Journal of Soft Computing and Computer Applications

Volume 1 | Issue 1 Article 1007

2024

Optimization of Resources Allocation using Evolutionary Deep Learning

Sanaa Ali Jabber
Department of Computer Science, University of Technology - Iraq, Baghdad, Iraq,
cs.20.12@grad.uotechnology.edu.iq

Soukaena H. Hashem
Department of Computer Science, University of Technology - Iraq, Baghdad, Iraq,
soukaena.h.hashem@uotechnology.edu.iq

Shatha H. Jafer
Department of Computer Science, University of Technology - Iraq, Baghdad, Iraq,
shatha.h.jafer@uotechnology.edu.iq

Recommended Citation
Jabber, Sanaa Ali; Hashem, Soukaena H.; and Jafer, Shatha H. (2024) "Optimization of Resources Allocation using Evolutionary Deep Learning," *Journal of Soft Computing and Computer Applications*: Vol. 1: Iss. 1, Article 1007.
DOI: https://doi.org/10.70403/3008-1084.1007

Scan the QR to view the full-text article on the journal website

ORIGINAL STUDY

Optimization of Resources Allocation using Evolutionary Deep Learning

Sanaa Ali Jabber [*], Soukaena H. Hashem, Shatha H. Jafer

Department of Computer Science, University of Technology – Iraq, Baghdad, Iraq

ABSTRACT

The Bidirectional Long Short-Term Memory (Bi-LSTM) network structure enables data analysis, enhances decision-making processes, and optimizes resource allocation in cloud computing systems. However, achieving peak network performance relies heavily on choosing the hyperparameters for configuring the network. Enhancing resource allocation improves the Service Level Agreement (SLA) by ensuring efficient utilization and allocation of computational resources based on dynamic workload demands. This paper proposes an approach that integrates a Multi-Objective Evolutionary Algorithm (MOEA) with deep learning techniques to address this challenge. This approach combines the optimization capabilities of MOEA with the learning predictive models to establish a framework for resource allocation in cloud environments. Snake Optimizer Algorithm (SOA) and Genetic Algorithm (GA) are utilized to identify specific hyperparameters through optimization procedures, which serve as the foundation for setting up the Long Short-Term Memory (LSTM) network. The search process begins with a random selection of parameters. The process is iterative in nature and concentrates on the best results for the parameters, with initial random sampling done according to the results obtained. This methodology considers all areas and examines the most effective areas, which improves the effectiveness of the approach when solving the problem of moving through the hyperparameter space. The modifications that were made to the snake algorithm have the goal of optimizing resource management within the data center in cloud environment. LSTM-SOA approach achieved an accuracy level of 97% on the dataset created in the Cloudsim simulation environment. This approach is based on multi-objectives to meet the requirements of determining the optimal data center. By comparison, the LSTM-AG approach achieved an accuracy level of 87%.

Keywords: Cloud computing, Service level agreement, Long short-term memory, Snake optimizer algorithm, Genetic algorithm, Resource allocation

1. Introduction

Cloud computing goals and features include providing convenient, smooth, and on-demand access to shared computing resources. There are numerous ways that deployment

Received 16 March 2024; accepted 12 May 2024.
Available online 27 June 2024

* Corresponding author.
E-mail addresses: cs.20.12@grad.uotechnology.edu.iq (S. A. Jabber), soukaena.h.hashem@uotechnology.edu.iq (S. H. Hashem), shatha.h.jafer@uotechnology.edu.iq (S. H. Jafer).

https://doi.org/10.70403/3008-1084.1007

models are provided in cloud computing, such as Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS) [1]. Companies that provide cloud computing services to organizations and customers are called Cloud Service Providers (CSPs). These services include online storage, computing resources, databases, applications, and others in cloud computing administration and service provisioning. The Quality of Service (QoS) and Service Level Agreement (SLA) are of the utmost importance. QoS refers to the level of performance, dependability, and accessibility provided by services [2]. The QoS aspect is essential for vendors and consumers within the cloud computing domain. Providers must negotiate with consumers to agree. As explained in the SLA, to determine the QoS, the service providers must provide the service within the requirements of the service level agreement and meet the standards. It must be clarified that the service provider will face financial consequences if they do not meet the set standards for QoS [3, 4]. Guaranteeing the QoS for consumers and cloud providers is becoming vital due to the rapid growth of distributed cloud computing resources. Consequently, there emerged a necessity to achieve multiple objectives that could occasionally be conflicting [5]. Therefore, the SLA negotiation process must enhance cloud service performance, foster trust between providers and consumers, and minimize SLA violations [6]. Developing an automated negotiation system is crucial to expediting the negotiation rounds and enhancing the negotiation process, thereby reducing the communication time between the involved parties. Furthermore, the service provider may avoid necessary financial losses to maintain QoS by facilitating the exchange of information between parties. This exchange allows each party to communicate their negotiation objectives and standards, especially when there are multiple objectives to be considered. In this sense, it is crucial for parties to be aware of various factors that may arise during the negotiation process [7, 8]. The main goal of this paper is to reduce negotiation time between providers and consumers and reduce SLA violations. The primary objective of this paper is to enhance the negotiation processes and steer clear of monetary fines, all to uphold the QoS. Many researchers have studied the issue of resource allocation in cloud computing; some of them and their summarization are given in Table 1.

The authors in [9] introduced the Intelligent Multi-Agent Reinforcement Model (IMARM), a resource allocation model combining intelligent multi-agent systems and reinforcement learning. IMARM dynamically allocates resources based on changing consumer demands, leveraging the properties of multi-agent systems. The reinforcement learning component learns the optimal state of virtual machines given the environment thus enhancing the performance parameters such as power consumption and fault tolerance. Experimental results demonstrate IMARM's superiority over comparable algorithms, achieving better load balancing and execution times while maintaining efficiency and fault tolerance.

In [10], the authors presented a novel approach called Multi-Objective Genetic Algorithm (MOGANS) for dynamic resource allocation in multi-virtual machine environments, and this emphasizing stability and optimization. MOGANS comprehensively considers current and predicted application load data, relocation costs, and placement stability. Simulation results show that MOGANS did achieve longer stability times and optimizes energy savings when compared to GA-NN. MOGA-C is proposed based on MOEA/D for virtual machine distribution for the purpose of addressing limitations. Experimental simulations provide evidence that MOGA-C shows faster convergence and, at the same time, achieves comparable optimization results at the same computational scale.

The Multi-Objective Improved Cuckoo Search (MOICS) algorithm for enhancing task scheduling in a cloud computing atmosphere was proposed in [11]. The proposed algorithm aims to reduce processing time and overall costs. The methodology addresses discrete

Table 1. Summarization of resource allocation in cloud computing previous studies.

