DOI: http://dx.doi.org/10.26483/ijarcs.v16i3.7291

ISSN No. 0976-5697

Volume 16, No. 3, May-June 2025

International Journal of Advanced Research in Computer Science

RESEARCH PAPER

Available Online at www.ijarcs.info

AN EFFICIENT HYBRID FRAMEWORK FOR OPTIMAL RESOURCE ALLOCATION IN IOT-FOG-CLOUD SYSTEM

A. Priyadharshini
Ph.D. Research Scholar
Department of Computer Science
Rathinam College of Arts and Science
Coimbatore-641021, Tamil Nadu, India

S. Dhinakaran
Assistant Professor
Department of Computer Science
Rathinam College of Arts and Science
Coimbatore-641021, Tamil Nadu, India

Abstract: The amount of data produced by intelligent devices has grown significantly with the emergence of the Internet of Things (IoT) systems and other technologies. To analyze and store this data, cloud computing offers limitless processing and storage capacity. However, it suffers from a lack of geographic awareness, excessive energy usage, and high transmission delay. Additionally, it is not suitable for handling this data since the generated data is delay-sensitive. Therefore, the fog paradigm has been developed, which enables data to be analyzed in the proximity of IoT systems. However, its capacity constraints make it unsuitable for analyzing massive amounts of data. To ensure the efficient completion of delay-sensitive tasks and handle the massive amount of data generated, it is essential to integrate the fog and cloud paradigms with a common goal. This article proposes an effective Resource Allocation (RA) technique to utilize fog and cloud resources for completing delay-sensitive tasks and handling the massive amount of data generated by IoT devices. Initially, tasks in the arrival queue are categorized and assigned to appropriate resources in the cloud and fog layers based on the task guarantee ratio. A Deep Neural Network (DNN) classifier is then applied to historical allocation data to categorize new arriving tasks and assign suitable resources for execution in their respective layers. Besides, the optimal resource allocation in the fog and cloud layers is achieved using the Groupers and Moray Eels (GME) optimization algorithm, which effectively reduces the system's execution time and latency. Extensive simulations demonstrate that the DNN-GME algorithm outperforms existing algorithms in IoT-fog-cloud settings.

Keywords: IoT, cloud computing, fog computing, resource allocation, task classification, DNN, groupers and moray eels

I. INTRODUCTION

The proliferation of IoT devices has revolutionized modern information and communication technologies [1]. Beyond commonplace devices like smartphones and tablets, the Internet of Things has extended network connections to a wide range of smart equipment, including televisions, wearables, cars, and security cameras. This increase has led to an increasing number of IoT applications that produce massive data streams with critical latency requirements [2]. It is anticipated that there will be more than 75 billion connected IoT devices by 2025, indicating the market's explosive growth [3]. Additionally, according to IDC projections on the amount of data that IoT devices will generate, they will produce 73.1 ZB of data [4]. Multiple computing technologies, such as cloud, fog, and edge computing, with different communication protocols for improved connection, have become more and more necessary to tackle this situation [5].

Cloud computing is seen by smaller businesses as a powerful data management tool that can effectively store the massive amounts of data generated by IoT devices [6]. Due to the vast distance between IoT devices and cloud data centers, the process of moving massive and diverse data volumes to consolidated cloud settings results in latency and longer reaction times [7]. To meet the latency requirements of Internet of Things systems, cloud data center restrictions were successfully addressed by the creation of fog computing. By implementing network-edge services that shorten the distance between IoT data origin locations, fog computing expands the capabilities of cloud computing [8]. Fog devices are close together because they reduce processing time, which improves processing speed and speeds up application response times while using less bandwidth [9]. Fog devices continue to have less processing power and storage than cloud networks. Since its early success, fog computing has experienced consistent advancements through academic investigations and industry applications. Both cloud and fog computing have unique advantages and disadvantages that make it impossible for them to fully meet the needs of data-intensive Internet of Things applications [10]. To build an effective computing environment, fog and cloud models must be combined. Managing resources between fog and cloud for Internet of Things applications is difficult due to the variety and poor coupling of fog devices [11–12]. Consequently, a successful IoT-fog-cloud system requires an enhanced RA technique that takes fog and cloud problems into account.

