in this article, so TCP (Transmission Control Protocol) protocol is chosen as the transmission protocol in this article. Task computation includes task scheduling and computation execution. The results generated during the task execution process are merged and validated. The efficient computing power of edge servers improves task processing speed and system performance. Finally, after the uninstallation task is processed, the server transmits the processing results of the task to the corresponding device via the downlink.

3.2. Long Short-term Memory Network

Long short-term memory network (LSTM) (Moghar and Hamiche, 2020) is an improved recurrent neural network (Tu et al., 2022), which is specially designed to capture long-term dependencies in time series data and can effectively alleviate the problems of gradient disappearance (Han et al. 2020) and gradient explosion (Liu et al. 2021). It dynamically controls and updates information by introducing gating mechanisms such as input gate, forget gate and output gate, thereby improving the sequence modeling ability (Lindemann et al. 2021). Figure 4 shows the network structure of LSTM.

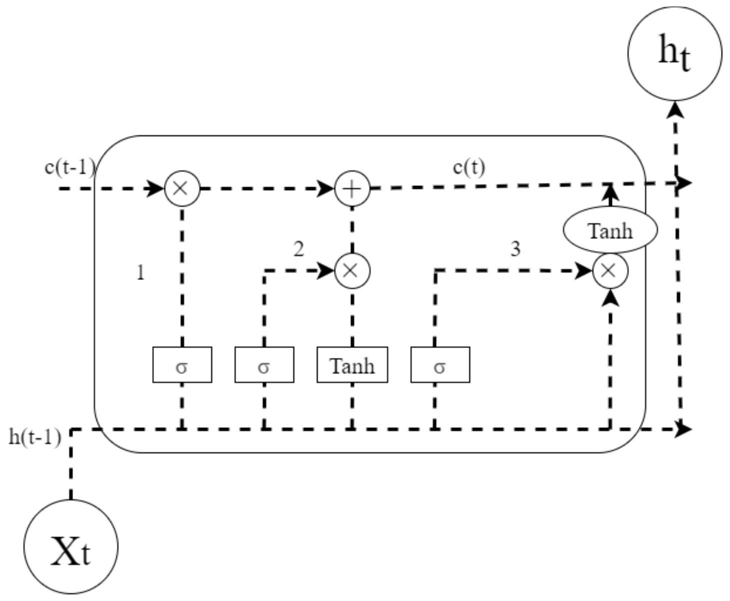

Fig 4: LSTM network structure

In the multi-objective optimization MEC task offloading scenario in this paper, LSTM is used to learn the complex mapping relationship between historical network status, computing resource distribution and task characteristics, and realize dynamic decision-making on offloading strategy. Taking time t as an example, the input of the model includes network delay, bandwidth, edge server load and task computing requirements, and the output is the offloading selection of the current task.

Among them, the input gate controls the influence of new input information on the unit state, which is calculated as follows:

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \quad (1)$$

The forget gate is used to decide whether to retain or discard previous state information:

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \quad (2)$$

The output gate determines the current hidden state:

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \quad (3)$$

$$h_t = o_t \cdot \tanh(C_t) \quad (4)$$

The cell state C_t integrates the combined effects of the historical state and the current input for long-term dependency modeling. The formula is as follows:

$$C_t = f_t \cdot C_{t-1} + i_t \cdot \tilde{C}_t \quad (5)$$

3.3. Markov Decision Process

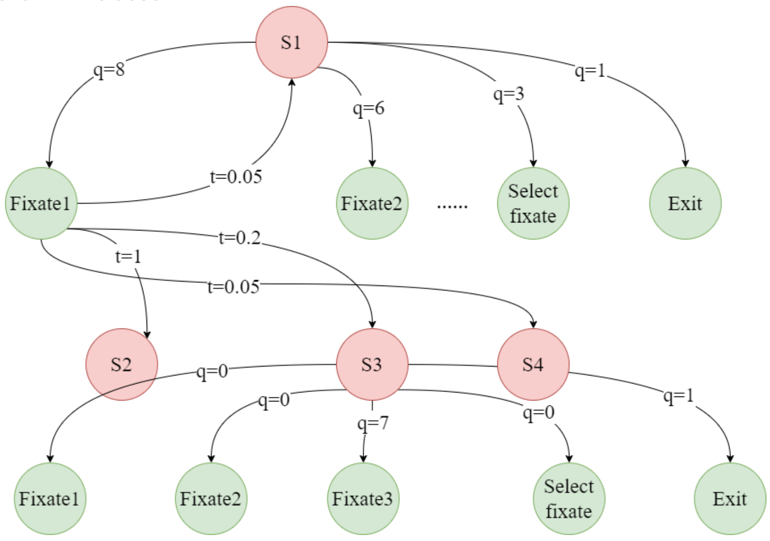

Fig 5: Markov decision structure diagram

Markov decision process (MDP) (Chades et al. 2021) is a powerful mathematical tool invented to solve sequential decision problems. It is suitable for task offloading decisions in dynamic and uncertain environments. Its basic principle is to model the task offloading problem as MDP and achieve optimal decision-making in diverse network and computing environments. Its structure is shown in Figure 5.

For each state s_t, the action a_t represents an optional offloading strategy, including local execution, offloading to a specific edge node edge_i, or uploading to the cloud. Due to the high dynamics of the MEC environment, traditional static modeling methods are difficult to accurately characterize the state evolution process. This paper introduces the LSTM network to model the state transition probability, that is:

$$P(s_{t+1}|s_t, a_t) = \text{LSTM}(s_t, a_t) \quad （7）$$

By learning from historical states and action sequences, LSTM can dynamically predict the impact of offloading behavior under different network and resource conditions, and enhance the strategy's ability to adapt to environmental changes.