Ref.	Problem	Solution method
[9]	Inefficient resource allocation in response to dynamic demand and suboptimal virtual machine state.	Combines intelligent multi-agent systems and reinforcement learning for efficient resource allocation and virtual machine optimization.
[10]	Dynamic resource allocation in multi-virtual machine environments.	Multi-objective Genetic Algorithm for addressing limitations in virtual machine distribution.
[11]	Discrete multi-objective task scheduling, automating work assignment to cloud nodes.	Multi-Objective Improved Cuckoo Search Algorithm (MOICS) for optimizing task scheduling in cloud environments.
[12]	Management of fluctuating traffic patterns.	LSTM for dynamic resource allocation, analyzing application resource usage to optimize resource additions in near real-time.
[13]	Intersection of cloud task scheduling, distribution, and machine learning to optimize cloud environments.	Cloud Linear Regression (CLR) for efficient resource management.
[14]	Distributing workloads evenly between the clouds and the edges of the network.	XCS learning classifier systems (LCS), namely, XCS and BCM-XCS, to balance the power consumption at the edge of the network and to reduce delays in the processing of workloads.
[15]	Optimizing task scheduling in cloud computing to reduce resource usage and operational costs.	(WOA) to cloud task scheduling, enhanced with IWC for improved solution search aiming at improving the performance of a cloud system with given computing resources.

multi-objective task scheduling, automating work assignments to cloud nodes to optimize resource allocation effectively. Implementation results demonstrate that the proposed approach minimizes makespan and costs significantly compared to Modified Particle Swarm Optimization (MPSO), Bee Life Algorithm (BLA), a Time–Cost aware Scheduling (TCaS) algorithm, and Round Robin (RR) algorithm, highlighting its effectiveness in cloud task optimization.

The authors in [12] focused on implementing LSTM for dynamic resource allocation, analyzing application resource usage to optimize resource additions in near real-time. The study integrates LSTM with dynamic routing algorithms for cloud traffic optimization, demonstrating effective management of fluctuating traffic patterns. Recent comparisons with Monte Carlo Tree Search (MCTS), a method to which LSTM is often compared and whose versatility is best seen, demonstrate that LSTM can sustain its performance while significant pattern changes occur. For the load balancing techniques, as in other models designed by researchers, the proposed LSTM model provides relatively acceptable SLAs, improves the accuracy rate by 10 to 15%, and decreases the error rate of the traffic load blocking probability by 9%, 5–10%, and 2%. Future research will explore energy-efficient load balancing using heuristics and machine learning approaches in cloud data centers.

In his dissertation, the author [13] presented Cloud Linear Regression (CLR), a novel machine learning model that integrates cloud technology with linear regression principles. CLR explores the intersection of cloud task scheduling, distribution, and machine learning to optimize cloud environments. CLR achieves high predictive accuracy and response times across various scenarios, demonstrating effectiveness in handling big data environments. CLR offers efficient resource management, including task scheduling, provisioning, allocation, and availability assurance. Performance results highlight CLR's capabilities, achieving significant improvements in multitasking resource utilization, memory utilization, logical disk utilization, and bandwidth utilization.

The Extended Classifier System (XCS) and Best Classifier Memory (BCM)-XCS methods for workload balancing and reducing delays at the network edge have been presented in [14]. BCM-XCS demonstrates superiority over XCS, minimizing processing and communication delays. These methods control processing delay fluctuations significantly and reduce processing delay by 42% with moderate power consumption. Additionally, they improve battery recharge by 18% compared to state-of-the-art methods.

The authors in [15] applied the latest metaheuristic, the Whale Optimization Algorithm (WOA), for cloud task scheduling within a multi-objective optimization framework to enhance cloud system performance using available computing resources. Accordingly, they introduced an advanced approach called Improved WOA for Cloud Task Scheduling (IWC) to enhance the optimal solution search capability of the WOA-based method. The detailed implementation of IWC was presented, and simulation-based experiments demonstrated that IWC achieves better convergence speed and accuracy in finding optimal task scheduling plans compared to existing metaheuristic algorithms. Additionally, it performs better in terms of system resource utilization for both small and large-scale tasks.

The main contributions of this paper can be summarized as follows:

a. Proposing a new approach for adjusting Long Short-Term Memory (LSTM) hyperparameters to expedite the process of choosing the most suitable data center for a task, cutting down on negotiation time and boosting profits.
b. Configuring data set for LTSM training by multi-objective cloud computing resource allocation standards.
c. Increasing the ability of LSTM to correspond with sequential data and capture temporal patterns makes the proposed approach adaptable to the changing SLA and evolving requirements. The LSTM units can learn and adapt to new data patterns. This ensures that the negotiation strategies remain effective even in dynamic and fickle scenarios.

The rest of this paper is organized as follows: Section 2 explains the background of LSTM. Section 3 describes the proposed approach. Section 4 shows the testing and validation processes. Finally, Section 5 illustrates the conclusion.

2. Long Short-Term Memory architecture

In 1997, Hochreiter and Schmidhuber developed an LSTM model that can perform well on sequence learning tasks, unlike other recurrent neural networks. LSTM structure is composed of a block of interconnected cells [16]. In plain Recurrent Neural Networks (RNNs), the vanishing gradients can occur in two ways: the error gradient could show exponential growth over time, or it could diminish significantly. LSTM was firmly aimed at surpassing this problem by equipping both long-term and short-term memories. Specifically, LSTM consists of a memory cell that can retain information for a long time, ensuring the algorithm performs well in long-term dependencies and non-linear data profiles. Using gating mechanisms, LSTM manages the information flow inside the network, which helps disclose key issues and block noise [17, 18]. This encompasses voice identification, online handwriting recognition, voice synthesis, language translation, emotion detection, acoustic modeling, and speech synthesis, among other capabilities. These networks are advantageous in areas including language modeling, protein structure prediction, video and audio analysis, and human behavior analysis, as referenced in sources [19–21]. These are just a few of their numerous practical uses. Several factors impact the performance of networks, including the network structure, the learning approach employed, and the activation function utilized at each node. Nevertheless, the primary focus of neural network research is the study of learning algorithms and designs, with less attention given to investigating activation functions [21]. The activation function's value establishes the boundaries for decision-making and the overall intensity of the node's input and output signals. The activation functions may also impact the intricacy and efficacy of the networks, as well as the convergence of the algorithms [22, 23]. The careful selection of activation functions significantly impacts the network's performance.