To take advantage of fog and cloud paradigms, several research studies on RA based on optimization techniques and other methodologies have emerged in recent decades [13–15]. Tasks are distributed across available fog resources based on response times, bandwidth, and processing speed rates. Any unassigned tasks will be sent to the cloud for completion until there are no more free resources on the fog layer. However, due to fog resource limits and fog-based allocation of time-insensitive activities based on arrival time, these strategies cause time-sensitive jobs to queue after time-insensitive tasks for cloud processing. In fog systems, time-sensitive activities are typically severely hampered by the lack of resources.

This study proposes an efficient resource allocation system for utilizing fog and cloud resources to execute delay-sensitive operations and handle the massive volume of data created by individuals. There are two steps involved in assigning resources to tasks. First, the task guarantee ratio on the cloud and fog layers is used to categorize each task in the arrival queue. Appropriate resources are assigned to the categorized tasks according to their class layer using a GME optimization method. Second, a DNN classifier is used to categorize recently arriving tasks and match them with appropriate resources for execution in the layer of their respective classes based on prior allocation history data. With this strategy, delay-sensitive operations will benefit from the fog's faster response and higher Quality of Service (QoS), while computation- and data-intensive tasks will benefit from the cloud's infinite processing and storage capacity. Accordingly, the suggested technique can greatly reduce scheduling time and enhance overall system performance by shortening resource search period in both the fog and cloud levels.

A. Main Objectives and Contributions

The primary goals of this work can be described below:

- Create an RA framework that optimizes workload allocation between fog and cloud resources.
- To reduce task execution time, latency, and task failure owing to inadequate resource allocation while considering the job's deadline.
- Reduce total resource allocation time by categorizing activities based on historical RA history.

The following are the contributions made by this paper:

- To guarantee the best possible workload allocation across fog and cloud resources, the DNN classifier is first introduced based on the task guarantee ratio. It also minimizes the system overhead caused by a complete search for the suitable layers and resources for task execution.
- To create the best possible RA that minimizes the overall system execution time and latency, the RA problem is formulated as a multi-objective optimization dilemma and solved by introducing the GME optimization algorithm, drawing inspiration from the associative hunting between groupers and moray eels.

The subsequent portions are prepared in the following manner: Section 2 includes the related works. Section 3 discusses the proposed technique, while Section 4 demonstrates its efficacy. Section 5 highlights the findings and outlines further improvements.

II. LITERATURE SURVEY

This section provides a survey of current research on resource management and scheduling systems in fog and cloud computing environments. In [16], a layer fit technique was created to evenly distribute tasks across the fog and cloud based on their priorities. Additionally, the Modified Harris Hawks Optimization (MHHO) algorithm was used to assign the best available resource to a task inside a layer while meeting QoS criteria. However, it has high energy usage and an execution period.

In fog-IoT systems, Periasamy et al. [17] created the Efficient RA and Management with Energy Efficiency (ERAM-EE) technique. IoT devices were assigned to fog nodes via resource blocks based on the channel gain matrix of the interconnected networks. Initially, a single fog node was mapped to every IoT device via a single resource block by determining its maximum channel gain. Subsequently, the remaining resource blocks were mapped to fog nodes that had not yet been assigned, so they could be offloaded later. However, the algorithm's primary flaw was the dynamic and diverse nature of IoT devices. Additionally, the network efficiency may be impacted by CPU memory and processing power limitations.

