Computer & Robotics

Task Scheduling Algorithm in Fog Computing Layer for Optimizing Multiple Quality of Service Parameters Using Jellyfish Search Optimization

Zahra Jafari[a], Ahmad Habibizad Navin [b], Azadeh Zamanifar[c]

a Department of Computer Engineering, Science and Research Branch, Islamic Azad University, Tehran, Iranb Department of Computer Engineering, Tabriz Branch, Islamic Azad University, Tabriz, Iran

c Department of Computer Engineering, Science and Research Branch, Islamic Azad University, Tehran, Iran

Received 18 October 2024, Accepted 22 December 2024

Abstract

The fog computing layer demonstrates significant potential for processing data and executing tasks for various Internet of Things (IoT) applications that are sensitive to latency. However, the resource constraints in fog devices limit the deployment of multiple applications, primarily due to inefficient resource management and discovery mechanisms in heterogeneous IoT environments. An efficient resource allocation strategy is one of the most effective ways to enhance the quality of service (QoS) and improve system performance. However, identifying the optimal resource allocation strategy for IoT applications involving multiple QoS parameters presents a complex, NP-hard challenge. This study proposes a task scheduling algorithm called CLJSO (ConvLstm-Jellyfish Search Optimization) for the fog computing layer, designed to optimize three crucial parameters: task completion time, cost, and energy consumption. Due to the high complexity and dynamic nature of scheduling in the fog layer, combining ConvLSTM and metaheuristic algorithms is proposed as an efficient solution to improve prediction accuracy and optimal resource allocation. The task scheduling process begins with predicting the workload on machines based on resource characteristics and request volumes using a ConvLstm neural network. Subsequently, the initial population of machines is generated and input into the Jellyfish Search Optimization (JSO) algorithm to execute the scheduling. Experimental results indicate that the proposed CLJSO algorithm surpasses existing approaches regarding task scheduling efficiency within the fog layer, including CGO, AOS, CSA, JS, EHEO, FSPGSA, and HGSWC, achieving an improvement in the average objective function of 14.03% to 56.44% compared to these algorithms.

Keywords: Task scheduling, Fog computing layer, Optimization algorithms, Deep Learning, Multi-objective, Internet of Things, Metaheuristic algorithms

1. Introduction

The Internet of Things (IoT) has experienced significant growth in recent years. One of the primary drivers of this evolution is the advancement of communication technologies and global computer networks [1]. IoT is applied in various sectors, including agriculture [2], healthcare [3], transportation [4], industry [5], and many others. According to research by Cisco, the number of connected devices is expected to reach 2.5 billion by 2025 [6,7], and estimates suggest that smart connected objects will continue to increase exponentially in the coming years [8]. This growing number of users and connected smart devices generate vast amounts of data, requiring distributed architectures to process the big data and handle tasks generated by IoT users and devices [9].

IoT devices typically have limited computational and energy capacities, often necessitating data offloading to more resource-rich computing layers for further

a.habibizad@srbiau.ac.ir

processing and execution. The cloud computing paradigm offers pay-per-use computation, storage, networking, and management resources. However, IoT devices' significant volume of data and the physical distance between IoT users and cloud data centers make it difficult to meet stringent Quality of Service (QoS) requirements [10–14]. To address these challenges and optimize QoS in IoT applications, a practical approach is mapping the set of services to available computing resources to meet at least one objective. In many cases, IoT users seek to optimize more than one objective simultaneously. One popular method for multi-objective task scheduling is the weighted sum method, which considers multiple objective functions with user-defined weights [15]. Multi-objective task scheduling aims to find an optimal mapping between tasks and available computing resources while optimizing several objectives.

In a fog computing environment, there is usually no order in server performance, resource utilization, or downtime patterns. The number of IoT applications and their resource demands are also almost random. As a result, metaheuristic solutions cannot efficiently address the scheduling problem for IoT applications in fog computing environments [23]. Deep learning can play a crucial role in task scheduling for fog systems due to its strong capability to learn complex patterns from data. The key reasons for utilizing deep learning include its ability to predict the behavior of cloud and fog systems using historical data, its capacity to automate decision-making in scheduling, and its adaptive nature to adjust to environmental changes and the varying requirements of different tasks. Combining neural network approaches with metaheuristic algorithms can lead to the development of an optimized and dynamic scheduling system.

This paper proposes a task scheduling method that improves QoS by minimizing task completion time, energy consumption, and costs in the fog computing layer. The proposed approach combines profound learning predictions with metaheuristic algorithms, allowing the system to leverage previous data to enhance the performance of the metaheuristic algorithm. By utilizing data processed through deep learning, metaheuristic algorithms can navigate search spaces more efficiently, enabling the discovery of better and faster solutions.

The important contributions of this article are as follows:

1. Formulating the multi-objective task scheduling problem in fog computing environments to optimize makespan, cost, and energy consumption.
2. Proposing a hybrid metaheuristic algorithm combined with deep learning for multi-objective task scheduling in fog computing environments.
3. Predicting user request demand in the fog layer using ConvLSTM neural networks.
4. Implement the algorithm and compare it with other algorithms using various parameters.
5. Demonstrating that the hybrid algorithm outperforms other state-of-the-art algorithms.

The paper is organized as follows: Section II reviews task scheduling concepts and related works. Section III presents the proposed method for task scheduling in the fog layer. Section IV describes the implementation parameters of the proposed algorithm, the evaluation criteria, and a comparison of experimental results with existing methods. Finally, Section V provides conclusions, key findings, and potential future directions.

2. Related Works

Given the rapid adoption of IoT applications across various sectors and the limited resource capacities of IoT devices, optimizing computational resources has become crucial to maximizing their utilization. Over the past few decades, researchers have extensively studied resource allocation and task scheduling in the fog computing environment. This section reviews notable works related to task scheduling in fog computing.

Scheduling algorithms generally evaluate vital parameters such as makespan, energy consumption, cost, resource utilization, efficiency, throughput, security, and availability. Recently, many researchers from academia and industry have leveraged heuristic optimization algorithms to discover optimal solutions for resource allocation. These algorithms aim to optimize various factors, including minimizing power consumption, reducing latency in delay-sensitive applications (e.g., intelligent healthcare and smart traffic), increasing network bandwidth, improving throughput, and distributing tasks to prevent the overutilization or underutilization of fog computing resources. Population-based metaheuristic algorithms have been proposed to maximize fog computing

resource utilization and optimize task scheduling for IoT applications [15].

In [15], the authors formulated a classic weighted multi-objective model to allocate IoT services, optimizing three key scheduling parameters: makespan, cost, and energy. The results demonstrated that the hybrid GA-SA (Genetic Algorithm-Simulated Annealing) approach outperformed other advanced algorithms. In [16], the authors sought to balance energy consumption and service delay when placing services on fog nodes. They introduced the NSGA-II genetic algorithm to minimize both parameters. In [17], the authors developed a task scheduling approach using Ant Mating Optimization (AMO) to minimize makespan and energy consumption as part of the optimization problem. In [18], a multi-objective optimization algorithm was formulated for system load balancing to improve overall system performance. The proposed Opposition-based Sparrow Search with Gravitational Search Algorithm (OSS-GSA) successfully enhanced energy consumption and reduced response time, performing efficiently in terms of quick response time.

In [19], the authors proposed a multi-objective scheduling problem to optimize energy, cost, delay, and load balancing in a three-layer IoT fog-cloud architecture. They implemented the Multi-Objective Particle Swarm Optimization (MS-PSO) algorithm and compared it with advanced algorithms such as standard PSO, GA, Differential Evolution (DE), and hybrid GA-PSO. Results indicated that MS-PSO outperformed standard PSO for workflow-based scientific applications. In [20], the authors introduced a scheduling model using Adaptive Optimal Deep Learning in cloud computing, proposing a modified Butterfly Optimization Algorithm combined with a Convolutional Neural Network (CNN) for task scheduling. This model aimed to maximize throughput and minimize latency.