3. The proposed approach for service level agreement negotiation

When combining LSTM with meta-heuristic optimization techniques, as shown in Fig. 1, one can solve the problem of allocating resources in cloud computing and reduce negotiation times. The main objectives of the proposed approach are to address its large scope, scalable nature, and adaptability nature in cloud environments, as shown in Algorithm 1.

Due to the dynamic nature of the cloud environment and its handling of time-series data, the LSTM network, with its feedback connections, can process entire data sequences rather than just individual data points. This makes LSTM an ideal choice for effectively handling time-series data. The proposed approach incorporated two types of LSTM architectures, unidirectional and bidirectional, to compare their performance and determine the optimal structure for the proposed approach.

First, unidirectional Long Short-Term Memory (Uni-LSTM) and Bidirectional Long Short-Term Memory (Bi-LSTM) layers are implemented. Dropout layers are then added to prevent overfitting by randomly setting a fraction of the input units to 0 during each update in training sessions. Multiple LSTM layers with decreasing units are then integrated to facilitate hierarchical feature learning, whereby higher layers can capture more abstract and complex representations.

The final layer of the proposed approach architecture consists of a dense layer with a softmax function. The softmax function gives a series of probability values, which is beneficial

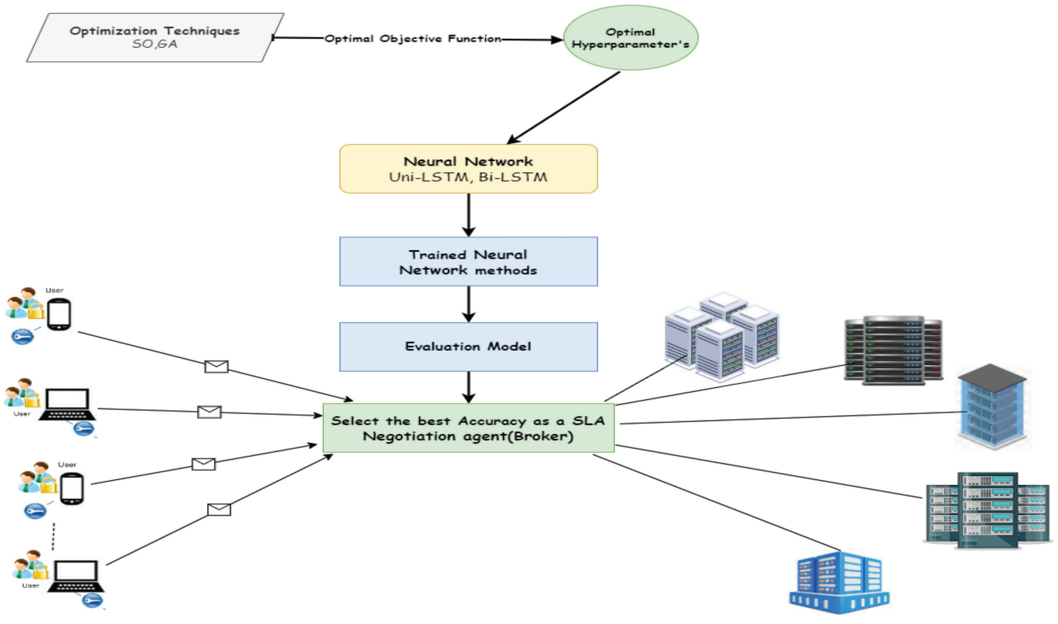

Optimization Techniques
SO,GA
Optimal Objective Function
Optimal Hyperparameter's
Neural Network
Uni-LSTM, Bi-LSTM
Trained Neural Network methods
Evaluation Model
Select the best Accuracy as a SLA Negotiation agent(Broker)
User
User
User
User

Fig. 1. General architecture of the proposed approach.

Algorithm 1. SLA negotiation.

```
Input: Task characteristic, Datacenter (Dc) characteristic
Output: mapping (Task, Dc)
1. Training the LSTM network on the generated dataset
2. For all Tasks in the Task list
    a. Dc-ID = production-STM (Task, Dcs)        //mapping Tasks to the optimal Dc
    b. If (Dc−ID != null)                        //available resource for this Task
        i. Update mapping (Task, Dc-ID)
        ii. Transmit the task to the selected Dc
    c. Else
           Update the task status to (failed)    //Tasks failed because no resource is
                                                 available
    d. End If
3. End For
4. Update the Task list
5. Repeat step 2 until the Task list is empty
6. End
```

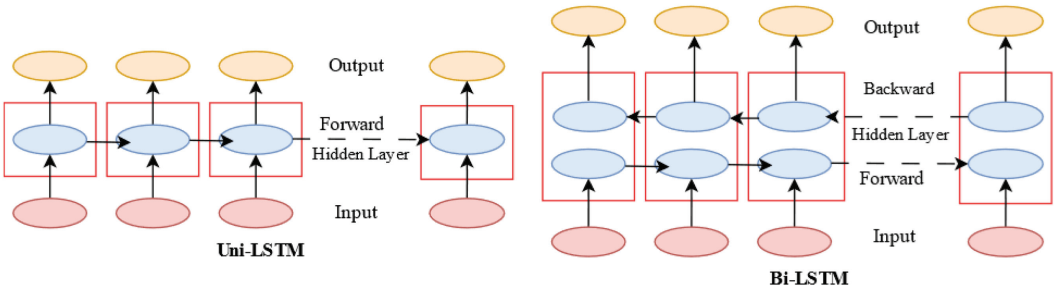
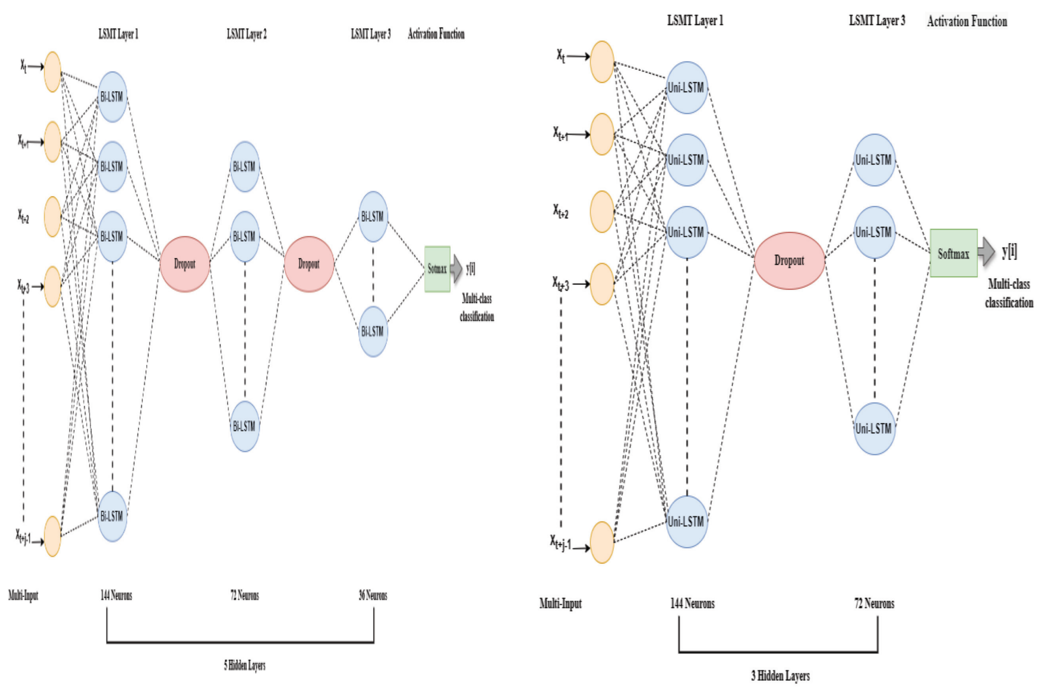