Based on the workloads produced by a collection of nodes at the network edge, a load-aware RA method was presented for heterogeneous fog networks that minimizes execution period and latency by effectively utilizing fog resources [18]. During RA, the amount of data produced by the cluster of nodes was taken into account. However, the algorithm's resilience was limited because it only addressed rate factor sensing for RA tasks. An IoT task scheduling method utilizing the Multi-Objective Moth-Flame Optimization (MOMFO) algorithm was presented in [19]. It decreased task request completion times, throughput, and energy consumption, thereby improving the quality of fog-cloud computing-based IoT services. However, energy consumption has remained high.

A hybrid Particle Swarm Optimization with Simulated Annealing and Load Balancing (PSOSA-LB) approach was developed in [20] to optimize resource allocation in the fog-cloud scenario. The PSO velocity update was combined with a load imbalance adjustment factor that guided particles toward solutions that maximized an equitable workload distribution among available resources. However, because the LB method relies on constant monitoring of cloud and fog resource status, communication cost was high.

An energy-conscious task offloading and LB method for time-sensitive IoT fog cloud applications was presented [21]. By examining resource variety in conjunction with system parameters like network bandwidth, job size and resource consumption, and latency limitations, the fuzzy logic system was able to find offloading target layers. A Binary Linear-Weight JAYA (BLWJAYA) method was used to schedule the incoming IoT queries and assign them to cloud and fog nodes. However, it has high computational overhead and complexity.

To categorize the requests and identify the target layers for processing in fog-cloud systems, Srichandan et al. [22] introduced the Adaptive Neuro-Fuzzy Inference System (ANFIS). Additionally, such requests were scheduled at the target layer using a Chaotic Honey Badger Algorithm (CHBA). To improve the convergence of HBA, an Opposition-based Learning (OBL) scheme was used in conjunction with a chaotic mapping function. However, it is challenging to forecast the loads of each compute node to schedule requests on available nodes.

An Optimized LB (OLB) method was created by Ala'anzy et al. [23] that modifies resource allocation by taking into account processing power and traffic volumes at every fog node. However, it takes a long time to execute and consumes a lot of energy. In [24], a Dynamic Energy-and-Latency-Aware Task (DELTa) scheduling for fog-cloud environment. A multi-level queue technique was applied to prioritize tasks and find the suitable node for offloading. Then, the DELTa technique was used to effectively schedule tasks onto the chosen nodes for execution. However, it may struggle in large-scale, dynamic, and heterogeneous settings. A One-to-One-based optimizer with priority and LB (O2O-LB) method was developed [25] for fog-cloud settings. However, its latency and reaction time are high. The techniques discussed above are summarized in Table 1, which includes the assessment metrics used, the benefits, and drawbacks of the suggested approach.

The present research reveals that many solutions manage RA within IoT-fog-cloud architecture; however, numerous issues, such as excessive energy utilization, latency, and poor performance in real-world applications or heterogeneous environments, persist. Existing models fail to account for job categorization and appropriate resource management when workloads change. This disadvantage necessitates the creation of an effective task categorization algorithm using RA. The primary goal of this research is to present a novel DNN-GME method that improves resource usage and network stability across IoT, fog, and cloud paradigms.

III. Proposed Methodology

This section provides an in-depth methodology for the proposed DNN-GME algorithm. Fig. 1 depicts the visual layout of this suggested system, including the task creators, the task categorizer, the resource distributor, the fog layer, and the cloud layer resources. Initially, user-created tasks are placed in the arrival queue and classified as cloud or fog. Tasks categorized as cloud are implemented by cloud