Tong et al. [21] introduced an artificial intelligence-based algorithm called Deep Q-learning task Scheduling (DQTS), which integrates Q-learning with deep neural networks for task scheduling optimization. In [22], a novel algorithm named QL-HEFT was proposed, combining Q-learning with the Heterogeneous Earliest Finish Time (HEFT) algorithm. This method utilizes the ranking values (Ranku) from the HEFT algorithm as intermediate rewards within the Q-learning framework, improving task scheduling efficiency.

[33], This article focuses on optimizing the placement of high-demand services near data generation sources to enhance scheduling efficiency in fog computing environments. This approach is divided into three interconnected modules. The first module is the service type estimator, responsible for distributing services to appropriate nodes; the second module is the service dependency estimator, which manages service dependencies; and the third module is resource demand scheduling, which estimates resource demand to facilitate optimal scheduling.

In [34], a fog computing scheduler for executing bag-of-tasks (BoT) IoT applications in fog environments to minimize makespan and maximize reliable execution simultaneously. To this end, a new reliability model is introduced. Then, the BoT scheduling problem is formulated as a multi-objective discrete optimization problem with makespan and reliability optimization perspectives. To solve this combinatorial problem, a multi-objective discrete cuckoo search algorithm (MoDCSA) is developed. The MoDCSA optimization algorithm leverages several discrete operators that are intelligently invoked.

In [35], mathematically formulates the task scheduling problem to optimize by reducing energy consumption and cost while improving QoS through decreasing response time and the number of deadline violations for IoT tasks. Then, a method called Energy-efficient and Deadline-aware Task Scheduling (ETFC) in fog computing is proposed, which predicts the traffic of fog nodes using a Support Vector Machine (SVM) and divides them into low-traffic and high-traffic groups. The ETFC method schedules the low-traffic section using a reinforcement learning-based algorithm and the proposed ICLA-SOA algorithm, which is based on irregular cellular learning automata, and schedules the tasks of the high-traffic section with a metaheuristic algorithm based on the Non-dominated Sorting Genetic Algorithm (NSGA-III).

[36], This paper presents an architecture that utilizes a multi-criteria scheduling algorithm and multi-level queues. The proposed architecture determines a duplication coefficient for each queue, and tasks are placed in these queues using a triple-tuple approach composed of task priority, time pressure, and workload prediction of required resources. The proposed architecture employs various methods to calculate the desired parameters. CPE is used to compute the priority of tasks. Workload prediction is carried out using Long-Short Term Memory (LSTM) neural networks.

[37], This study proposes a multi-objective virtual machine placement method to jointly minimize energy costs and scheduling. The paper presents a

modified memetic algorithm, demonstrating that this proposed approach can reduce energy costs, carbon footprint, service-level agreement (SLA) violations, and the overall response time of IoT requests.

[38], In this paper, the IGWO algorithm, an improved version of GWO that utilizes the "hill-climbing" method and chaos theory to achieve better results, is proposed. The proposed algorithm can increase the convergence speed of GWO and prevent it from getting trapped in local optima. Then, a binary version of the IGWO algorithm is introduced using various S and V functions to address the workflow scheduling problem in cloud computing data centers, aiming to minimize execution cost, makespan, and power consumption.

3. Proposed Method

This section outlines the proposed method for optimizing task scheduling in the fog computing layer.

3.1 The Framework of the Proposed Method

The task scheduling problem in fog computing is classified as NP-hard, meaning that finding an optimal solution in a reasonable time frame becomes increasingly difficult as the problem grows. Metaheuristic solutions are commonly used to address NP-hard problems, but they often rely on complete global information and require solution designers to handle variability. However, server performance, resource utilization, and downtime are unpredictable in fog and cloud computing environments. Additionally, the number of IoT applications and their corresponding resource demands are often random and volatile. For these reasons, traditional metaheuristic solutions struggle to efficiently manage the scheduling of IoT applications in fog computing environments [24].

Deep learning has emerged as a powerful tool in task scheduling for cloud and fog systems due to its ability to learn from complex patterns within large datasets. Critical advantages of deep learning in scheduling include:

Accurate Prediction: Deep learning can predict the behavior of fog and cloud systems based on historical data, such as workload patterns or the time required to execute different tasks.

Automated Decision-Making: Deep learning can automate decision-making by learning from environmental conditions and available resources, dynamically adjusting task scheduling.

Adaptive Management: Deep neural networks can adapt to changing environments and the varying requirements of different tasks, ensuring efficient scheduling.

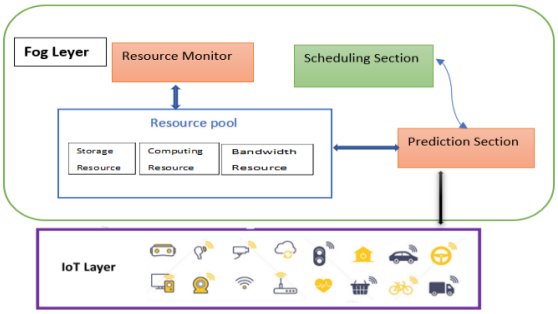

Fig. 1. Task Scheduling Framework

A more optimal and dynamic scheduling system can be developed by combining neural networks with metaheuristic algorithms.

The proposed task scheduling framework for the fog layer is depicted in Figure 1. The framework comprises two main modules: prediction, scheduler, and monitoring systems. In the fog layer, the prediction module uses ConvLSTM (Convolutional Long Short-Term Memory), a deep learning architecture, to forecast future task requests based on historical data from user requests and resource characteristics. The workload prediction generated by ConvLSTM serves as the initial input to the scheduler module.

Task scheduling is then performed using the Jellyfish Search Optimization (JSO) algorithm, an optimization method that searches for the best scheduling solutions based on workload predictions. These two components, deep learning for workload prediction and JSO for optimization, work in tandem to improve the efficiency of task scheduling in the fog layer.

Deep learning, intense neural networks, is highly efficient in predicting future events and is critical in managing dynamic environments like fog computing. In the proposed method, the ConvLSTM neural network is employed to predict the number of machines required to meet the demand in the fog layer.

ConvLSTM is an extension of the traditional Long Short-Term Memory (LSTM) network, which incorporates convolutional operations into the LSTM framework. This integration allows ConvLSTM to capture long-term dependencies in sequential data while leveraging convolutional layers to extract spatial features. ConvLSTM is particularly effective

for processing spatiotemporal data, making it well-suited for predicting sequences of events across various fields. Applications of ConvLSTM include computer vision, meteorology, and autonomous driving, which have demonstrated superior performance in analyzing sequential data. Its ability to understand both temporal and spatial dependencies makes it ideal for video prediction, weather forecasting, and trajectory analysis.

Compared to traditional LSTM networks, ConvLSTM manages spatial information within sequences more effectively, resulting in improved prediction accuracy and better generalization capabilities. Figure 2 provides an overview of the ConvLSTM model's process.

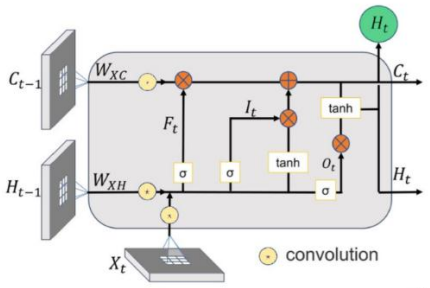

Fig.2. Structure of ConvLSTM model. [25]

3.2. Model for Predicting User Request Demand

One of the critical challenges in managing data centers is the high energy consumption associated with their operations. A practical approach to optimizing energy use in physical machines involves minimizing the number of active machines, thereby reducing overall electricity consumption. Ideally, the system should aim to run as few physical machines as possible, powering down unused machines to decrease energy usage. In the proposed method for the fog computing layer, the ConvLSTM model is used due to its superior ability to predict computational load compared to other methods.