Fig. 2. Uni-LSTM vs. Bi-STM architecture.

for solving multi-class classification tasks. Adam Optimizer was chosen to compile the approach due to its computational efficiency and low memory requirements, making it well-suited for the proposed approach. During the training phase, early stopping was included to improve the performance of the proposed approach. This feature monitors a specified metric, specifically the validation loss, over several epochs to prevent overfitting. Fig. 2 illustrates the difference between Uni-LSTM and Bi-LSTM architectures. Fig. 3 shows the proposed approach of the LSTM network architecture for SLA algorithms.

3.1. Creating a dataset for training multi-objective methods

The multi-objective LSTM model is designed to deal with a series of complicated cases where more than three objectives are involved. The new approach has been developed to allocate tasks based on task requirements while at the same time aiming to balance loads between various data centers concerning four factors, which include the Cost of Executing a Task (CET), the Network Delay (DelayN), the Data Center Workload (DCload) and geographical location. To train the proposed network for improving service level agreements (SLA), developing a specific dataset that aligns with the system's requirements based on multitasking operations is essential. After deploying and configuring the virtual environment data centers and virtual devices, mathematical modeling was used to build these datasets. It was then used to build a dataset for the service level agreement (SLA) improvement phase. This process went through several training iterations where features were removed one by one to find the most closely linked attributes that were related to task



a. Bi-LSTM network Architecture for SLA algorithm

b. Uni-LSTM network Architecture for SLA algorithm

Fig. 3. The LSTM network architecture for the SLA algorithm.

distribution across the data centers. During each of the training stages the evaluation was made depending on the system accuracy. This approach aimed to reduce task execution time and enhance the service level agreement (SLA). The dataset consists of boundless attributes, contains 19 columns, and is utilized as the input source while training the proposed approaches. This number is directly connected to the number of rows in the dataset and reaches 4000, reflecting the complexity of the resource allocation model as well.

3.2. Mathematical model for dataset configuration

Equations (1) to (5) are used to model the tasks mathematically. The objective function shown in Equation (6) seeks to determine the data center with the least load, the least amount of network delay, and the lowest execution cost.

The proposed approaches' data centers differ in terms of processing, memory, storage, transferring data, and power costs.

The entire task completion in the data center refers to the CET, as shown in Equation (1) [24]:

$$\begin{aligned}\mathrm{CET}(m,n) = {} & (\mathrm{Exe}(m,n)\times \mathrm{CPUcost}(n) + (\mathrm{R}(m)\times \mathrm{RAMcost}(n)) \\ & + (\mathrm{Storage}(m)\times \mathrm{Storagecost}(n)) + (\mathrm{F}(m)\times \mathrm{Bcost}(n)))\end{aligned} \tag{1}$$

where: CPUcost is the cost in dollars per second for CPU usage, R(m) is the RAM required for the task m in megabytes (MB), RAMcost is the cost in dollars per megabyte (MB) of RAM, Storage(m) is the storage required for task m in megabytes (MB), Storagecost is the cost in dollars per megabyte (MB) of storage, F(m) is the size of task m's file in kilobytes (KB), and Bcost(m) is the price per megabyte (MB) for data transport.

The delay in the network is computed based on the data center bandwidth and task data length, as shown in Equation (2):

$$\mathrm{DelayN}(m, n) = \frac{\mathrm{LT}(m)}{\mathrm{Bv}(n)} \tag{2}$$

where: DelayN (m, n) represents the delay during the transmission of the task m to data center n, IT(a) is the task length measured in million instructions (MI), and Bv(n) is the bandwidth of virtual machine n in megabits per second (Mbps).

The load of the data center is computed in Equation (3):

$$\mathrm{DCload} = \sum_{l=0}^{m} \mathrm{Hload}(l) \tag{3}$$

$$\mathrm{Hload}(l) = \left(\sum_{i=0}^{n} \frac{\mathrm{MIPSused}(i)}{\mathrm{MIPSTotal}(l)} \right) \Big/ 100 \tag{4}$$

where: DCload is the data center utilization representing the overall load on the data center, Hload(l) measures the frequency of use of the host l, MIPSused(i) represents the total number of Millions of Instructions Per Second (MIPS) used by all tasks running in a virtual machine i, MIPSTotal(l) is the total MIPS capacity available in virtual machine l, m is the number of hosts in the data center, and n is the number of virtual machines in host j.

The distance between Cloudlet and the data center is shown in Equation (5):

$$\mathrm{Distance} = \sqrt{[\mathrm{DC}_x - \mathrm{CL}_x]^2 + [\mathrm{DC}_y - \mathrm{CL}_y]^2} \tag{5}$$

where: DC_x, DC_y represents the coordinates (x, y) of the data center, CL_x, CL_y represents the coordinates (x, y) of the Cloudlet.

The objective function shown in Equation (6) aims to determine the data center with the least load, the least amount of network delay, and the lowest execution cost:

$$\mathrm{Min\ F}(xi) = 0.4 \times \mathrm{CET}(i) + 0.3 \times \mathrm{NDelay} + 0.3 \times \mathrm{DCload} + 0.5 \times \mathrm{Distance} \tag{6}$$

3.3. The genetic algorithms algorithm strategy for hyperparameter search

The Genetic Algorithm, known as GA, offers a unique approach to fine-tuning the hyperparameters of LSTM networks by drawing inspiration from natural selection and evolution. This method mirrors the concept of natural selection, where individuals with higher fitness levels are more likely to pass on their traits to future generations. As the population undergoes repeated selection, crossover, and mutation processes, it gradually moves towards hyperparameter configurations that yield optimal results for the LSTM network's specific task.