In order to optimize the overall performance of the offloading strategy, the reward function designed in this paper comprehensively considers task execution delay, energy consumption overhead, and offloading success rate, and is defined as follows：

$$\mathcal{R}(s_t, a_t) = w_1 \cdot L_t + w_2 \cdot C_t + w_3 \cdot R_t \quad （8）$$

Among them, L_t represents the delay of task execution, C_t represents energy consumption and resource cost, and R_t represents the reliability of successful offloading; w_1, w_2, and w_3 are corresponding weight parameters, which are used to reflect the importance of different optimization objectives. By introducing LSTM-based state modeling and multi-objective reward mechanism, the task offloading framework constructed in this paper can achieve more robust and real-time optimal strategy generation under variable network conditions.

3.4. Designing MEC Task Offloading Strategies Using MDP and LSTM Applications

In order to achieve a more efficient task offloading strategy, this paper constructs a multi-objective optimization model, which aims to minimize the total task delay, minimize energy consumption and maximize the reliability of task execution. Assume that at time step t, the system state is s_t, the offloading action is a_t, and the corresponding performance indicators include task delay L_t, energy consumption C_t and reliability score R_t. The multi-objective optimization problem can be formally expressed as:

$$\min_{a_t \in A} J(a_t) = w_1 \cdot L_t + w_2 \cdot C_t - w_3 \cdot R_t \quad (9)$$

Among them, w_1, w_2, w_3 are the importance weight coefficients of delay, energy consumption and reliability respectively, satisfying $w_1, w_2, w_3 \in [0,1]$ and $w_1 + w_2 + w_3 = 1$. Each performance indicator in the above objective function satisfies the following constraints

$$L_t \leq L_{\max}, C_t \leq C_{\max}, R_t \geq R_{\min}, a_t \in \{\text{local}, \text{edge}_i, \text{cloud}\} \quad (10)$$

In order to provide high-quality initial environment states and offloading scenario samples, this paper uses the public Partial Computation Offloading for MEC dataset, which is specially built for edge computing scenarios. It contains network parameters (bandwidth, latency), edge server configuration (computing power, load), task characteristics (data volume, computing requirements, deadlines), and multi-dimensional combinations of offloading actions and results, which can support the training of representative optimization models.

In terms of model architecture, this paper proposes to organically combine LSTM and MDP to solve the problem of inaccurate state modeling in traditional reinforcement learning in dynamic task offloading scenarios. Specifically, the MDP structure provides a theoretical framework for modeling task offloading strategies, which is suitable for describing state transitions and reward feedback processes; while the LSTM network is used to model the state transition function of P(s_(t+1) |s_t,a_t), which has the ability of time series modeling and can effectively capture time-dependent characteristics such as network delay fluctuations and edge node resource changes. Compared with standard reinforcement learning methods based on Q-learning or policy gradient, this method has better generalization ability when facing high-dimensional environmental states and uncertain state transitions, and is especially suitable for delay-sensitive and resource-heterogeneous edge computing environments. The goal of multi-objective optimization is to reduce the total time from task offloading to result return, reduce the energy consumption of edge and cloud resources, and increase the probability of successful task execution, that is, minimize delays, minimize energy consumption, and maximize reliability. This study selects data from the Partial Computation Offloading for MEC dataset as the initial state information of the network and system. This dataset is a deep learning dataset specifically used to study task offloading optimization in edge computing environments. It contains multi-dimensional data such as network environment parameters, edge server parameters, task parameters, offloading decision parameters, and performance indicators.

LSTM model training adjusts model parameters through backpropagation algorithms. To evaluate the convergence of the algorithm, this article conducts experiments using different learning rates and batch sizes. The data is normalized and scaled to the range of [0,1]. An LSTM model consisting of an input layer, three LSTM layers, and a fully connected layer is constructed, and the mean square error is selected as the loss function. In terms of optimizer, Adam is selected and four learning rates are set: random, 0.1, 0.01, 0.001. In terms of batch training, the training data is divided into different small batches, namely random, 16, 8, and 4. Then, each batch of data is input into the LSTM layer and fully connected layer to calculate the results. Based on the predicted results and actual labels, the energy consumption cost is calculated. Finally, the backpropagation algorithm is adopted to calculate gradients and update model parameters. The above steps are followed to traverse the training dataset 5 times, and the training loss and validation loss at the end of each iteration are recorded, gradually optimizing the model parameters. This article sets a system training period of 815 cycles, and task offloading lasts for 100 time slots in each cycle. The specific experimental results are as follows:

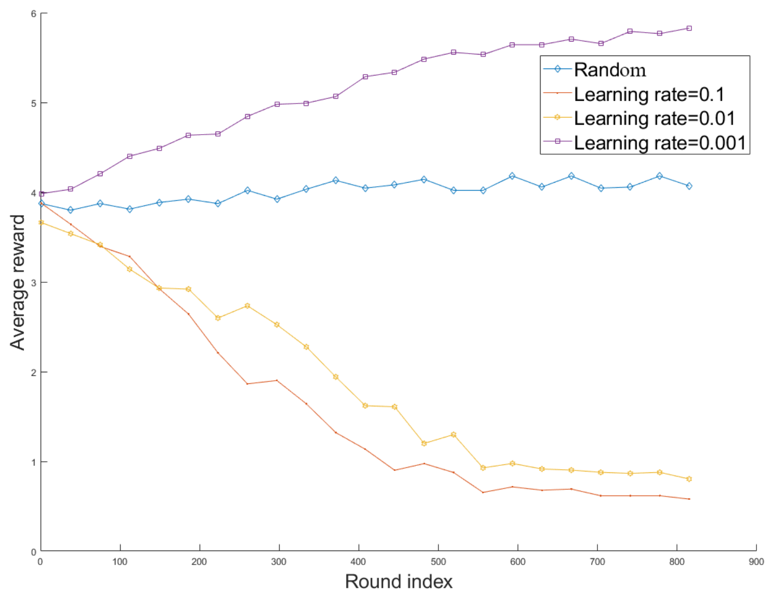

Fig 6: Convergence performance under different learning rates

As mentioned earlier, four types of learning rates are set during model training: random, 0.1, 0.01, and 0.001. Energy consumption costs (convergence reward values) are recorded during the iteration process. As shown in Figure 6, with a random learning rate, the initial energy cost is 3.873. As the number of iterations increases, the energy cost shows little fluctuation and does not show a clear convergence trend throughout the entire training process, indicating that the random learning rate method has a high energy cost and is stable at high energy consumption costs without any convergence trend. When the learning rate is 0.1, the initial energy cost is 3.877, which rapidly decreases in the subsequent iteration process. In subsequent iterations, the energy cost tends to stabilize. In the 815th iteration, the energy cost is 0.580, showing good convergence. When the learning rate is 0.01, the initial energy cost is 3.66, and then it rapidly decreases. In the 815th iteration, the energy cost is 0.804, which also shows good convergence, but slightly worse than when the learning rate is 0.1. When the learning rate is 0.001, the energy cost does not decrease but increases, and the algorithm shows very poor convergence.

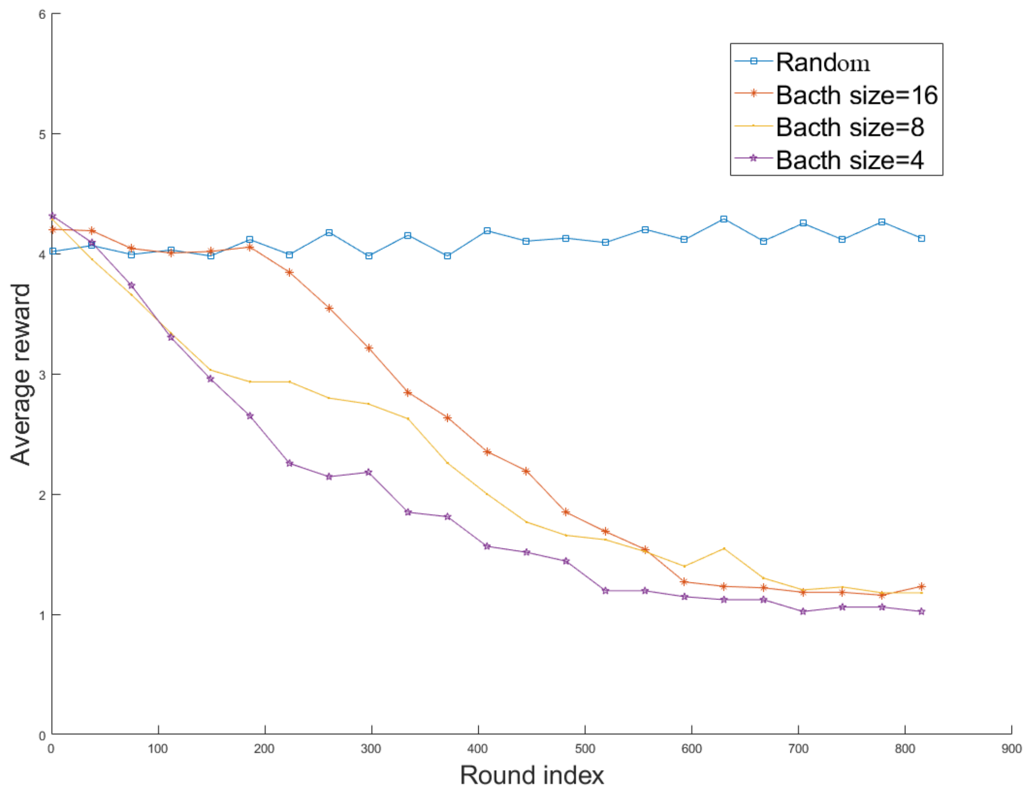

Fig 7: Convergence performance under different sampling batches

As shown in Figure 7, during random sampling, the initial energy cost is 4.0164, and there is almost no change in the energy cost during the iteration process. The final energy cost is 4.1273, indicating that the model fails to converge effectively and the energy cost remains at a high level. When the sampling batches are 16, 8, and 4, the algorithm shows good convergence, gradually starting to converge after 500 iterations. When the batch value is 16, the final energy consumption cost is 1.232; when the batch value is 8, the final energy consumption cost is 1.178; when the batch value is 4, the final energy cost is 1.023. When observing the sampling batches of 16, 8, and 4, it is found that the smaller the sampling batch, the faster the convergence speed of the model, and the lower the final energy consumption cost. However, if the sampling batch is too small, it also increases training time and computational resource consumption. Therefore, in balancing the convergence effect and computational cost, this article tends to adopt a sampling batch size of 8.