Table I. Summary of Existing RA Techniques in IoT-Fog-Cloud Systems

Ref. No.	*Techniques*	*Merits*	*Demerits*	*Assessment metrics*
[16]	MHHO	It lessens the oversaturation in the fog layer because of increasing requirement for resources.	Energy consumption and execution cost remained high.	Mean makespan, execution cost, and energy consumption
[17]	ERAM-EE	It reduces the bandwidth consumption, response period, and latency.	A primary constraint lies in the heterogeneous and dynamic nature of IoT systems. Also, CPU memory and processing power constraints limit the network performance in heterogeneous IoT scenarios.	Energy efficiency, processing time, and response time
[18]	Load-aware RA method	It achieves low execution cost and latency compared to the classical fog and cloud paradigms.	It requires more parameters in the RA procedure to enhance the algorithm's robustness.	Network consumption, latency, and execution cost
[19]	MOMFO	It enhances the resource usage significantly.	It has low throughput and high energy consumption.	Makespan, throughput, energy consumption, and resource usage
[20]	PSOSA-LB	It reduces the execution period, energy consumption, latency, and load imbalance effectively.	It is based on continual monitoring of fog and cloud resource conditions, which results in significant communication overhead. This can be problematic in situations with low bandwidth or considerable communication delays between the fog and cloud levels.	Execution time, energy consumption, latency, and load imbalance
[21]	BLWJAYA	It minimizes the latency and energy consumption while increasing the resource usage.	It has high complexity and computation overhead.	Resource utilization, latency, energy consumption, and load balancing rate
[22]	ANFIS, CHBA, and OBL	It achieves better resource usage.	The latency and energy consumption were high.	Makespan, service delay, delay violation, service cost, resource usage, and energy consumption
[23]	OLB	It yields greater adaptability and network efficiency.	It needs to integrate machine learning algorithm to further improve decision making procedures within the fog layer.	Latency, network usage, execution time, energy consumption
[24]	DELTa	It enhances the tasks processing efficiency when reducing energy usage and increasing resource use.	It may struggle in large-scale, dynamic, and heterogeneous settings. Also, it solely focuses on independent tasks, which may impact execution efficiency and scheduling in practical cases.	Makespan, service latency, energy consumption, and resource utilization
[25]	O2O-PLB	It consistently sustains low latency and faster response periods in high dependability and low-delay applications.	Its suitability in real-world applications was low because the constant resource load threshold value was not suitable for real-time data.	Response time, latency, load imbalance, and task failure rate

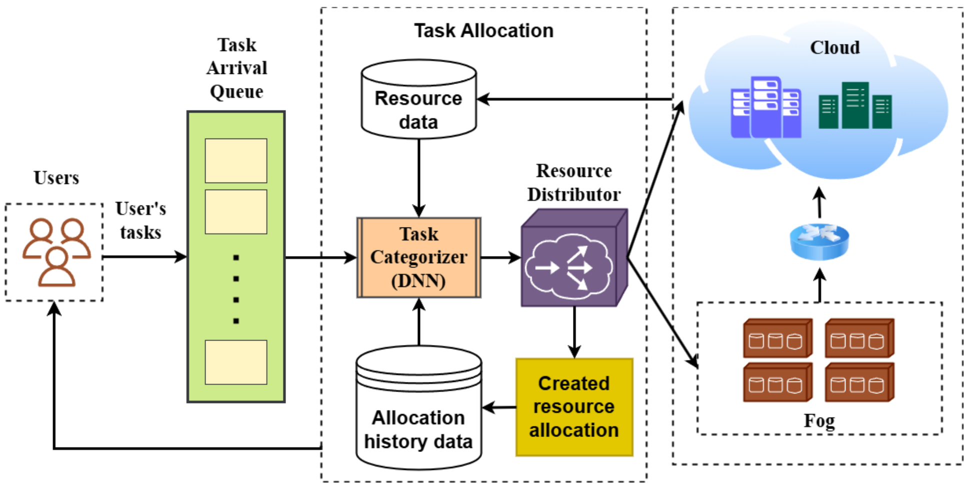

Figure 1. Visual Layout of Proposed DNN-GME-Based RA Algorithm

resources, and tasks categorized as fog are implemented by fog resources at the fog layer. The categorizer then sends the categorized tasks to the resource distributor, who allocates resources on either the cloud or fog layer. Two steps are involved in the task categorization process: (i) the tasks in the arrival queue are categorized based on the task guarantee ratio in the first step, and (ii) the newly arrived tasks are categorized using the DNN classifier based on the data from the allocation history in the second step.