The proposed system begins by using a prediction module to estimate the computational load, which enables efficient resource allocation to meet user demands. The ConvLSTM network forecasts future demand patterns by analyzing historical data on user requests and resource usage. $\{S_{used}, C_{used}, P_{used}\}$ The critical resources analyzed include memory consumption per number of requests (in gigabytes), processing power consumption (in Million Instructions Per Second, or MIPS), and power consumption per request (in watts). By examining these historical data points, the prediction module can accurately estimate the number of physical machines required to handle future demand.

3.3. Input Features for the ConvLSTM Model

The input features used in the ConvLSTM prediction module include the following parameters, which capture essential aspects of computational load:

Number of requests per second (MIPS): The range of requests handled by the system falls between 1,000 and 10,000 MIPS.

Memory consumption per number of requests (MB): Memory usage typically ranges between 8,000 and 32,000 megabytes.

Processing power consumption per number of requests (MIPS): This parameter varies from 500 to 8,000 MIPS, depending on the number of requests.

Power consumption per number of requests (Watts): The energy consumption per request ranges from 1,000 to 60,000 watts.

The output of this model is the number of physical machines required in the fog layer. Based on the predicted computational load, the number of fog machines typically varies between 1 and 10.

By accurately predicting user request demand, the model helps optimize resource allocation, ensuring that only the necessary number of physical machines is active, ultimately reducing energy consumption in the fog computing layer. This prediction-based approach ensures efficient management of resources while maintaining a balance between performance and energy efficiency.

3.4. Task Scheduling in the Fog Layer

The multi-objective task scheduling problem is an optimization problem that involves finding a set of optimal computing resources for various IoT tasks while optimizing more than one QoS parameter, subject to constraints. The three optimization objectives in our case are minimizing task completion time, energy consumption, and cost. The architecture assumes two layers, where the lower layer consists of intelligent hardware and IoT devices. The devices generate data and requests in this layer as tasks are sent to the fog layer. Each task t_i is described using a set of features $t_i = \{DE_i, L_i, D_i\}$, where $DE_i, L_i and D_i$ represent the deadline (seconds), task length (MI - Million Instructions), input data size (bits), and priority, respectively.

The total number of fog nodes is m, represented by $F = \{f_1, f_2, f_3, \ldots, f_m\}$.. The characteristics of each fog device are defined as $S_j^{\max}, C_j^{\max}, E_j^{\max}, P_j^{\max}$ where $S_j^{\max}, C_j^{\max}, E_j^{\max}, P_j^{\max}$ represent the maximum memory capacity (in GB), computational capacity (in MIPS), total battery capacity (in watt-hours), and power consumption (in watts), respectively.

The task scheduling problem involves assigning n IoT tasks to m fog nodes to optimize QoS parameters. Assigning a task t_i to a fog node f_j is denoted as a_{ij}, representing a mapping. We use a weighted linear combination approach to formulate the objective function, as shown in equation (1), [39].

(1)

$$\arg\min_{A_{assign}} \bigcup(A_{assign})$$

$$\bigcup(A_{assign}) = \sum_{i=1}^{n}\sum_{j=1}^{m} a_{ij} Q(i, j)$$

$$Q(i, j) = w_t \times T_{total}(a_{ij}) + w_e \times E_{total}(a_{ij}) + w_c C_{total}(a_{ij})$$

In this relation, w_t, w_e, and w_c are the three weighting coefficients, each set to $w_t = w_e = w_c = \frac{1}{3}$.

The scheduler finds a solution that is represented as a matrix $A_{assign} = [a_{ij}]_{n.m}$, where an element $a_{ij} = 1$ indicates that the j-th fog node is selected to execute the i-th task, and $a_{ij} = 0$ indicates that the node is not selected for that task.

The fog layer's proposed scheduling method uses the Jellyfish Search optimization (JSO) algorithm. This algorithm features a search mechanism based on water waves and active and passive search modes. In this context, the jellyfish represent the solutions, and the current optimal solution is the position of the prey toward which the jellyfish move [27].

The reasons for using the Jellyfish algorithm in the fog layer can be summarized as follows:

1. The Jellyfish algorithm employs a swarm intelligence approach, allowing multiple agents to explore the problem space in parallel.
2. In the initial iterations, the Jellyfish algorithm focuses on a global search to avoid local optima.
3. In the final iterations, after escaping local optima, the algorithm performs a more refined local search around the current optimal solution to increase accuracy.

The Jellyfish algorithm uses the parameter $C(t)$ to balance global and local search modes. In equation (2) [27], t is the iteration counter, and Maxt represents the maximum number of iterations for the JS algorithm.

(2)

$$\mathrm{c}\,(t) = \left|\left(1 - \frac{t}{Maxt}\right) \times (2.\mathrm{rand} - 1)\right|$$

Random Search: Equation (3) [27] defines random search, where μ is the population mean, β is a numerical parameter set to approximately 3, X^* is the best solution, and $rand(0,1)$ is a random number between 0 and 1.

(3)

$$X_i\,(t+1) = X_i\,(t) + rand(0,1) \times (X^* - \beta \times rand(0,1) \times \mu)$$

Passive Search: In equation (4) [27], U_b and L_b represent the upper and lower bounds for each solution, and γ is a numerical coefficient equal to 0.1.

(4)

$$X_i\,(t+1) = X_i\,(t) + \gamma.rand(0,1) \times (U_b - L_b)$$

Active Search: In active behavior, a jellyfish $\boldsymbol{X_i}$ randomly selects another jellyfish $\boldsymbol{X_j}$, If the fitness of $\boldsymbol{X_i}$ is better than $\boldsymbol{X_j}$, equation (5) [27] is used for movement; otherwise, equation (6) [27] is applied.

(5)

$$X_i\,(t+1) = X_i\,(t) + rand.\left(X_j\,(t) - X_i\,(t)\right)$$

(6)

$$X_i\,(t+1) = X_i\,(t) + rand.(X_i\,(t) - X_j\,(t)$$

Once the prediction model has determined the required number of fog nodes, tasks are scheduled across the fog nodes. The scheduling algorithm for the fog layer is presented in Algorithm 1. Algorithm 1 uses Algorithm 2 to create the initial population and Algorithm 3 to evaluate the objective function. As shown in Algorithm 2, the creation of the initial population is not entirely random in the proposed method. Some initial constraints are considered when selecting an appropriate initial population, which improves accuracy and reduces overall task completion time during scheduling.

The pseudocode of the CLJSO algorithm for the task scheduling

Input ⇐ N, T, F[1,...,m], T[1,...,n] ▷ The population size N and maximum number of iteration T, n, m number of tasks n, number of nodes m
Output ⇐ Jellyfish ▷ optimal mapping of tasks
Create an initial population of task to resource mappings $X_{i\ (i=1,2,...N)}$ with "Population generation Algorithm"
Initialized the current best X^*
for t=1 to T do ▷ maximum number of iteration T
  Best $X^* = X^*$ ▷ Calculate the fitness value of X^* With "Fitness Calculation Algorithm". Determining the most optimal mapping of tasks to resources as best jellyfish positions
  for i=1 to N do ▷ the population size N, each X_i
    Update $C = |(1 - t/Maxt) \times (2.rand - 1)|$
    If $(C \geq 0.5)$ then ▷ Exploration phase (jellyfish follows ocean current)
      $X^* = X_i\ (t) + rand(0,1) \times (X^* - \beta \times rand(0,1) \times \mu)$
    Else if $((C < 0.5)\ and\ (rand \geq 1 - C))$ ▷ Passive
      $X^* = X_i\ (t) + \gamma.rand(0,1) \times (U_b - L_b)$
    Else if $(C < 0.5)\ and\ (rand < 1 - C))$ ▷ Active
      $X_i\ (t)\ randomly\ selects\ X_j\ (t)$
      If $(fitness(X_i\ (t)) > fitness(X_j\ (t)))$
        $X^* = X_i\ (t) + rand.(X_j\ (t) - X_i\ (t))$
      Else
        $X^* = X_i\ (t) + rand.(X_i\ (t) - X_j\ (t))$
Return X^*