3.5. Construction of Risk Assessment Model

The main risk factors that need to be evaluated in the risk assessment model of this article are latency, energy consumption fluctuations, reliability (task success rate), and security risks. Compared with the task offloading strategy, there is an additional security risk. Relevant evaluation indicators are defined as the time fluctuation from task submission to result return, energy consumption fluctuations generated during task calculation and transmission, fluctuation in task success execution rate, and security evaluation of data during transmission and calculation. In terms of dataset, the Partial Computation Offloading for MEC mentioned earlier is still used, but there are slight differences in the type of data. The data sources in this section include historical task execution data in the MEC environment, network performance data, device status data, and security event records. After data cleaning, outlier handling, and data standardization, four types of risk related features are extracted from these data: mean and standard deviation of latency, trend of energy consumption, task failure rate, and frequency of safety events. These features are used as inputs for the risk assessment model. When training the model, accuracy (ACC), precision (P), recall (R), and F1 score (F1) are set as the four indicators for model evaluation, and the maximum number of iterations is set to 300. Table 1 shows a brief process of training the model:

Table 1. Training of risk assessment model

Number of iterations	ACC	P	R	F1
10	0.5173	0.5204	0.5261	0.5182
35	0.5421	0.5468	0.5522	0.5436
60	0.5694	0.5721	0.5782	0.5708
85	0.6043	0.6075	0.6139	0.6061
110	0.6507	0.6522	0.6588	0.6520
135	0.6972	0.7004	0.7065	0.6999
160	0.7344	0.7379	0.7431	0.7368
185	0.7841	0.7873	0.7937	0.7865
210	0.8483	0.8505	0.8561	0.8493
235	0.9145	0.9162	0.9400	0.9100
260	0.9234	0.9181	0.9391	0.9112
285	0.9153	0.924	0.9357	0.9173
300	0.9169	0.9254	0.9334	0.9137

From Table 1, it can be seen that the initial values of all evaluation indicators are between 0.5100 and 0.5300, and their values gradually increase with the increase of iteration times. Specifically, the accuracy (ACC), precision (P), recall (R), and F1 score (F1Score) all reach above 0.6500 after 110 iterations. This indicates that the model effectively learns the features in the training data, and improves overall classification accuracy, recognition ability for positive samples, and detection ability for positive samples, and balances accuracy and recall in the early training stage. The algorithm begins to converge on these four indicators at the 235th iteration, and the values of all four indicators exceed 0.9100, indicating that the algorithm has good convergence. The values of the indicators after the algorithm begins to converge do not fluctuate more than 0.100 compared to the values at the 235th iteration, indicating that the model is robust. Overall, the indicators of the model rapidly increase in the early iterations, steadily increase in the mid-term iterations, and converge to a relatively high level in the later iterations.

Next, this study has made more rigorous improvements and refinements to the risk level classification of the risk assessment model. Based on the analysis of a large amount of experimental data and the in-depth insights of experts in the field, clear and operational risk level classification standards are set for each risk factor, as follows:

For delay risk, we determine the risk level by calculating the time fluctuation from task submission to result return, that is, the standard deviation of delay σ, combined with the mean value of delay μ. If $\mu + \sigma \leq T1$ (T1 is the delay threshold set according to the actual application scenario), it is determined as a level 0 risk; if T1< $\mu + \sigma \leqslant T2$(T2 is the delay threshold 2), it is determined as a level I risk; and so on, until $\mu + \sigma$ >T4 is determined as a level IV risk.

For energy consumption fluctuation risk, the risk level is divided according to the energy consumption fluctuation generated during task calculation and transmission, that is, the standard deviation of energy consumption σ_e. Set energy consumption fluctuation thresholds E1, E2, E3, and E4. When $\sigma_e \leqslant$E1, it is a level 0 risk; when E1<$\sigma_e \leqslant$E2, it is a level I risk; and so on, when σ_e>E4, it is a level IV risk.

For reliability risk, the risk level is determined according to the fluctuation of the task success execution rate, that is, the task failure rate p. Determine the reliability risk thresholds P1, P2, P3, and P4. If p⩽P1, it is a level 0 risk; P1<p⩽P2 is a level I risk; and so on, p>P4 is a level IV risk.

For safety risk, it is measured in combination with the frequency of safety incidents f. Set safety risk thresholds F1, F2, F3, and F4. When f⩽F1, it is a level 0 risk; when F1<f⩽F2, it is a level I risk; and so on, f>F4 is determined to be a level IV risk.

When determining the final risk level by comprehensively considering various risk factors, a weighted summation method is adopted to assign corresponding weights w1, w2, w3, w4 (the sum of the weights is 1) to each risk factor, and the comprehensive risk value Rv is calculated according to the following formula:

$$Rv = w1 \times R_{delay} + w2 \times R_{energy} + w3 \times R_{reliability} + w4 \times R_{security} \quad (11)$$

Among them, R_{delay}, R_{energy}, $R_{reliability}$, and $R_{security}$ represent the level values corresponding to delay, energy consumption fluctuation, reliability, and security risk respectively (integer values of 0 to 4). According to the size of the comprehensive risk value Rv, it is divided into different risk level intervals, and then the final risk level is determined. Table 2 shows the parameter adjustment during the model training process. Through multiple rounds of combined experiments on key parameters such as learning rate, regularization parameter, optimizer, and batch size, their comprehensive impact on the model convergence speed is deeply explored.