A. *System Model*

The proposed system paradigm may efficiently handle IoT device data in a hierarchically organized IoT-fog-cloud context. As seen in Fig. 2, the model is composed of three primary layers: edge, fog, and cloud. Because each component layer takes advantage of its characteristics, this IoT-fog-cloud architecture offers an efficient way to handle the intricate activities of the IoT system. Additionally, it facilitates more efficient data processing by reducing latency and improving resource utilization.

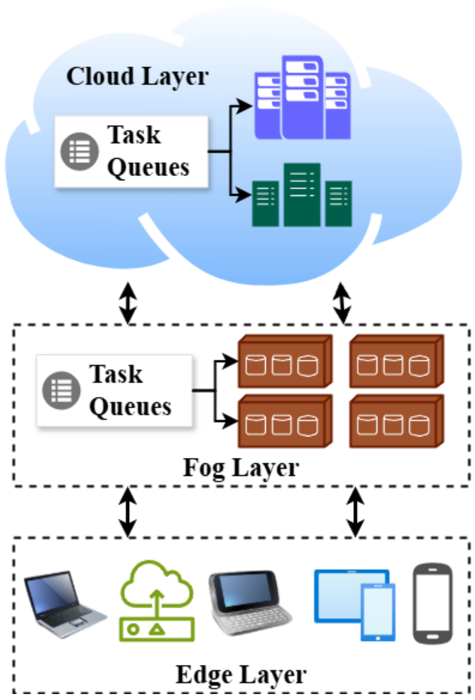

Figure 2. IoT-Fog-Cloud System Model

Depending on processing demands, the edge layer—which consists of sensors, actuators, and intelligent wearables—creates task requests and sends them to the fog or cloud layer. Servers or transitional devices, such as fog nodes, are located closer to the network edge at the fog layer to provide local processing and storage capabilities. Numerous data centers at the cloud layer are capable of providing substantial storage and processing capacity, along with sophisticated data processing capabilities. To identify the proper processing layer and allocate resources to tasks, respectively, tasks generated by an edge device are transmitted from the task categorizer and resource distributor. While time-insensitive activities are transferred to the cloud queue, time-sensitive jobs are delivered to the fog queue.

B. *Task Classification*

The process of categorizing user tasks and allocating them to the proper layer queues (cloud or fog) for execution is covered in this section. Tasks are ranked according to their QoS requirements and the processing power of each layer's available resources. Assume $V = \{v_1, \dots, v_n\}$ is a group of n Virtual Machines (VMs) in the cloud. If $v_i, i \in [1,2, \dots, n]$ is the i^{th} VM, then v_i has 3 attributes, denoted as $\langle v_i^p, v_i^m, v_i^B \rangle$, where v_i^p is the CPU processing power, v_i^m is the memory, and v_i^B is the bandwidth resources. As well, $F = \{f_1, \dots, f_m\}$ is the group of fog nodes in the fog layer. If $f_i, i \in [1,2, \dots, m]$ is the i^{th} fog node, then f_i has 3 attributes, denoted as $\langle f_i^p, f_i^m, f_i^B \rangle$, signifying the CPU processing power, memory, and bandwidth resources, respectively.

Consider $\mathcal{T} = \{\mathtt{t}_1, \dots, \mathtt{t}_k\}$ is a set of k tasks in the system arrival queue. Each task $\mathtt{t}_i, i \in [1,2, \dots, k]$ is the i^{th} task in the queue, $\mathtt{t}_i$ has also 3 attributes denoted as $\langle \mathtt{t}_i^p, \mathtt{t}_i^m, \mathtt{t}_i^d \rangle$, signifying the processing requirements of the task, data storage, and task deadline, respectively. A resource is supposed to complete a task and return the results of that execution to the user before the task's deadline passes. A task is successful if it is returned to the user before the deadline; otherwise, it is unsuccessful.