Algorithm 2. LSTM-GA optimizer.

```
Input: Current Hyperparameters
Output: Best Hyperparameters
1. Create population
      Population = Population (indv_template, size = 50)
2. Define GA operators
   a. selection = Roulette Wheel Selection
   b. crossover = Uniform Crossover (pc = 0.8, pe = 0.5)
   c. mutation = Flip Bit Mutation (pm = 0.1)
3. best-accuracy = 0
4. Create GA Engine
   a. For each iteration in range (10)
      i. Engine = GA Engine (population = population, selection = selection, crossover = crossover,
         mutation = mutation)
      ii. engine.fitness_register
            def fitness (indv)
      iii. Use hyperparameters from individual
            hyperparameters = indv
      iv. Train the model with the hyperparameters
         1. history = train Model (hyperparameters)      //call your train model function
         2. accuracy = max (hist.history [accuracy])     //extract validation accuracy
         3. If accuracy > best-accuracy
               best-accuracy = accuracy
   b. End For
5. Access the best individual
   a. best_individual = engine.best_individual ()
   b. best_hyperparameters = best_individual.solution
6. End
```

Algorithm 2 outlines termination criteria, such as reaching a specified number of generations or observing minimal improvements in fitness scores over multiple iterations, which halt the optimization procedure.

3.4. Snake movements and strategy for hyperparameter search

The number of LSTM units, dropout rates, learning rates, and batch sizes are the hyperparameters for the LSTM network. To navigate the search space effectively, the SO needs a suitable representation of these hyperparameters, where each parameter represents a dimension in the search space. The snake's location at any given time represents a unique combination of these hyperparameters.

A real snake adjusts its movements based on its prey's location. This movement is really important in the search space for the success of the SO. Just as a real snake adjusts its movements based on fitness scores, a virtual snake adjusts its path. It employs both greedy exploitation and random exploration, as shown in Algorithm 3.

3.5. Methodology of training and optimization

Training a model goes beyond just feeding it data; it involves skillfully ensuring that the model grasps the data's essence. The model's learning uses fresh and unseen data by dividing the data into distinct training and validation sets. The training procedure entails cycling through numerous epochs with specified batch sizes. Each epoch represents a full cycle of both forward and backward passes across all training instances. The training stops when the model reaches its learning limit or starts to overfit. Algorithm 4 illustrates the training model.

Algorithm 3. LSTM-SOA optimizer.

```
Input: current-hyperparameters
Output: Best Hyperparameters
1. For iteration in range (10):
    a. history = train-model (current-hyperparameters)
    b. accuracy = train-model (current-hyperparameters)
    c. Update best hyperparameters if better ones are found
        i. If accuracy > best-accuracy:
            1. best-accuracy = accuracy
            2. best-hyperparameters = current-hyperparameters
            3. Greedy Exploitation: Refine search around the best hyperparameters
            4. perturbing the hyperparameters slightly
                Current-hyperparameters
                a. Lstm-unit = max (32, best-hyperparameters[lstm-unit] + random ([−16, 16])
                b. dropout-rate = min (0.3, max (0.1, best-hyperparameters [dropout-rate] +
                   random ([−0.1,0.1]))
                c. learning-rate = min (0.005, max (0.001, best-hyperparameters [learning-rate]
                   × random ([0.5, 1.5]))
                d. batch-size = max (32, best-hyperparameters [batch-size] + random ([−16, 16]))
        ii. Else
            Random Exploration
                Current-hyperparameters
                a. Lstm-unit = random (lstm-units)
                b. dropout-rate = random (dropout-rates)
                c. learning-rate = random (learning-rates)
                d. batch-size = random (batch-sizes)
2. End For
3. return (Best Hyperparameters)
```

Algorithm 4. Training model (dataset).

```
Input: dataset
Output: model-history
Function to train a model with given hyperparameters
1.  model = Sequential ()
2.  model.add (Bidirectional (LSTM (hyperparameters [lstm-unit]), activation = "tanh"))
3.  model.add (Dropout (hyperparameters [dropout-rate]))
4.  model.add (Bidirectional (LSTM (int (hyperparameters [lstm-unit]/2), activation = "tanh"))
5.  model.add (Dropout (hyperparameters ["dropout-rate"]))
6.  model.add (Bidirectional (LSTM (int (hyperparameters [lstm-unit]/4), activation = "tanh"))
7.  model.add (Dropout (hyperparameters [dropout-rate]))
8.  model.add (Dense (1, activation = "Softmax"))
9.  opt = Adam (lr = hyperparameters [learning-rate])
10. model.compile (loss = "binary-cross-entropy", optimizer = opt, metrics = [accuracy]) early stopping =
    Early Stopping (monitor = "val-loss", patience = 3, restore-best-weights = True)    #loss =
    "sparse-categorical-cross-entropy", For SLA
11. history = model.fit (X-train, y-train, epochs = 5, batch-size = hyperparameters ["batch-size"],
12. validation-data = (X-test, y-test), callbacks = [early-stopping])
13. return model-history
End
```

4. Experimental results

The approach was implemented using CloudSim 3.0.3, which supports simulations involving multiple data centers and users. The specifications of each data center influence the generation of virtual machines and the creation of various tasks within the simulation.

Table 2. Distribution of workloads within a cloud environment using CloudSim simulator received Cloudlets size = 100.

Output								
Cloudlet ID	Data center ID	VM ID	Time	Start time	Finish time	Distance (km)	DcLoad	Status
3	5	11	0.48	0.1	0.58	12553.0	45.0	Success
50	5	12	0.59	0.1	0.69	13601.0	45.0	Success
14	5	13	0.59	0.1	0.69	10975.0	45.0	Success
2	4	10	0.83	0.1	0.93	14021.0	51.0	Success
13	4	8	0.94	0.1	1.04	1518.0	51.0	Success
20	5	13	0.41	0.69	1.1	9333.0	45.0	Success
35	4	9	1.17	0.1	1.27	11378.0	51.0	Success
64	3	4	1.27	0.1	1.37	6009.0	42.0	Success
–	–	–	–	–	–	–	–	–
5	4	7	1.28	0.1	1.38	15509.0	51.0	Success
86	5	13	0.38	1.1	1.48	2063.0	45.0	Success

Table 3. Comparison between the genetic model and the snake model.