Table 2. Parameter tuning

Number	Learning rate	Regularization parameter	Optimizer	Batch size	Convergence speed(epoch)
1	0.01	0.1	Adam	32	50
2	0.001	0.01	SGD	64	200
3	0.005	0.05	RMSprop	128	120
4	0.01	0.01	Adam	64	80
5	0.001	0.1	SGD	32	250
6	0.01	0.05	RMSprop	64	100
7	0.005	0.01	Adam	128	90
8	0.01	0.1	SGD	32	300
9	0.005	0.05	RMSprop	128	110
10	0.001	0.01	Adam	64	180

Note: Adam (Adaptive Moment Estimation), SGD (Stochastic Gradient Descent), RMSProp (Root Mean Square Propagation)

Observing the entire Table 2, it is not difficult to find that during the first parameter tuning, the learning rate is set to 0.01; the regularization parameter is set to 0.1; the optimizer is set to Adam; the batch size is set to 32. The final convergence speed is 50 epochs, making it the best parameter combination for these 10 experiments. Furthermore, the setting with a learning rate of 0.01 shows a faster convergence rate. The convergence rates for Experiment 1 and Experiment 4 are 50 and 80 epochs, respectively, but in Experiment 8, they are 300 epochs. When the regularization parameter is 0.1, the convergence speed in Experiment 5 and Experiment 8 is 250 epochs and 300 epochs, showing poor convergence. On the contrary, it shows good convergence in low regularization (0.01). The Adam optimizer performs well in most cases, and the batch size does not show significant convergence. Based on the above analysis, this article mainly considers the parameters of Experiment

1 in the selection of model parameters, but makes slight adjustments by setting the regularization parameter to 0.01.

Finally, the designed risk assessment model is integrated into the MEC task offloading system to monitor the system's operational status and performance indicators in real-time, obtain risk prediction results, and help system administrators understand the system's operational status, so as to formulate optimization strategies and improvement measures, and achieve dynamic adjustment of offloading strategies.

4. Simulation Experiment Design

In terms of hardware, CPU: Intel Core i7-9700K; Memory: 32 GB DDR4; Storage: 512 GB SSD; GPU: NVIDIA GTX 1650. The simulator adopts Android Virtual Device. In terms of software, both servers and edge nodes run the Ubuntu 20.04 LTS operating system; the mobile device simulator runs Windows 10; TensorFlow and PyTorch are selected for machine learning modeling; OpenStack and Kubernetes are used for edge computing framework and container orchestration.

4.1. Strategy Verification

In order to verify the effectiveness and superiority of the multi-objective optimization task offloading strategy proposed in this paper, this section conducts a rigorous experimental design and applies four swarm intelligence optimization algorithms for comparative experiments: Gorilla Troops Optimizer (GTO) (Abdollahzadeh et al. 2021; Hussien et al. 2024; El-Dabah et al. 2022), Whale Optimization Algorithm (WOA) (Rana et al. 2020), Northern Goshawk Optimization (NGO) (El-Dabah et al. 2023) and Dung Beetle Optimizer (DBO) (Xue and Shen, 2023). These algorithms were selected because of their unique advantages in multi-objective optimization and MEC task offloading, providing diverse solutions to multi-objective optimization problems. The GTO algorithm is known for its global search capability; the WOA algorithm achieves a good balance between global and local search; the NGO algorithm combines high-altitude and low-altitude search strategies to adapt to different task offloading requirements; and the DBO algorithm is selected for its simple structure and easy implementation on edge nodes.

The experimental data set has been strictly split and preprocessed, including data cleaning, standardization and cross-validation, to ensure the reliability and repeatability of the results. In terms of evaluation indicators, we adopted three goals: latency, energy consumption and reliability (task execution success rate). Latency refers to the total time from uploading a task to the edge server, calculating it on the edge server, and downloading the result back to the terminal device. Energy consumption includes the sum of computing resource energy consumption and network transmission energy consumption.

Tables 3, 4 and 5 show the evaluation indicators of these algorithms and the algorithm in this paper at different task arrival rates, including statistical information such as mean, standard deviation, confidence interval, and a significance test was performed to verify the statistical significance of the results:

Table 3. Indicators of the algorithm when the task arrival rate is 0.3

Algorithm	Latency (ms)	Average energy consumption	Reliability	Confidence Interval (Delay)	Confidence interval (energy consumption)	Confidence interval (reliability)
TAITP	451 ±3.7	3.11±0.35	0.986±0.005	[512, 534]	[3.1, 3.3]	[0.98, 0.99]
GTO	531 ±6.3	3.23±0.3	0.978±0.007	[520, 540]	[3.0, 3.2]	[0.97, 0.99]
WOA	491 ±4.8	5.09±0.5	0.981±0.006	[482, 500]	[4.9, 5.3]	[0.97, 0.99]
NGO	698 ±8.5	4.65±0.45	0.912±0.010	[685, 710]	[4.5, 4.8]	[0.90, 0.93]
DBO	523 ±5.2	6.67±0.6	0.976±0.008	[444, 458]	[6.4, 6.9]	[0.97, 0.99]

Note: TAITP represents the algorithm in this article (the algorithms in this article)

Table 4. Indicators of the algorithm when the task arrival rate is 0.6

Algorithm	Latency (ms)	Average energy consumption	Reliability	Confidence Interval (Delay)	Confidence interval (energy consumption)	Confidence interval (reliability)
TAITP	467 ±3.9	3.12±0.25	0.991±0.004	[510, 526]	[3.0, 3.2]	[0.98, 1.00]
GTO	567 ±7.2	3.65±0.4	0.912±0.009	[555, 579]	[3.5, 3.8]	[0.90, 0.93]
WOA	511 ±4.5	5.9±0.55	0.923±0.007	[502, 520]	[5.6, 6.2]	[0.91, 0.94]
NGO	716 ±9.3	4.65±0.45	0.901±0.011	[703, 729]	[4.5, 4.8]	[0.89, 0.92]
DBO	518 ±4.8	6.9±0.65	0.921±0.009	[459, 475]	[6.6, 7.2]	[0.91, 0.94]