Algorithm 1: Scheduling algorithm in the fog layer

Population generation

Input Set of T and Set of F
Output $T \rightarrow F$
  for $k = 0$ *to pop_num do*
    for $i = 1$ *to size*(T) *do*
      $F[i]$ = *random choice from F*
      while $F[R[i][S]] > T[i][D]$ && $T[i][DE] > T_{total}(T[i]F[i])$ && $F[R[i]E] > E_{total}(T[i]F[i])$ && $F[R[i][C] > T_{execut}(T[i]F[i])$ *do*
        $T[i] \rightarrow F[i]$
      End while
    End for
  End for
Return P

Algorithm 2: The initial population creation algorithm

Fitness Calculation

Input: Set of T_i *and Set of* F_j
Q= []
$\omega_t = \omega_e = \omega_c = \frac{1}{3}$
For $k = 0$ *to size*$(population)$
  For $i = 0$ *to n do*
    For $j = 0$ *to m do*
      $T_{up}(a[i][j]) = \frac{T[i][D]}{BW[i][j]}$
      $T_{exe}(a[i][j]) = \frac{T[i][L]}{F[R[j][C]]}$
      $T_{total}(a[i][j]) = T_{up}(a[i][j]) + T_{exe}(a[i][j])$
      $E_{total}(a[i][j]) = T_{up}(a[i][j]) \times F[R[j][E]] + T_{exe}(a[i][j]) \times F[R[j][E]]$
      $C_{total}(a[i][j]) = T_{up}(a[i][j]) \times F[R[i][TYPE]] + T_{exe}(a[i][j]) \times T[i][Type]$
      $Q[i][j] = \omega_t \times N(T_{total}(a[i][j])) + \omega_e \times N(E_{total}(a[i][j])) + \omega_c \times N(E_{total}(a[i][j]))$
      Fit=fit + $Q[i][j]$
    End for
  End for
  Q[k]=fit
End for
Return Q

Algorithm 3: Fitness function calculation algorithm

3.5. Time Complexity Analysis of the CLJSO Algorithm

The CLJSO algorithm consists of three primary stages, each contributing to the overall time complexity.

Population Generation: In this stage, the initial population of task-to-resource mappings is created. For each solution in the population of size N, random mappings for each of the n tasks are generated. The time complexity for this stage depends on the population size N and the number of tasks n, giving a complexity of $O(n \times N)$.

Fitness Calculation: This stage calculates the fitness for each mapping of tasks to resources. The fitness calculation involves three nested loops: one for each member of the population N, one for each task n, and one for each resource m. Within the innermost loop, parameters such as transfer time, execution time, energy consumption, and cost are computed, each with a time complexity of O (1). Therefore, the total time complexity for fitness calculation is: $O(n \times m \times N)$

Main Optimization Loop in CLJSO: The main loop iterates T times to update the population and improve the optimal task-to-resource mappings. In each iteration, updates are performed for each solution in the population, resulting in a complexity of $O(T \times N)$, where T is the number of iterations, and N is the population size.

The overall time complexity of the CLJSO algorithm can be derived by summing the complexities of the three stages:

$$O(n \times N) + O(n \times m \times N) + O(T \times N)$$

In most cases, $O(n \times m \times N)$dominates the other terms. Thus, the time complexity of the CLJSO algorithm in the general case for large inputs can be approximated as:

$$O(n \times m \times N)$$

The overall time complexity $O(n \times m \times N)$indicates that the algorithm scales linearly concerning the population size N, the number of tasks n, and the number of resources m. While this complexity suggests high computational cost for large inputs, the algorithm's structure and multiple optimizations ensure that it remains scalable for complex task scheduling problems.

3.6 Impact of the Proposed Approach on Global and Local Search in the CLJSO Algorithm

The proposed approach, which integrates resource prediction using ConvLSTM and controlled population generation, positively impacts two key phases in metaheuristic algorithms, namely global search (exploration) and local search (exploitation) in the CLJSO algorithm.

Impact on Global Search (Exploration):

1. Resource prediction using ConvLSTM helps to adjust the initial search space based on real resource requirements. This directs the global search to focus on a constrained search space with a higher probability of achieving optimal solutions, rather than exploring a large and entirely random space.
2. Through controlled population generation, where machines suitable for each task are specifically chosen, the initial population is more closely aligned with the problem requirements. This process increases the accuracy of the exploration phase and improves the likelihood of finding effective combinations of resource allocation and task assignments in the initial stages of the algorithm.

Impact on Local Search (Exploitation):

1. The optimized initial population achieved through targeted selection allows the algorithm to focus its local search on more promising regions of the search space in subsequent phases. This optimization of the initial selection enables the algorithm to concentrate computational resources on refining and improving assignments that were closer to optimal from the outset.
2. This approach leads to an increased convergence speed for the algorithm, as it spends less time on refining poor solutions and moves more quickly toward optimal solutions. As a result, local search can prioritize adjustments to reasonable and effective allocations rather than unrelated or inefficient combinations.

3.7 Analysis of the Proposed Method

In this study, an innovative approach for task scheduling in dynamic environments is presented, consisting of two key stages: resource prediction using ConvLSTM and task scheduling optimization with the CLJSO algorithm. This combination improves both accuracy and efficiency in resource allocation and task management.

Resource Prediction Using ConvLSTM:

A ConvLSTM model is used to predict the number of required resources before applying the CLJSO algorithm to enhance resource allocation accuracy. This model leverages historical data and workload patterns to estimate resource needs according to actual demand. Advantages of this stage: Efficiency: By accurately predicting required resources, the CLJSO search space is reduced, leading to faster convergence and lower computational burden. Accuracy: Resource prediction helps avoid under- or over-allocation issues, ensuring that tasks are assigned to appropriate resources. This ultimately increases the scheduling precision and overall quality.

Controlled Population Generation:

In the population generation algorithm, instead of generating an initial population randomly, the ability of machines to perform specific tasks is first evaluated, and only machines capable of handling a given task are selected as viable candidates. Advantages of this stage: Improved Scheduling Accuracy: Ensuring tasks are assigned only to suitable machines results in a more rational and accurate initial population, which accelerates the convergence process. Reduced Search Space: By eliminating unsuitable allocations from the initial population, the algorithm can focus on more probable solutions, improving both speed and accuracy while also decreasing processing time.

Combining ConvLSTM resource prediction with controlled population generation in the CLJSO algorithm has resulted in the following improvements:

1. Higher Resource Allocation Accuracy: With accurate estimation of resource requirements and initial selection of suitable machines, scheduling precision is significantly improved.
2. Reduced Computational Load: By limiting the search space, the algorithm converges more quickly to optimal solutions, reducing computational time.
3. Enhanced Adaptability in Dynamic Environments: This approach adapts effectively to real-time changes, optimizing resource allocation in line with actual requirements.

4. Analysis

In this section, the proposed method is analyzed and evaluated, and the experimental results are presented, followed by a comparison with similar methods.

4.1 Implementation Parameters

This section outlines the experimental setup and configuration of parameters used during the experiments. The performance of the proposed method was evaluated and compared with existing scheduling algorithms using Python 3.8. All experiments were conducted on a personal computer with the specifications in Table 1.