1) *Task Guarantee Ratio Categorization*

The task classifier arranges the tasks according to their processing needs in the first categorization stage. A task group is made up of functions having comparable processing needs. The task classifier uses the following formula to determine each group's task guarantee ratio on the cloud layer and the ratio on the fog layer:

$$\psi_C = \min \textstyle\sum_1^k \sum_1^n \left(\frac{\mathtt{t}_i^p, \mathtt{t}_i^m}{v_i^p, v_i^B} \right) \tag{1}$$

$$\psi_F = \min \textstyle\sum_1^k \sum_1^n \left(\frac{\mathtt{t}_i^p, \mathtt{t}_i^m}{f_i^p, f_i^B} \right) \tag{2}$$

The classifier then uses the ψ_C and ψ_F values to determine if each set of tasks is a cloud or fog. A task group is categorized as a cloud task if its ψ_F value falls between 0.6 and 1, indicating that it involves more processing and data. The task group is categorized as a fog task if the ψ_F value falls between 0.1 and 0.5. Since the layer's waiting queue will have more tasks since there aren't enough cloud resources to handle every kind of task instantly, the system is predicted to experience more task failures when ψ_C is greater than 1. The resource allocator receives the categorized tasks and chooses the optimal resource for each task within its class layer. Based on the allocation history data, the classifier uses the DNN algorithm to categorize new tasks as they come in.

2) *Deep Neural Network Classifier*

For a task $\mathtt{t}_i$ with attribute set $X = \{x_1, x_2, x_3\}$ signifying task processing requirements, memory, and task deadline. Consider M is the collection of task class $\rho_i, i = 1,2, \dots, M$. To categorize the tasks with their corresponding classes, the DNN classifier is constructed. The DNN is a descendant of the traditional Artificial Neural Network (ANN). The three levels of the DNN architecture, including the input, hidden, and output layers, are depicted in Fig. 3.

The network receives the pre-processed input data from the input layer. The neural network's input neuron count is equal to the dataset's input features. So, the input layer with D inputs is represented as:

$$X = \{x_1, \dots, x_D\} \tag{3}$$

In (3), $D \in [1,2,\dots,k]$. Since the DNN permits the addition of several hidden layers, the next layer is the hidden layer. The input X is mapped by the hidden layer using a bias(b_j) and random weights (w_i). So, inputs for the hidden layer are written as:

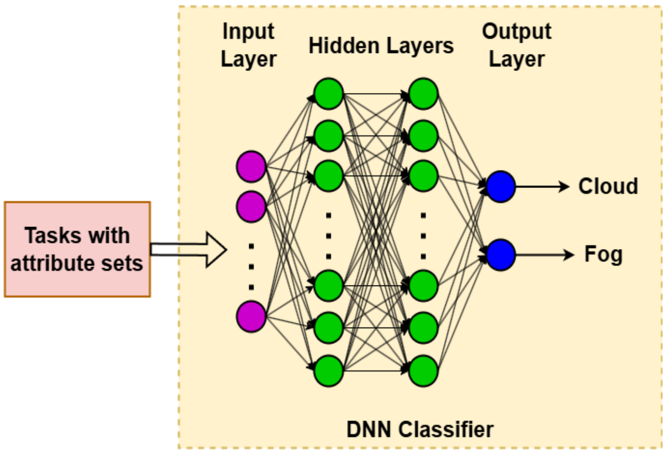

Figure 3. Task Categorization Using DNN Classifier

$$h_j = \sum_i w_i x_i + b_j \tag{4}$$

In (4), $j \in [1,\dots,l]$ is the number of hidden units in the DNN. A nonlinear activation function is linked to each hidden layer. A Rectified Linear Unit (ReLU) has improved performance and sped up the DNNs' training procedure. The elimination of the disappearing and inflating gradient issue was ReLU's primary innovation. Consequently, the hidden layer's output is stated as follows:

$$h = f(h_j) \tag{5}$$

$$Where\, f(h_j) = ReLU(h_j) \tag{6}$$

The output layer generates the DNN's outputs by processing the inputs from the hidden layer to the output layer's activation function. For task categorization, the output layer's nonlinear activation function, such as softmax, is used. It transforms inputs into a class of probabilities $(\sigma(X)_j)$. Thus, the DNN's output is written as:

$$\sigma(X)_j = \frac{e^{X_j}}{\sum_{j=1}^{l} e^{X_j}} \tag{7}$$

In (7), X represents a vector of inputs to the output layer, and $j \in [1,\dots,l]$ is the number of output units.