Note: TAITP represents the algorithm in this article (the algorithms in this article)

Table 5. Indicators of the algorithm when the task arrival rate is 0.9

Algorithm	Latency (ms)	Average energy consumption	Reliability	Confidence Interval (Delay)	Confidence interval (energy consumption)	Confidence interval (reliability)
TAITP	490±4.2	3.11±0.28	0.986±0.005	[516, 532]	[3.0, 3.2]	[0.98, 0.99]
GTO	698±8.7	3.98±0.5	0.901±0.012	[685, 711]	[3.8, 4.2]	[0.89, 0.92]
WOA	561±6.5	6.12±0.6	0.887±0.008	[550, 572]	[6.0, 6.3]	[0.88, 0.90]
NGO	765±10.2	4.89±0.52	0.849±0.013	[753, 777]	[4.7, 5.1]	[0.84, 0.86]
DBO	524±5.0	7.14±0.7	0.915±0.010	[482, 498]	[7.0, 7.3]	[0.90, 0.93]

Note: TAITP represents the algorithm in this article (the algorithms in this article)

As shown in Table 3, Table 4, and Table 5, this experiment takes three task arrival rates of 0.3, 0.6, and 0.9. The task arrival rate indicates the number of tasks generated per unit time, measured by the number of tasks arriving per second, and reflects the amount of computing task offloading that the system needs to process per unit time. Setting different task arrival rates can well reflect the robustness of the five algorithms. As can be seen from the table, the algorithm in this paper maintains a stable advantage under different task arrival rates. In terms of energy consumption, when the task arrival rate is 0.3, the algorithm in this paper is lower than the WOA, NGO, and DBO algorithms, and only slightly higher than the GTO algorithm; in terms of reliability, the algorithm in this paper always maintains a high task execution success rate, and is always higher than the other four algorithms. Although the WOA algorithm achieves lower latency in some cases, its energy consumption is significantly higher than the algorithm proposed in this paper, which shows that in practical applications, it is important to weigh different goals according to specific needs.

4.2. Risk Assessment

This section conducts risk assessment on these five algorithms during the calculation of task offloading process. Using the task offloading cases of the five algorithms with task arrival rates of 0.3, 0.6, and 0.9 set in the previous text, in order to visually demonstrate the risks during task offloading, this article links risks to colors. When there is a risk, the system displays the corresponding colors, as shown in Table 6:

Table 6. Corresponding hazard levels and risk levels

Degree of danger	Risk free	General	Relatively serious	Serious	Particularly serious
Color	No display	Blue	Yellow	Orange	Red
Risk levels in this article	0	I	II	III	IV

Table 7. Risk assessment

	TAITP	GTO	WOA	NGO	DBO
MAR=0.3	0	0	0	0	0
MAR=0.6	0	0	0	I	I
MAR=0.9	0	II	I	I	I

Note: TAITP represents the algorithm in this article (the algorithms in this article)

As shown in Table 7, MAR represents the Mission Arrival Rate. When the task arrival rate is 0.3, the risk assessment results of all algorithms are 0, which means that under low load conditions, the computing task offloading system under all algorithms runs stably. When the task arrival rate is 0.6, the risk assessment results of the algorithm in this article, GTO, and WOA algorithms are still 0. These three algorithms can maintain low risk under moderate load, while the NGO algorithm and DBO algorithm are both judged as general risks and require certain security measures. In this case, encrypted transmission and access control can be used as two coping methods. When the task arrival rate is 0.9, only the risk assessment result of the algorithm in this article is still 0. WOA, NGO algorithm, and DBO have general risks, while the risk level of GTO algorithm is relatively serious. In more serious risk situations, a security audit mechanism can be established to monitor the operation behavior during task offloading. The above analysis indicates that the algorithm proposed in this article can effectively address security risks.

5. Discussion

(1) Effectiveness analysis of multi-objective optimization strategy

The multi-objective optimization strategy proposed in this paper outperforms the comparative algorithms in terms of task offloading delay, energy consumption and reliability. This is mainly due to the combination of the long short-term memory network (LSTM) and the Markov decision process (MDP). LSTM can capture long-term dependencies in time series data and memorize complex mapping relationships such as historical network states, providing a basis for offloading decisions. MDP provides a mathematical framework to calculate the optimal offloading strategy based on state and action to maximize long-term cumulative rewards. The reward function cleverly balances the relationship between delay, energy consumption and reliability, and its weight parameters are carefully adjusted to reflect the actual importance of different goals. This is the key to the strategy's superiority over other algorithms.

(2) Performance of the strategy under different task arrival rates

In experiments with different task arrival rates, the proposed strategy shows good adaptability and robustness, especially under high task arrival rates, the performance remains stable. When the task arrival rate increases, the edge node load increases, and the network environment becomes complex. At this time, the proposed strategy can adjust the unloading decision in time, allocate resources reasonably, and ensure efficient task allocation and execution by relying on the dynamic learning ability of LSTM and the adaptability of MDP. In contrast, other algorithms lack this dynamic adjustment ability, resulting in performance degradation, which further highlights the advantages of the strategy in this paper.

(3) Accuracy of the risk assessment model

The risk assessment model can accurately identify potential risks in the task offloading process and classify them into different levels, providing decision support for system administrators. The high accuracy of the model is due to the carefully selected features, which fully characterize the risk situation and provide reliable data for training. The training process adjusts and optimizes the algorithm parameters to make the model converge to the optimal state and achieve accurate assessment. The model can identify high-risk scenarios in advance, assist administrators in taking preventive measures, and improve system security and stability.