Table 1
Characteristics of the test environment

CPU	Intel Core i7-8700K
RAM	64GB
Operation System	Ubuntu 18.04
Graphics Processing Unit	NVIDIA GTX 1080 Ti
Programming Language	Python 3.8
Python Libraries	Pandas, Keras, Scikit-learn, NumPy, Matplotlib
Learning Framework Deep	TensorFlow 2.4

Table 2 shows the configuration of hosts in the fog environment.

Table 2
the configuration of hosts in the fog environment

Sr.No.	Simulation Parameters	Value
1	Number of Fog Data Centers	1
2	Number of Fog Nodes	10
3	The Processing Rate of Fog Nodes	[40000 80000] MIPS
4	Cloud Nodes Memory Capacity (MB)	8000-16000
5	Fog Energy Consumption Rate (Watts per sec)	1000-6000
6	Fog CPU Usage Cost (per sec)	0.1
7	Fog Memory Usage Cost (per sec)	0.01
8	Fog Bandwidth Usage Cost (per sec)	0.001
9	Fog Bandwidth Capacity	100(Mbps)
10	Requests Per Second	1000-10000 MIPS

Synthetic datasets were used to compare the proposed method with existing algorithms. The synthetic workload consisted of 1,000 tasks generated using a uniform distribution. The task lengths varied between 1,000 and 20,000 million Instructions (MI), and the tasks were assumed to be independent and non-preemptive.

Table 3 presents the specific parameter settings for the proposed method and the comparison algorithms. It provides a detailed breakdown of the critical settings used for the algorithms under evaluation.

Table 3
Parameters settings of comparative algorithms

Algorithm	Parameter	value
HGSWC	a	[-1,1]
	f	2
	FADs	0.2
	P	0.5
	Iterations	100
CGO	α, β	[0,1]
	Iterations	100
AOS	Maximum Number of layers	5
	Foton Rate	0.1
	Iterations	100
CSA	□	2
	α	4
	ρ	1
	β	3
	Iterations	100
FSPGSA	Initial Temperature	30
	Cooling coefficient	10
	GA Iterations	60
	Mutation Probability	0.2
	Stopping time for SA	5s
	Iterations	100
EHEO	α1	1
	α2	2
	GP	0.5
	F	0.8
	CR	0.9
	Iterations	100
Jellyfish Search	β	3
	γ	0.1
	Iterations	100

Table 4 contains the critical parameters and settings used to train the ConvLSTM model. It outlines the parameters related to the model architecture, including the number of layers and neurons, and the training process settings, such as the number of epochs, batch size, and learning rate. The table also lists the hyperparameters used to optimize and evaluate the model's performance.

Table 4
Parameters and settings of ConvLSTM model training

Parameters	Description	Value
Architecture Parameters		
Dropout	Percentage of randomly dropped neurons to prevent overfitting	0.2
Activation Function	Activation function for hidden layers	Relue
output_activation	output layer	Softmax
Fit Parameters		

batch_size	Size of the batch for each training step	256
epochs	Number of iterations for training	100
CallBacks	Functions for predicting or evaluating events	ModelCheckPoint() CSVLogger() EarlyStopping() ReduceLROnPlateau()
Compile Hyperparameters		
Optimizer	Optimizer for updating weights during training	Adam(learning_rate=0.001)
Loss	The loss function for calculating error between predicted and actual values	sparse_categorical_crossentropy
metrics	Metric that measures the proportion of correctly predicted instances out of the total .instances	accuracy

4.2 Evaluation Metrics

The metrics used to compare the proposed method with similar scheduling strategies are presented in this section.

Task Completion Time (Makespan)

Makespan is the most well-known metric for evaluating the quality and efficiency of task-scheduling algorithms. It is defined as the completion time of the last task in a set of tasks.

If the time required to execute the task $t_i$on fog node f_j is denoted by a_ij, then the total time required to execute the task $t_i$on fog node f_j Is mathematically expressed as [39].

(7)

$$T_{total}(a_{ij}) = T_{up}(a_{ij}) + T_{execute}(a_{ij})$$

Here, $\boldsymbol{T_{up}}$Corresponds to the amount of data uploaded from the IoT device to the fog or cloud node. In the equation above, $\boldsymbol{T_{up}}$ is defined as the ratio of the task data size $\boldsymbol{D_i}$ to the bandwidth capacity $\boldsymbol{BW_{ij}}$ of the transmission channel between the IoT device and the fog node $\boldsymbol{f_j}$, as formulated in equation (8)[39].

(8)

$$T_{up}(a_{ij}) = \frac{D_i}{BW_{ij}}$$

The execution time of the task $\boldsymbol{t_i}$ on fog node $\boldsymbol{f_j}$ is defined as the amount of computation required for the task $\boldsymbol{t_i}$ divided by the computational capacity of the fog node $\boldsymbol{f_j}$, Which is mathematically expressed in equation (9)[41].

(9)

$$T_{execute}(a_{ij}) = \frac{L_i}{C_j^{\max}}$$

If there are n tasks and m fog nodes, the total time required to execute all tasks is expressed in equation (10), [39], [41]. In this equation, if the element $\boldsymbol{a_{ij} = 1}$, then fog node j-th is selected to execute the task $\boldsymbol{t_i}$; otherwise, if $\boldsymbol{a_{ij} = 0}$, That fog node is not selected. $\boldsymbol{T_{total}(a_{ij})}$ represents the time required to execute the task $\boldsymbol{t_i}$ on virtual machine $\boldsymbol{f_j}$.

(10)

$$T_{total} = \begin{bmatrix} T_{total}(a_{11}) & T_{total}(a_{12}) & \dots & T_{total}(a_{1j}) \\ T_{total}(a_{21}) & T_{total}(a_{22}) & \dots & T_{total}(a_{2j}) \\ \dots & & & \\ T_{total}(a_{i1}) & T_{total}(a_{i2}) & \dots & T_{total}(a_{ij}) \end{bmatrix}$$

$$T_{total} = \sum_{i=1}^{n}\sum_{j=1}^{m} a_{ij} T_{total}(a_{ij})$$

Total Energy Consumption

Similar to the modeling of time, the total energy consumed for executing a task t_i, Which is to be assigned to the fog node f_j, is defined as the sum of the energy consumed for transferring task t_i to f_j Moreover, the energy is consumed for its execution. The total energy consumed is mathematically expressed as [39].

(11)

$$E_{total}(a_{ij}) = E_{up}(a_{ij}) + E_{execute}(a_{ij})$$

This relationship models the total energy consumed for executing a task. t_i assigned to the fog node f_j, Combining the energy required for data transfer to the fog or cloud and the energy for task execution. Equation (12)[39] formulates the total energy consumed.

(12)

$$E_{total}(a_{ij}) = E_{up}(a_{ij}) + E_{execute}(a_{ij})$$

If n tasks and m fog nodes exist, the total energy required to execute all tasks is represented in equation (13) [39][41]. Here, $E_{total}(a_{ij})$denotes the energy needed to execute the task $t_i$on virtual machine $\boldsymbol{f_j}$.

(13)

$$E_{total} = \begin{bmatrix} E_{total}(a_{11}) & E_{total}(a_{12}) & \dots & E_{total}(a_{1j}) \\ E_{total}(a_{21}) & E_{total}(a_{22}) & \dots & E_{total}(a_{2j}) \\ \dots & & & \\ E_{total}(a_{i1}) & E_{total}(a_{i2}) & \dots & E_{total}(a_{ij}) \end{bmatrix}$$

$$\boldsymbol{E_{total}} = \sum_{i=1}^{n}\sum_{j=1}^{m} \boldsymbol{a_{ij}} \boldsymbol{E_{total}}(\boldsymbol{a_{ij}})$$

Total Cost

The total cost of executing a task t_i, which is to be assigned to the fog node $\boldsymbol{f_j}$, is defined as the sum of the costs for data transfer, memory usage, and execution. The total cost is mathematically expressed as [40].