	Genetic model summary		Snake model summary	
Layer (type)	Output shape	Param	Output shape	Param
LSTM	(None, 1, 70)	21280	(None, 1, 128)	68608
Dropout	(None, 1, 70)	0	(None, 1, 128)	0
LSTM	(None, 35)	14840	(None, 64)	49408
Dense	(None, 3)	108	(None, 3)	195
Total params	36228 (141.52 KB)		118211 (461.76 KB)	
Trainable params	36228 (141.52 KB)		118211 (461.76 KB)	

Specific modifications were made to the classes of the CloudSim simulator to accommodate the implementation of the approach and assess the approach's performance.

Table 2 illustrates the configuration of cloud infrastructure using the CloudSim simulator. It outlines the setup of three data centers, each with a different number of hosts. The table demonstrates how 100 requests are evenly distributed among these data centers in a simulated scenario. This example showcases the allocation of workloads within a cloud environment, emphasizing the effective utilization of resources.

Table 3 compares the genetic model and the snake model. The table reveals that the snake model surpasses the genetic model in terms of complexity, as it contains multiple LSTM layers and a total of 118,211 trainable parameters, whereas the genetic model contains only 36,228 trainable parameters. Both models include a dropout layer, which serves to reduce over-learning and does not contain trainable parameters. They, both, also utilize the dense layer for final classification with few trainable parameters.

Consequently, the snake model exhibits greater complexity than the genetic model, characterized by its multiple LSTM layers and higher parameter count. Moreover, based on the provided data, the snake model demonstrates superior prediction accuracy relative to the genetic model. Fig. 4 also shows the difference in prediction accuracy and loss between the genetic and snake models. It shows that, based on the given data, the snake model is more accurate than the genetic model, and the difference in loss shows that the snake model performs better than the genetic model.

The Bi-LSTM model architecture is more efficient at allocating resources compared to the Uni-LSTM model architecture. In both types of models, the LSTM-SOA model outperforms

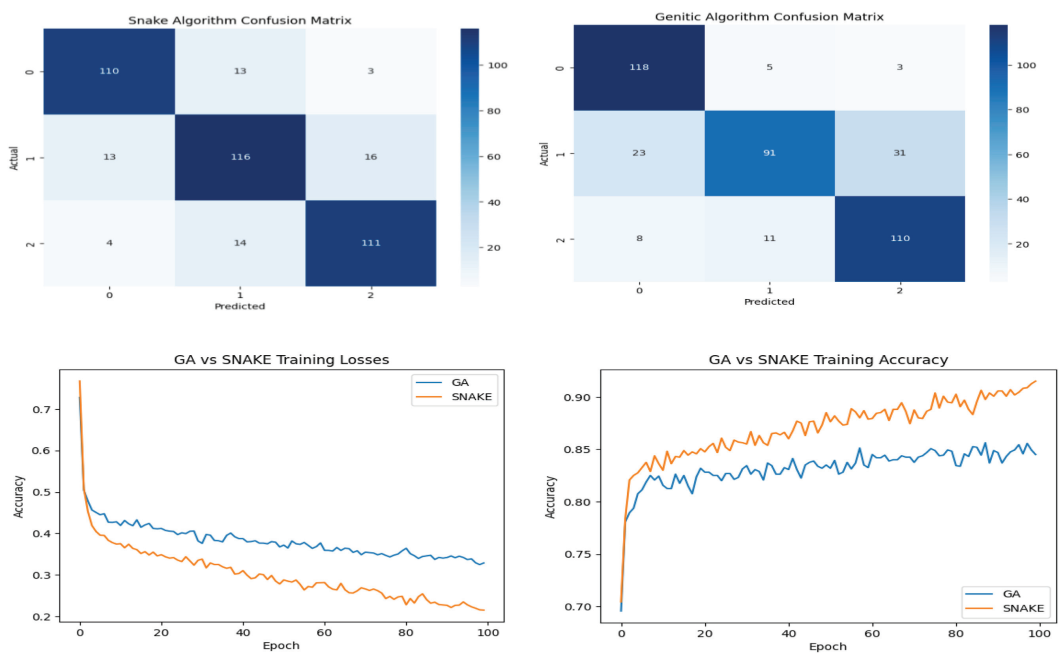


Fig. 4. Comparison of prediction accuracy and loss between genetic model and snake model.

Table 4. The Uni-LSTM-SOTA algorithm performance compared with the Uni-LSTM-GA algorithm.

	Simulation time (in seconds)		Average waiting time		Average completed time		Throughput		SLA violation rate	
Task	LSTM-SOA	LSTM-GA	LSTM-SOA	LSTM-GA	LSTM-SOA	LSTM-GA	LSTM-SOA	LSTM-GA	LSTM-SOA	LSTM-GA
50	0.87	0.97	0.28	0.29	0.11	0.14	57.47	51.34	3.6	7.4
100	1.52	1.63	0.58	0.67	0.11	0.12	65.62	61.2	4.5	6.8
150	2.18	2.41	0.86	0.99	0.11	0.13	68.68	62.24	2.8	4.7
200	2.73	3.07	1.87	1.51	0.11	0.15	73.15	65.15	2.16	8.6
250	3.29	3.84	1.97	2.52	0.11	0.16	75.99	65.1	2.9	6.2
300	4.05	4.5	2.08	2.82	0.11	0.13	74	66.67	2.6	4.6
350	4.71	5.16	2.12	2.13	0.11	0.13	74.25	67.83	2.8	3.4
400	5.37	5.71	2.19	2.48	0.11	0.15	74.43	70.05	3.2	2.3

the LSTM-GA method, as shown in Tables 4 and 5. It has lower SLA violation rates, faster performance, higher throughput, and less waiting time.

When comparing the two approaches in terms of speed, it was found that the LSTM-SOA approach in both models achieved a speed of 3.95. However, the LSTM-GA approach did not exceed a speed of 2.9 when executing 300 requests. This indicates that using the LSTM-SOA approach will lead to an increase of 30%. As for waiting time, the LSTM-SOA approach proved its ability to quickly and effectively distribute tasks to the closest and least loaded data centers, which led to faster work completion and reduced waiting time, thus increasing productivity by 20% more than the LSTM-GA. Fig. 5 illustrates the average waiting durations between the LSTM-SOA and LSTM-GA prediction approaches and assesses the throughput discrepancy between the LSTM-SOA and LSTM-GA prediction approaches.

Table 5. The Bi-LTSM-SOA algorithm performance compared with the Bi-LSTM-GA algorithm.