(4) The impact of parameter tuning on model performance

Experiments show that hyperparameters such as learning rate and batch size have a significant impact on model performance. Taking the combination of a learning rate of 0.01 and a batch size of 8 as an example, the model shows faster convergence speed and lower training energy consumption. An appropriate learning rate ensures that the model converges quickly and stably, and a suitable batch size can improve training efficiency. This finding provides an important reference for subsequent model training, reminding researchers to pay attention to the selection of hyperparameters to improve model performance.

(5) Future research directions

Although this study has achieved certain results, it still has limitations. The insufficient coverage of the data set suggests that future research needs to use a wider range of data sets to optimize parameter tuning and improve the generalization ability of the model. In addition, in the face of large-scale task offloading scenarios, it is necessary to explore more efficient algorithms to optimize the model structure, reduce computing costs, and improve scalability. In the future, it is necessary to further study the impact of practical factors such as task heterogeneity and network dynamics on strategies, strengthen cooperation with the industry, and promote the application of results.

6. Conclusions

This study proposes a comprehensive MEC task offloading framework to address the key challenges of multi-objective optimization in dynamic edge computing environments. Its main contributions include: (1) a novel LSTM-MDP ensemble approach that achieves superior performance in terms of latency, energy consumption, and reliability metrics; (2) a robust risk assessment model that enables real-time system monitoring and threat classification; and (3) empirical validation showing sustained performance improvement under varying network loads. Experimental results show that latency and energy consumption are reduced compared to existing approaches while maintaining high reliability. However, this study also has limitations, including limited dataset scope and the need for real-world validation outside of simulation environments. Future research should focus on: (1) validating the framework in heterogeneous real MEC deployments; (2) extending the risk model to incorporate cybersecurity threats; (3) investigating adaptive weight adjustment mechanisms for dynamic multi-objective optimization; and (4) exploring federated learning ensembles to enhance privacy-preserving task offloading. The proposed framework provides a solid foundation for next-generation MEC systems that require intelligent, risk-aware task management.

Acknowledgments

This work is partially supported by the Natural Science and Technology Project Plan in Yulin of China (Grant No. CXY-2020-002-04), The Science and Technology Project Plan in Yuiln High Tech Industry Development Zone of China (Grant No. CXY-2021-37), and Thanks for the help.

References

Alikhan, J. S., Alageswaran, R., Amali, S. M. J. (2023). Dingo optimization based network bandwidth selection to reduce processing time during data upload and access from cloud by user. *Telecommunication Systems*, 83(2): 189-208. https://doi.org/10.1007/s11235-023-01002-8

Abdollahzadeh, B., Soleimanian Gharehchopogh, F., & Mirjalili, S. (2021). Artificial gorilla troops optimizer: a new nature-inspired metaheuristic algorithm for global optimization problems. *International Journal of Intelligent Systems*, *36*(10), 5887-5958. https://doi.org/10.1002/int.22535

Baas, J., Schotten, M., Plume, A., Côté, G., Karimi, R. (2020). Scopus as a curated, high-quality bibliometric data source for academic research in quantitative science studies. *Quantitative Science Studies*, 1(1): 377-386. https://doi.org/10.1162/qss_a_00019

Chades, I., Pascal, L. V., Nicol, S., Fletcher, C. S., Ferrer-Mestres, J. (2021). A primer on partially observable Markov decision processes (POMDPs). *Methods in Ecology and Evolution*, 12(11): 2058-2072. https://doi.org/10.1111/2041-210X.13692

Chen, J., Yang, Y., Wang, C., Zhang, H., Qiu, C., & Wang, X. (2021). Multitask offloading strategy optimization based on directed acyclic graphs for edge computing. *IEEE Internet of Things Journal*, *9*(12), 9367-9378. https://doi.org/10.1109/JIOT.2021.3110412

Gorawski, M., Pasterak, K., & Gorawska, A. (2023). The stream data warehouse: Page replacement algorithms and quality of service metrics. *Future Generation Computer Systems*, *142*, 212-227. https://doi.org/10.1016/j.future.2022.12.030

Hussien, A. G., Bouaouda, A., Alzaqebah, A., Kumar, S., Hu, G., & Jia, H. (2024). An in-depth survey of the artificial gorilla troops optimizer: outcomes, variations, and applications. *Artificial Intelligence Review*, *57*(9), 246-324. https://doi.org/10.1007/s10462-024-10838-8

El-Dabah, M. A., Hassan, M. H., Kamel, S., Zawbaa, H. Z. (2022). Robust parameters tuning of different power system stabilizers using a quantum artificial gorilla troops optimizer. *IEEE Access*, 10: 82560-82579. https://doi.org/10.1109/ACCESS.2022.3195892

Han, Q., Zhao, H., Min, W., Cui. H., Zhou, X., Zuo, K., Liu, R. (2020). A two-stream approach to fall detection with MobileVGG. *IEEE Access*, 8: 17556-17566. https://doi.org/10.1109/ACCESS.2019.2962778

Icarte, R. T., Klassen, T. Q., Valenzano, R., McIlraith, S. A. (2022). Reward machines: Exploiting reward function structure in reinforcement learning. *Journal of Artificial Intelligence Research*, 73: 173-208. https://doi.org/10.1613/jair.1.12440

Jiang, P., Fu, S., Ding, C. (2024). A computational offloading strategy based on improved particle swarm optimization in multi-device and multi-task scenarios. *Journal of Heilongjiang Bayi Agricultural Reclamation University*, 36 (01): 98-107. doi:10.3969/j.issn.1002-2090.2024.01.015