(14)

$$C_{total}(a_{ij}) = C_{cpu}(a_{ij}) + C_{memory}(a_{ij}) + C_{bandwidth}(a_{ij})$$

The cost of executing a task t_i assigned to the fog node $\boldsymbol{f_j}$ Is formulated in equation (15) [40].

(15)

$$C_{cpu}(a_{ij}) = T_{execute}(a_{ij}) \times p_{ij}$$

Where $\boldsymbol{p_{ij}}$ Is the cost per unit of time for executing a task $\boldsymbol{t_i}$ on fog node $\boldsymbol{f_j}$. The memory cost is defined as [40].

(16)

$$\boldsymbol{C_{memory}}(\boldsymbol{a_{ij}}) = \boldsymbol{T_{execute}}(\boldsymbol{a_{ij}}) \times \boldsymbol{m_{ij}}$$

Where m_{ij} Represents the cost of memory usage. The bandwidth cost is given by [40].

(17)

$$C_{bandwidth}(a_{ij}) = T_{execute}(a_{ij}) \times b_{ij}$$

where b_{ij} Is the cost of bandwidth per unit of time for executing a task $\boldsymbol{t_i}$ on fog node $\boldsymbol{f_j}$.

If n tasks and m fog nodes exist, the total cost required to execute all tasks is represented in equation (18). Here, $C_{total}(a_{ij})$ denotes the cost of executing the task $\boldsymbol{t_i}$ on virtual machine $\boldsymbol{f_j}$.

(18)

$$C_{total} = \begin{bmatrix} C_{total}(a_{11}) & C_{total}(a_{12}) & \dots & C_{total}(a_{1j}) \\ C_{total}(a_{21}) & C_{total}(a_{22}) & \dots & C_{total}(a_{2j}) \\ \dots & & & \\ C_{total}(a_{i1}) & C_{total}(a_{i2}) & \dots & C_{total}(a_{ij}) \end{bmatrix}$$

$$\boldsymbol{C_{total}} = \sum_{i=1}^{n}\sum_{j=1}^{m} \boldsymbol{a_{ij}} \boldsymbol{C_{total}}(\boldsymbol{a_{ij}})$$

Performance Improvement Rate (PIR)

The PIR is a metric that estimates the percentage improvement in performance for each developed method compared to the baseline methods from related works. It is defined by equation (19) [41].

(19)

$$PIR(\%) = \frac{Y - Y'}{Y'} \times 100$$

In this equation, Y' represents the fitness value obtained by the proposed scheduling method, and Y Represents the fitness value obtained by each compared method.

5. Evaluation and Comparison

The simulation results of the proposed method are presented in two sections:

5.1 Results of the Predictor Module Evaluation

All experiments were conducted in a consistent hardware and software environment to ensure reproducibility and fairness. The hardware specifications are detailed in Table 1. For evaluating the predictor module, we used 70,000 random data points for training, 15,000 for validation, and 15,000 for testing across the models: LSTM, GRU, ConvGRU, and ConvLSTM. The dataset consists of time series data, capturing the temporal patterns of the request rate per second (MIPS) and the corresponding resource demands for the predictor module. The module output determines the number of machines required in the fog layer.

The overall training process was conducted in three phases: data preprocessing, model training, and model evaluation, as follows:

Data Preprocessing

Data Cleaning: We carefully examined the dataset values and used interpolation techniques to correct outliers, ensuring data accuracy and consistency. Additionally, given that the created datasets were imbalanced, we applied the SMOTE (Synthetic Minority Over-sampling Technique) to balance the data and prevent the model from biasing towards the majority class, thus improving prediction accuracy.
Data Standardization: We employed min-max normalization to normalize the data, ensuring consistency in the scale of data dimensions.
Data Sorting: The data was sorted chronologically. Time-series data must be arranged by date and time to preserve the chronological order. Otherwise, models cannot correctly utilize the temporal relationships between data points.
Data Splitting: The entire dataset was split into three parts: 70% for training, 15% for validation, and 15% for testing. Due to the temporal dependencies, the data was not split randomly. Instead, newer data was allocated for testing and validation, while older data was used for training.
Time Windowing: The intervals were set to 1 second, with a time window of 5. We thoroughly examined the data splits to ensure that no future information from the test set was used during the training or validation phases.

Model Training

In the training process, we specified the model architecture and hyperparameters. Our model architecture was meticulously designed to address prediction complexities. The number and types of layers used in the models are shown in Figure 3. The hyperparameters related to the architecture, compile, and fit processes are provided in Table 4.

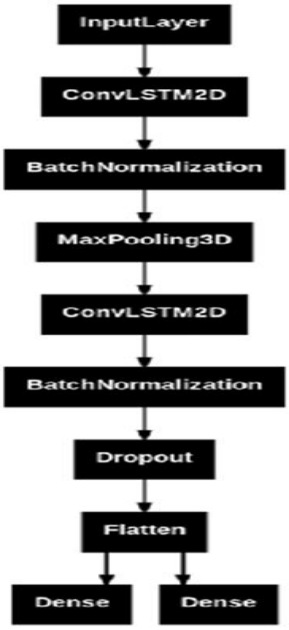

Fig3. ConvLSTM Model Architecture

Model Evaluation

We employed cross-validation and used performance metrics such as Accuracy, AUC, F1 Score, and Recall evaluating the models. A detailed comparison of the performance of LSTM, GRU, ConvGRU, and ConvLSTM models is shown in Table 5. The results indicate that the ConvLSTM model consistently outperformed the other models across all metrics, showing superior capability in predicting resource requirements.

Table 5
Evaluation of LSTM, GRU, ConvGRU, and ConvLSTM models

Model	Accuracy	Recall	F1 Score	AUC
LSTM	91.62	81.01	81.41	80.7
GRU	84.09	85.07	87.26	84.18
ConvGRU	86.44	85.33	80.25	85.62
ConvLSTM	94.30	91.33	88.70	91.28

5.2 Results of the Scheduler Module

This section presents experimental results that verify the performance of the proposed method in both fog and cloud computing layers. The evaluation metrics include task completion time, total energy consumption, and total cost for varying numbers of tasks. The proposed method's PIR (Performance Improvement Rate) is compared with similar methods. The results are compared with eight widely

used scheduling algorithms: JS [27], HGSWC [29], Chaos Game Optimization (CGO) [30], Atomic Orbital Search (AOS) [31], Chameleon Swarm Algorithm (CSA) [32], Enhanced Hybrid Equilibrium Optimizer (EHEO) [25], and FSPGSA [15]. The algorithm configurations are summarized in Table 3, and the characteristics of the fog nodes are provided in Table 2. To ensure a fair comparison, the population size for all algorithms was set to 100.

In Figure 3, the Makespan metric is compared across the algorithms JS, AOS, FSPGSA, CLJSO, EHEO, HGSWC, CSA, and CGO, with task counts ranging from [100 to 1,000] in the fog layer. The experimental results demonstrate that the proposed CLJSO algorithm consistently performs better than all other algorithms in the fog layer. Furthermore, algorithms with lower complexity, such as JS, AOS, and CSA, performed better in more straightforward cases compared to more complex algorithms like FSPGSA and EHEO.

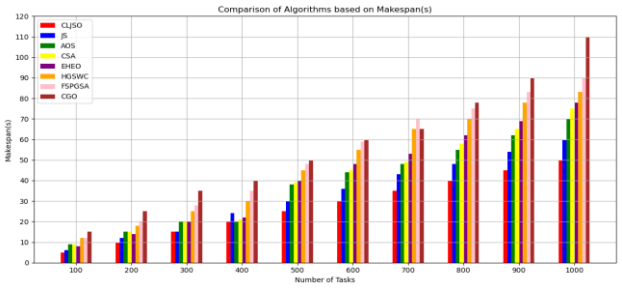

Fig. 4. Makespan comparison of the proposed method with meta-heuristic methods in the artificial data set in the fog layer

Figure 5 illustrates the energy consumption of the algorithms JS, AOS, FSPGSA, CLJSO, EHEO, HGSWC, CSA, and CGO in the fog layer with varying task numbers. As the number of tasks increases, overall energy consumption rises due to the longer processing times required, resulting in higher energy usage. For smaller task sets, the difference in energy consumption between CLJSO and the other algorithms is minimal. However, as the task numbers increase, the difference becomes more significant. CLJSO exhibits substantially lower energy consumption, which can be attributed to its predictor model for determining the necessary number of machines. This highlights the improvement in energy efficiency of the CLJSO algorithm.