	Total simulation time (in seconds)		Average waiting time		Average completed time		Throughput		SLA violation rate	
Task	LSTM-SOA	LSTM-GA	LSTM-SOA	LSTM-GA	LSTM-SOA	LSTM-GA	LSTM-SOA	LSTM-GA	LSTM-SOA	LSTM-GA
50	0.75	0.85	0.2	0.27	0.1	0.11	77.47	48.9	0	0
100	1.02	1.42	0.48	0.57	0.04	0.12	85.62	61.05	0	0.02
150	1.99	2.10	0.76	0.89	0.07	0.12	88.68	62.2	0.09	0.08
200	2.10	2.13	1.17	1.2	0.07	0.15	88.15	65.45	0.02	0.05
250	2.21	2.41	1.47	1.52	0.09	0.14	94.99	66.5	0	0.1
300	2.85	3.95	1.68	1.82	0.09	0.15	94	68.35	0.04	0.05
350	2.91	3.91	2	2.13	0.09	0.15	94.25	70.3	0.09	0.1
400	3.99	4.99	2.09	2.43	0.09	0.15	94.43	72.2	0.01	0.08

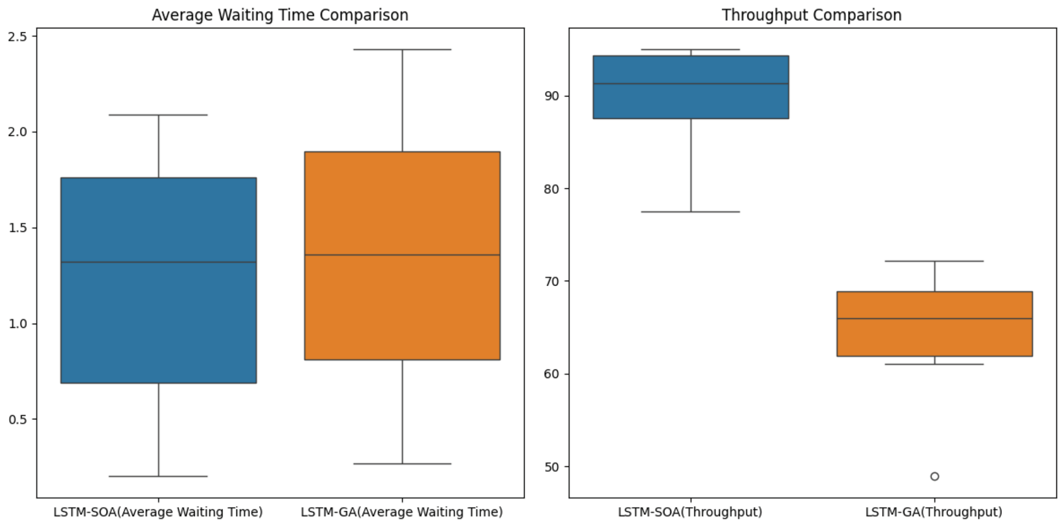


Fig. 5. Comparison of average waiting time and throughput between LSTM-SOA and LSTM-GA.

The lowest SLA violation rates illustrate the superior capability of LSTM-SOA to meet QoS standards, demonstrating its remarkable performance in this regard. Fewer violations occur because the objective function uses the deadline element as a constraint. With fewer tasks, the difference between the approaches may be small, but the difference in performance between the two approaches increases with the number of tasks performed. Fig. 6 shows the average SLA violation rate between two prediction approaches, LSTM-SOA and LSTM-GA.

5. Conclusion

Utilizing artificial intelligence to assign tasks across data centers in cloud computing enhances operational efficiency, elevates performance and QoS, and allows organizations to swiftly adapt to fluctuating workload dynamics, thereby providing a more adaptable and responsive cloud infrastructure.

Due to its bidirectional nature, Bi-LSTM model has higher capacity to understand complex patterns and dependencies in resource allocation data than unidirectional LSTM

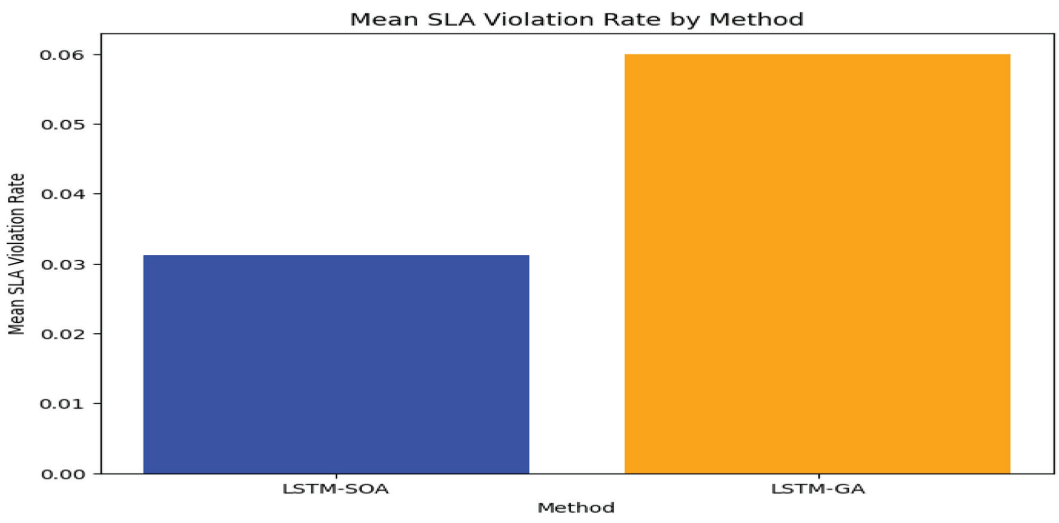


Fig. 6. The average SLA violation rate between the LSTM-SOA and LSTM-GA.

models. This improves the accuracy of predicting future resource needs and refining allocation strategies. Furthermore, Bi-LSTM minimizes the risk of information loss, which helps in making accurate decisions about resource allocation and improves the efficiency of the cloud computing environment.

The dataset created in the CloudSIM environment has proven efficiency in allocating resources in a cloud environment, taking into account multiple goals. The study identified that the geographic locations of data centers and customers play a critical role in task distribution and execution performance. Proper data center selection can reduce response times and improve overall system efficiency.