Liu, F., Huang, J., & Wang, X. (2023). Joint task offloading and resource allocation for device-edge-cloud collaboration with subtask dependencies. *IEEE Transactions on Cloud Computing*, *11*(3), 3027-3039. https://doi.org/10.1109/TCC.2023.3251561

Liu, M., Chen, L., Du, X., Jin, L., Shang, M. (2021). Activated gradients for deep neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 34(4): 2156-2168. https://doi.org/10.1109/TNNLS.2021.3106044

Li Yinghao, Liu Panpan, Wang Wenmeng, Liu Xiaoliang, Han Zhiyong & Liu Chengming. (2025). Task offloading method for deep reinforcement learning in edge-end collaboration scenario. *Journal of Small and Micro Computer Systems*, 46(02), 280-288. doi:10.20009/j.cnki.21-1106/TP.2023-0474.

Lindemann, B., Müller, T., Vietz, H., Jazdi, N., Weyrich, M. (2021). A survey on long short-term memory networks for time series prediction. *Procedia Cirp*, 99: 650-655. https://doi.org/10.1016/j.procir.2021.03.088

Ladosz, P., Weng, L., Kim, M., Oh. H.(2022). Exploration in deep reinforcement learning: A survey. Information Fusion, 85: 1-22. https://doi.org/10.1016/j.inffus.2022.03.003

Moustafa, N. (2021). A new distributed architecture for evaluating AI-based security systems at the edge: Network TON_IoT datasets. *Sustainable Cities and Society*, 72: 102994-103030. https://doi.org/10.1016/j.scs.2021.102994

Moghar, A., Hamiche, M. (2020). Stock market prediction using LSTM recurrent neural network. *Procedia computer science*, 2020, 170: 1168-1173. https://doi.org/10.1016/j.procs.2020.03.049

Murugan, S., Jaishankar, M., Premkumar, K. (2022). Hybrid DC−AC microgrid energy management system using an artificial gorilla troops optimizer optimized neural network. Energies, 15(21): 8187-8206. https://doi.org/10.3390/en15218187

Ramesh, G., Logeshwaran, J., Aravindarajan, V. (2022). The performance evolution of antivirus security systems in ultra dense cloud server using intelligent deep learning. *BOHR International Journal of Computational Intelligence and Communication Network*, 1(1): 15-19. https://doi.org/10.54646/bijcicn.004

Rana, N., Latiff, M. S. A., Abdulhamid, S. M., Chiroma, H. (2020). Whale optimization algorithm: a systematic review of contemporary applications, modifications and developments. *Neural Computing and Applications*, 32: 16245-16277. https://doi.org/10.1007/s00521-020-04849-z

Sriram, G. S. (2022). Edge computing vs. cloud computing: an overview of big data challenges and opportunities for large enterprises. *International Research Journal of Modernization in Engineering Technology and Science*, 4(1): 1331-1337. http://dx.doi.org/10.0202/Computin.2022197786

Sabireen, H., Neelanarayanan, V. (2021). A review on fog computing: Architecture, fog with IoT, algorithms and research challenges. *Ict Express*, 7(2): 162-176. https://doi.org/10.1016/j.icte.2021.05.004

Smith, R., Palin, D., Ioulianou, P. P., Vassilakis, V. G., Shahandashti, S. F. (2020). Battery draining attacks against edge computing nodes in IoT networks. *Cyber-Physical Systems*, 6(2): 96-116. https://doi.org/10.1080/23335777.2020.1716268

Siriwardhana, Y., Porambage, P., Liyanage, M., Ylianttila, M. (2021). A survey on mobile augmented reality with 5G mobile edge computing: Architectures, applications, and technical aspects. *IEEE Communications Surveys & Tutorials*, 23(2): 1160-1192. https://doi.org/10.1109/COMST.2021.3061981

Tu, Y., Chen, H., Yan, L., & Zhou, X. (2022). Task offloading based on LSTM prediction and deep reinforcement learning for efficient edge computing in IoT. *Future Internet*, *14*(2), 30-49. https://doi.org/10.3390/fi14020030

Xie, M., Huang, Z. F., Sun, H. (2024). Multi-user fine-grained task offloading scheduling strategy under cloud-edge-device collaboration. Telecommunications Science, 40(04), 107-121. http://kns.cnki.net/kcms/detail/11.2103.TN.20240508.1420.010.html.

Xue, J., Shen, B. (2023). Dung beetle optimizer: A new meta-heuristic algorithm for global optimization. *The Journal of Supercomputing*, 79(7): 7305-7336. https://doi.org/10.1007/s11227-022-04959-6

You, Q., & Tang, B. (2021). Efficient task offloading using particle swarm optimization algorithm in edge computing for industrial internet of things. *Journal of Cloud Computing*, *10*, 1-11.

https://doi.org/10.1186/s13677-021-00256-4

Yu, Z., Gong, Y., Gong, S., Guo. Y. (2020). Joint task offloading and resource allocation in UAV-enabled mobile edge computing. *IEEE Internet of Things Journal*, 7(4): 3147-3159. https://doi.org/10.1109/JIOT.2020.2965898

Zhou, C., Luo, S. (2024). Edge video task offloading strategy based on deep reinforcement learning. *Intelligent Computers and Applications*, 14 (02): 118-123. https://cs.hit.edu.cn/

Zhang, Z., Li, C., Peng, S. L., Pei, X. T. (2021). A new task offloading algorithm in edge computing. *EURASIP Journal on Wireless Communications and Networking*, 2021(1): 17-38. https://doi.org/10.1186/s13638-021-01895-6