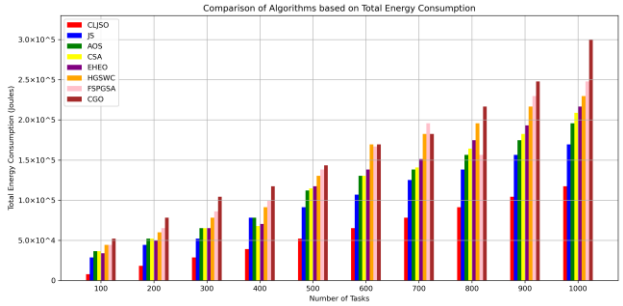

Fig. 5. Comparison of the energy of the proposed method with meta-heuristic methods in the artificial data set in the fog layer

Figure 6 presents the cost comparison for different algorithms as the number of tasks increases. The results indicate that CLJSO achieves the lowest cost in the fog layer.

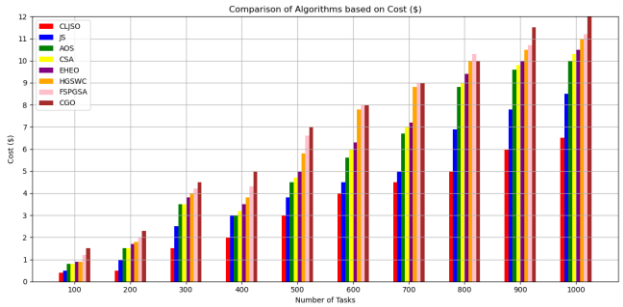

Fig. 6. Comparison of the cost of the proposed method with meta-heuristic methods in the artificial data set in the fog layer

To provide a more precise evaluation of the results, Table 6 shows the average improvement in the objective function (in percentage) achieved by CLJSO in the fog layer compared to its competitors. For example, CLJSO improved the average objective function by 48.87%-56.44% compared to the CGO algorithm and performed approximately 14.03%-21.98% better than JS.

Table 6
Comparison of PIR (%) average fitness function for different workloads in the fog layer

No of Tasks	CGO	FSPGSA	HGSWC	EHEO	CSA	AOS	JS
200	48.87	35.09	32.04	29.54	20.98	19.09	14.03
400	50.44	36.01	34.93	29	25.34	21.87	16.87
600	51.17	37.79	35.09	29.95	26.86	23.09	18.76
800	54.42	39.01	36.04	30.76	27.76	25.76	19.54
1000	56.42	40.47	37.01	34.45	29.87	26.87	21.98

6. Conclusion

The proposed method leveraged the ConvLSTM neural network to predict the number of machines required in the fog computing layer, and the tasks were scheduled using the CLJSO algorithm. Experimental results demonstrated that the proposed method outperformed algorithms such as JS, AOS, FSPGSA, EHEO, HGSWC, CSA, and CGO regarding cost, task completion time (Makespan), and energy consumption. The PIR (%) metric, which compared the average objective function with other approaches, confirmed that the proposed method achieved superior results.

The comparison results of total task execution time (makespan) demonstrated that the CLJSO algorithm significantly outperformed other algorithms. On average, CLJSO achieved a reduction of 15.19% compared to JS, 27.92% compared to AOS, 30.05% compared to CSA, 31.53% compared to EHEO, 43.71% compared to HGWSWC, 48.16% compared to FSPGSA, and 53.32% compared to CGO. These results highlighted CLJSO's superiority in optimizing task scheduling and reducing makespan compared to other algorithms.

The CLJSO algorithm demonstrated significant energy reduction compared to other algorithms. Specifically, CLJSO achieved a reduction of 52.73% relative to JS, 57.57% relative to AOS, 60.57% relative to CSA, 66.92% relative to EHEO, 68.92% relative to HGWSWC, 69.35% relative to FSPGSA, and 75.00% relative to CGO. These results highlighted the superior performance of CLJSO in optimizing energy consumption in task scheduling compared to other methods.

The results indicated that the CLJSO algorithm achieved a cost reduction of 23.22% compared to the JS algorithm, 38.15% compared to AOS, 40.14% compared to CSA, 42.71% compared to EHEO, 48.14% compared to HGWSWC, 50.52% compared to FSPGSA, and 52.82% compared to CGO. This demonstrated the effectiveness of CLJSO in minimizing costs compared to the other algorithms.

For 1000 tasks, the results showed that the CLJSO algorithm outperformed the other algorithms in terms of average improvement in the objective function. Specifically, CLJSO achieved an average improvement of approximately 56.42% over CGO, approximately 40.47% over FSPGSA, approximately 37.01% over HGWSWC, approximately 34.45% over EHEO, approximately 29.87% over CSA, approximately 26.87% over AOS, and approximately 20.98% over JS. These findings underscored the effectiveness of CLJSO in optimizing the objective function for 1000 tasks.

7. Future work

However, the proposed method has some limitations:

1. Metaheuristic algorithms are inherently non-deterministic, meaning they do not always provide the optimal solution in every iteration.
2. These algorithms may suffer from getting trapped in local optima during the task scheduling process.
3. Prediction models and deep learning algorithms require a training phase for accurate predictions, which can be time-consuming.

Addressing these challenges could be a focus of future research. Enhancing the robustness of metaheuristic algorithms to avoid local optima and improving the efficiency of the training process are potential areas for future work.

References

[1] Chai, F., Zhang, Q., Yao, H., Xin, X., Gao, R., & Guizani, M. (2023). Joint Multi-task Offloading and Resource Allocation for Mobile Edge Computing Systems in Satellite IoT. IEEE Transactions on Vehicular Technology.

[2] Pamuklu, T., Syed, A., Kennedy, W. S., & Erol-Kantarci, M. (2023). Heterogeneous GNN-RL Based Task Offloading for UAV-aided Smart Agriculture. IEEE Networking Letters.

[3] Subhan, F., Mirza, A., Su'ud, M. B. M., Alam, M. M., Nisar, S., Habib, U., & Iqbal, M. Z. (2023). AI-enabled wearable medical internet of things in the healthcare system: A survey. Applied Sciences, 13(3), 1394.

[4] Atiq, H. U., Ahmad, Z., Uz Zaman, S. K., Khan, M. A., Shaikh, A. A., & Al-Rasheed, A. (2023). Reliable resource allocation and management for IoT transportation using fog computing. Electronics, 12(6), 1452.

[5] Ai, Z., Zhang, W., Li, M., Li, P., & Shi, L. (2023). A smart collaborative framework for dynamic multi-task offloading in IIoT-MEC networks. Peer-to-Peer Networking and Applications, 16(2), 749-764.

[6] cisco, Cisco IoT Solutions, 2020, [Online;] https://www.cisco.com/c/en_in/ solutions/internet-of-things/overview.htm. (Accessed 19 July 2021).

[7] E. Lamarre, Internet of Things, 2020, [Online;] https://www.mckinsey.com/ featured-

insights/internet-of-things/how-we-help-clients. (Accessed 19 July 2021).

[8] Leo John, F., Lakshmi, D., & Kuncharam, M. (2023). Introduction to the Internet of Things: Opportunities, Perspectives and Challenges. Smart Grids and Internet of Things: An Energy Perspective, 1-34.