References

1. N. Kumar and S. Kumar, "Conceptual service level agreement mechanism to minimize the SLA violation with SLA negotiation process in cloud computing environment," *Baghdad Sci. J.*, vol. 18, no. 2, pp. 1020–1029, 2021, doi: 10.21123/bsj.2021.18.2(Suppl.).1020.
2. T. Edu-yaw and E. Kuada, "Service level agreement negotiation and monitoring system in cloud computing," in *Proc. 2018 IEEE 7th Int. Conf. on Adaptive Science & Technology (ICAST)*, Accra, Ghana, pp. 1–8, 2018, doi: 10.1109/ICASTECH.2018.8507131.
3. T. Lo Piparo, G. Hodosi, and L. Rusu, "Service-level agreement negotiation in cloud computing buying organizations," *Int. J. Innov. Digit. Econ.*, vol. 12, no. 3, pp. 1–16, 2021, doi: 10.4018/ijide.2021070101.
4. K. S. S. Kumar and N. Jaisankar, "An automated resource management framework for minimizing SLA violations and negotiation in collaborative cloud," *Int. J. Cogn. Comput. Eng.*, vol. 1, pp. 27–35, 2020, doi: 10.1016/j.ijcce.2020.09.001.
5. S. Tuli, S. S. Gill, P. Garraghan, R. Buyya, G. Casale, and N. R. Jennings, "START: straggler prediction and mitigation for cloud computing environments using encoder LSTM networks," *IEEE Trans. Serv. Comput.*, vol. 16, no. 1, pp. 615–627, 2023, doi: 10.1109/TSC.2021.3129897.
6. T. Faisal, J. A. O. Lucena, D. R. Lopez, C. Wang, and M. Dohler, "How to design autonomous service level agreements for 6G," *IEEE Communications Magazine*, vol. 61, no. 3, pp. 80–85, 2023, doi: 10.1109/MCOM.001.2200131.
7. D. N. Ganapathy and K. P. Joshi, "A semantically rich framework to automate cloud service level agreements," *IEEE Trans. Serv. Comput.*, vol. 16, no. 1, pp. 53–64, 2023, doi: 10.1109/TSC.2022.3140585.
8. J. F. Groote and T. A. C. Willemse, "A symmetric protocol to establish service level agreements," *Log. Methods Comput. Sci.*, vol. 16, no. 3, pp. 1–19, 2020, doi: 10.23638/LMCS-16(3:19)2020.

9. A. Belgacem, S. Mahmoudi, and M. Kihl, “Intelligent multi-agent reinforcement learning model for resources allocation in cloud computing,” *Journal of King Saud University - Computer and Information Sciences*, vol. 34, no. 6, pp. 2391–2404, 2022, doi: 10.1016/j.jksuci.2022.03.016.
10. Q. Li, F. Shi, and J. Lin, “Virtual machine resource allocation optimization in cloud computing based on multiobjective genetic algorithm,” *Computational Intelligence and Neuroscience*, vol. 2022, Art. no. 7873131, pp. 1–10, 2022, doi: 10.1155/2022/7873131.
11. S. Jaber, Y. Ali, and N. Ibrahim, “An automated task scheduling model using a multi-objective improved cuckoo optimization algorithm,” *International Journal of Intelligent Engineering and Systems*, vol. 15, no. 1, pp. 295–304, 2022, doi: 10.22266/IJIES2022.0228.27.
12. M. Ashawa, O. Douglas, J. Osamor, and R. Jackie, “Improving cloud efficiency through optimized resource allocation technique for load balancing using LSTM machine learning algorithm,” *Journal of Cloud Computing*, vol. 11, no. 1, Art. no. 87, 2022, doi: 10.1186/s13677-022-00362-x.
13. M. E. Seno, “Optimization of cloud resources provisioning and task scheduling based on machine learning,” PhD. dissertation, Computer Science Dep., University of Technology, Baghdad, Iraq, 2023.
14. M. Abbasi, M. Yaghoobikia, M. Rafiee, A. Jolfaei, and M. R. Khosravi, “Efficient resource management and workload allocation in fog–cloud computing paradigm in IoT using learning classifier systems,” *Computer Communications*, vol. 153, pp. 217–228, 2020.
15. X. Chen *et al.*, “A WOA-based optimization approach for task scheduling in cloud computing systems,” *IEEE Systems Journal*, vol. 14, no. 3, pp. 3117–3128, 2020, doi: 10.1109/JSYST.2019.2960088.
16. G. Capi, “Online robot strategy adaptation by learning and evolution,” *Journal of Intelligent Systems*, vol. 19, no. 1, pp. 1–16, 2010, doi: 10.1515/jisys.2010.19.1.1.
17. Y. Yu, X. Si, C. Hu, and J. Zhang, “A review of recurrent neural networks: Lstm cells and network architectures,” *Neural. Comput.*, vol. 31, no. 7, pp. 1235–1270, 2019, doi: 10.1162/neco_a_01199.
18. G. Park and M. Song, “Prediction-based resource allocation using lstm and minimum cost and maximum flow algorithm,” in *Proc. 2019 Int. Conf. on Process Mining (ICPM)*, Aachen, Germany, pp. 121–128, June 24–26, 2019, doi: 10.1109/ICPM.2019.00027.
19. M. Aibin, “LSTM for cloud data centers resource allocation in software-defined optical networks,” in *Proc. 2020 11th IEEE Annual Ubiquitous Computing, Electronics & Mobile Communication Conf. (UEMCON)*, New York, USA, pp. 0162–0167, October 28–31, 2020, doi: 10.1109/UEMCON51285.2020.9298133.
20. M. H. Essai Ali, A. B. Abdel-Raman, and E. A. Badry, “Developing novel activation functions based deep learning LSTM for classification,” *IEEE Access*, vol. 10, pp. 97259–97275, 2022, doi: 10.1109/ACCESS.2022.3205774.
21. A. Farzad, H. Mashayekhi, and H. Hassanpour, “A comparative performance analysis of different activation functions in LSTM networks for classification,” *Neural. Comput. Appl.*, vol. 31, no. 7, pp. 2507–2521, 2019, doi: 10.1007/s00521-017-3210-6.
22. S. Yang, X. Yu, and Y. Zhou, “LSTM and GRU neural network performance comparison study: taking yelp review dataset as an example,” in *2020 Int. Workshop on Electronic Communication and Artificial Intelligence (IWECAI)*, Shanghai, China, pp. 98–101, June 12–14, 2020, doi: 10.1109/IWECAI50956.2020.00027.
23. A. N. Toosi, R. N. Calheiros, and R. Buyya, “Interconnected cloud computing environments: challenges, taxonomy, and survey,” *ACM Comput. Surv.*, vol. 47, no. 1, pp. 1–47, 2014, doi: 10.1145/2593512.1-47.
24. S. Jaber, Y. Ali, and N. Ibrahim, “An automated task scheduling model using a multi-objective improved cuckoo optimization algorithm,” *Int. J. Intell. Eng. Syst.*, vol. 15, no. 1, pp. 295–304, 2022, doi: 10.22266/IJIES2022.0228.27.