[9] Hamdan, S., Almajali, S., Ayyash, M., Salameh, H. B., & Jararweh, Y. (2023). An intelligent edge-enabled distributed multi-task learning architecture for large-scale IoT-based cyber-physical systems. Simulation Modelling Practice and Theory, 122, 102685.

[10] F. Bonomi, R. Milito, J. Zhu, S. Addepalli, Fog computing and its role on the internet of things, in Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing, 2012, pp. 13–16.

[11] H. Ning, F. Farha, Z.N. Mohammad, M. Daneshmand, A survey and tutorial on ‘‘connection exploding meets efficient communication’’ in the Internet of Things, IEEE Internet Things J. 7 (11) (2020) 10733–10744.

[12] H. Li, X. Li, W. Wang, Joint optimization of computation cost and delay for task offloading in vehicular fog networks, Trans. Emerg. Telecommun. Technol. 31 (2) (2020) e3818.

[13] I.M. Abbadi, Toward trustworthy clouds’ internet scale critical infrastructure, in International Conference on Information Security Practice and Experience, Springer, 2011, pp. 71–82.

[14] M. Al-Khafajiy, T. Baker, H. Al-Libawy, Z. Maamar, M. Aloqaily, Y. Jararweh, Improving fog computing performance via fog-2-fog collaboration, Future Gener. Compute. Syst. 100 (2019) 266–280.

[15] Apat, Hemant Kumar and Sahoo, Bibhudutta and Goswami. (2024). A hybrid meta-heuristic algorithm for multi-objective IoT service placement in fog computing environments. IEEE Access, Future Generation Computer Systems 154 (2024) 266–280.

[16] M. Abbasi, E.M. Pasand, M.R. Khosravi, Workload allocation in iot-fog-cloud architecture using a multi-objective genetic algorithm, J. Grid Comput. (2020) 1–14.

[17] S. Ghanavati, J. Abawajy, D. Izadi, an energy-aware task scheduling model using ant-mating optimization in the fog computing environment, IEEE Trans. Serv. Comput. 15 (4) (2020) 2007–2017.

[18] V. Gowri, B. Baranidharan, Multi-objective hybrid load balancing based optimization algorithm for improving fog computing performance, 2023.

[19] D. Subramoney, C.N. Nyirenda, Multi-swarm PSO algorithm for static workflow scheduling in cloud-fog environments, IEEE Access 10 (2022) 117199–117214.

[20] Badri, S., Alghazzawi, D. M., Hasan, S. H., Alfayez, F., Hasan, S. H., Rahman, M., & Bhatia, S. (2023). An Efficient and Secure Model Using Adaptive Optimal Deep Learning for Task Scheduling in Cloud Computing. Electronics, 12(6), 1441.

[21] Tong, Z., Chen, H., Deng, X., Li, K., & Li, K. (2019). A Scheduling Scheme in the Cloud Computing Environment Using Deep Q-learning. Information Sciences. doi: 10.1016/j.ins.2019.10.035.

[22] Tong, Z., Deng, X., Chen, H., Mei, J., & Liu, H. (2019). QL-HEFT: a novel machine learning scheduling scheme based on a cloud computing environment. Neural Computing and Applications. doi:10.1007/s00521-019-04118-8.

[23] Aslanpour, Mohammad Sadegh and Toosi, Adel N and Cheema, Muhammad Aamir and Chhetri, Mohan Baruwal and Salehi, Mohsen Amini. (2024). Load balancing for heterogeneous serverless edge computing: A performance-driven and empirical approach. IEEE Access, Future Generation Computer Systems 154 (2024) 266–280**.**

[24] Zhang, Hui and Zhang, Weihua (2024) Application of GWO-attention-ConvLSTM Model in Customer Churn Prediction and Satisfaction Analysis in Customer Relationship Management. Heliyon, Elsevier.

[25] Rao, Muchang, and Qin, Hang (2024) Enhanced Hybrid Equilibrium Strategy in Fog-Cloud Computing Networks with Optimal Task Scheduling. Computers, Materials.

[26] Hosseinioun, P., Kheirabadi, M., Kamel Tabbakh, S. R., & Ghaemi, R. (2022). aTask scheduling approaches in fog computing: A survey. Transactions on Emerging Telecommunications Technologies, 33(3), e3792.

[27] Chou, J. S., & Truong, D. N. (2021). A novel metaheuristic optimizer inspired by the behavior of jellyfish in the ocean. Applied Mathematics and Computation, 389, 125535.

[28] Heidari, A. A., Mirjalili, S., Faris, H., Aljarah, I., Mafarja, M., & Chen, H. (2019). Harris Hawks optimization: Algorithm and applications. Future Generation Computer Systems, 97, 849-872.

[29] Abd Elaziz, M.; Attiya, I. An improved Henry gas solubility optimization algorithm for task scheduling in cloud computing. Artif. Intell. Rev. 2021, 54, 3599–3637.

[30] Talatahari, S.; Azizi, M. Chaos Game Optimization: A novel metaheuristic algorithm. Artif. Intell. Rev. 2021, 54, 917–1004

[31] Azizi, M. Atomic orbital search: A novel metaheuristic algorithm. Appl. Math. Model. 2021, 93, 657–683.

[32] Braik, M.S. Chameleon Swarm Algorithm: A bio-inspired optimizer for solving engineering design problems. Expert Syst. Appl. 2021, 174, 114685.

[33] Lin, Yongxing and Shi, Yan and Mohammadnezhad, Nazila. (2024). Optimized dynamic service placement for enhanced scheduling in fog-edge computing environments. Elsevier, Sustainable Computing: Informatics and Systems, 44, 101037. Elsevier.

[34] Hosseini, Seyed Mahyar, Shirvani, Mirsaeid Hosseini, and Motameni, Homayun. (2024). "Multi-objective discrete Cuckoo search algorithm for optimization of bag-of-tasks scheduling in a fog computing environment." Computers and Electrical Engineering, 119, 109480. Elsevier.

[35] Pakmehr, Amir, Gholipour, Majid, and Zeinali, Esmaeil. (2024). "ETFC: Energy-efficient and deadline-aware task scheduling in fog computing." Sustainable Computing: Informatics and Systems, 43, 100988. Elsevier.

[36] Abbasi, Felor Beikzadeh and Rezaee, Ali and Adabi, Sahar and Movaghar, Ali. (2023). " Fault-tolerant scheduling of graph-based loads on fog/cloud environments with multi-level queues and LSTM-based workload prediction." Computer Networks, 235, 109964. Elsevier.

[37] Abbasi-khazaei, Tahereh, and Rezvani, Mohammad Hossein. (2022). "Energy-aware and carbon-efficient VM placement optimization in cloud datacenters using evolutionary computing methods." Soft Computing, 26(18), 9287–9322. Springer.

[38] Mohammadzadeh, Ali, Masdari, Mohammad, Gharehchopogh, Farhad Soleimanian, and Jafarian, Ahmad. (2021). "Improved chaotic binary grey wolf optimization algorithm for workflow scheduling in green cloud computing." Evolutionary Intelligence, 14, 1997–2025. Springer.

[39] Movahedi, Zahra and Defude, Bruno and Hosseininia, Amir Mohammad. (2021). " An efficient population-based multi-objective task scheduling approach in fog computing systems." Journal of Cloud Computing, 10, 53. Springer.

[40] Sh Shadroo, A Rahmani, A Rezaee. "The two-phase scheduling based on deep learning in the Internet of Things", In Computer Networks, Vol 185, 11 February 2021, 107684, 2020.

[41] Attiya, I., Abualigah, L., Elsadek, D., Chelloug, S. A., & Abd Elaziz, M. (2022). An intelligent chimp optimizer for scheduling of IoT application tasks in fog computing. Mathematics, 10(7), 1100.