With this DNN configuration, the inputs to their corresponding class output (for example, cloud 1 and fog 0) are used to train the network. A large training dataset is used to train the DNN, and to reduce training mistakes, the weight of each input link is changed repeatedly. To train the network more quickly and effectively, the Adam optimizer adjusts the model parameters of the DNN. During training with the learning algorithm, these tuning parameters, also referred to as hyperparameters, are employed to regulate optimization functions and model selection. These hyperparameters determine whether the model overfits or underfits during the learning phase. The task categorization using DNN classifier is summarized in Algorithm 1.

Algorithm 1: DNN-Based Task Categorization

```
Input: Collection of new uncategorized tasks and allocation history data
Output: Task class labels
 1. Begin
 2. for(each uncategorized task $t_i$)
 3.   Create task $t_i$'s attribute vector $X = \{x_1, x_2, x_3\}$;
 4.   Construct DNN classifier;
        a. Initialize input layer, hidden layers, and output layer;
        b. Initialize bias $b$ and weights $w$;
        c. Set learning rate, training epochs, activation function, and batch size;
 5.   for(each epoch)
 6.     for(each batch)
 7.       Obtain $\sigma(X)_j$ with forward propagation;
 8.       Compute the loss function $L = \frac{1}{k}\sum_{i=1}^{k}(\sigma(X)_j - \widehat{\sigma(X)_j})^2$;
 9.       Update bias and weight values;
10.     end for
11.   end for
12. end for
13. Return task class labels
14. End
```

C. *Problem Formulation for Resource Allocation*

This section describes the modeling of task execution time and the delay that occurs between submission and the user receiving the execution result. The system is supposed to optimize the time and delay model as QoS parameters. The time and delay models are used to construct an objective function that has to be reduced.

1) *Task Execution Time Model*

The overall time required by the computing resource to finish a task's execution is known as the task's execution time on that resource. For $\mathcal{T} = \{t_1,\dots,t_k\}$ to be executed by resources within a layer (either fog or cloud), each task $t_i, i \in [1,2,\dots,k]$ has $\langle t_i^p, t_i^m, t_i^d \rangle$. Assume $R = \{r_1,\dots,r_n\}$ is the collection of resources on the layer which the task collection $\mathcal{T}$ is to be executed. Each resource $r_j, j \in [1,2,\dots,n]$ has 3 attributes denoted as $\langle r_i^p, r_i^m, r_i^B \rangle$, signifying the CPU processing power, memory, and bandwidth resources, respectively.

The execution period of the $\mathcal{T}$ on the collection of resources within the corresponding layer is determined by

$$ET = \sum_{i=1}^{k}\sum_{j=1}^{n}\left(\frac{t_i^p}{r_j^p}\right) \tag{8}$$

2) *Task Delay Model*

One of the crucial factors that must be taken into account, particularly while working on activities that are sensitive to delays, is the latency between the start and end times of jobs. The difference between the task's arrival and end times is known as the dwell time. Three delays are taken into account in this model: waiting, transmission, and execution delays, which occur when a job takes longer than expected to complete from the moment of arrival. Before being assigned to a layer resource for execution, a task is initially queued up when it is submitted to the system. For t_i from $\mathcal{T}$ to be executed by a layer resource, the waiting time (t_i^w) of the tasks is determined as the difference between the arrival time (t_i^a) and the start time (t_i^s) of the tasks as:

$$t_i^w = t_i^s - t_i^a \tag{9}$$

When a task is allocated to a cloud or fog processing layer, it must be sent to the layer via a wireless channel. Transmission from the edge layer to the fog or cloud layer causes a delay in the process. Compared to the cloud layer, which is located far from the edge, fog layer resources are closer to the end user and hence need less transmission. With a processing capacity of ψ_C, the entire cloud computing data center is regarded as a distributed computing node in this study. Consequently, the task transmission latency $\left(\mathtt{t}_i^l\right)$ is determined by

$$\mathtt{t}_i^l = \frac{\mathtt{t}_i^m}{\lambda_l} + \alpha \tag{10}$$

In (10), λ_l is the layer power and α is the network overhead factor.

A task's anticipated execution time can be met after its processing capacity has been allotted. An increased processing capacity quota to the task may shorter the task execution time, resulting in an execution delay. For $\mathtt{t}_i$ from $\mathcal{T}$ to be executed by a layer resource, the execution delay $\left(\mathtt{t}_i^{ed}\right)$ is determined as the fraction of tasks needed CPU and task assigned CPU:

$$\mathtt{t}_i^{ed} = 1 - \frac{\mathtt{t}_i^p}{r_j^p} \tag{11}$$

The total delay of $\mathtt{t}_i$ from its time it enters the system until it is finished is determined by

$$\mathtt{t}_i^{ted} = \mathtt{t}_i^w + \mathtt{t}_i^l + \mathtt{t}_i^{ed} \tag{12}$$

Therefore, minimizing the total execution time and delay of a task resource allocation x is the task allocation objective in this study, which is provided in (13).

$$f(x) = \min\left(\sum_{i=1}^{k} ET \times 0.5 + \sum_{i=1}^{k} \mathtt{t}_i^{ted} \times 0.5\right) \tag{13}$$

D. Resource Allocation Using GME Optimization Algorithm

This section describes the GME optimization algorithm to assign resources to tasks within the fog and cloud layer according to their corresponding classes. GME resembles the associative hunting behavior of groupers and moray eels [26]. Compared to the other nature-inspired optimization algorithms, like MOMFO, MHHO, etc., this GME optimization algorithm performs associative hunting to achieve a great trade-off between exploration and exploitation. Therefore, this GME algorithm is chosen in this study for optimal RA. Many animals, including chimps and lions, have cooperated while hunting. Cooperative hunting amongst animals of different species, known as associative hunting, is exceedingly unusual. Groupers and moray eels have complementary hunting strategies. Cooperation is thus mutually advantageous since it improves the chances of both species successfully acquiring prey. When the two predators collaborate, their hunting strategies complement one another, resulting in a multi-predator onslaught that is tough to avoid. The great level of cooperation between the two species distinguishes this hunting example from that of groups of animals of the same species.

The flock in GME is composed of an even number of search agents (i.e., different task-resource assignments throughput the cloud and fog layers), allowing for easy splitting into pairs. Each pair consists of two individuals: grouper fish and moray eels. Each individual serves a distinct purpose when the couple collaborates to capture the target. The grouper fish and moray eels work in four stages of hunting: Primary Search (PS), Pair Association (PA), Encircling Search (ES), and Attacking and Capturing (AC).

A population of possible solutions (i.e., particular configuration of task-resource assignments) is generated stochastically within the specified upper and lower boundaries of the issue (i.e., the maximum and minimum values of tasks and their requirements). The fitness function's best value represents the best candidate solution, while the worst value represents the worst search agent. The optimization process iteratively updates the population and takes into account the best solution found so far, aiming to converge towards an optimum solution. The first phase, PS, involves grouper fish searching for prey. The next step is selecting an eel to follow and hunt the prey amid the rocks. The PA phase completes the work, and the ES phase uses the same method. The last stage, AC, occurs when one of the two individuals, the moray eel or the grouper fish, catches the prey. Each stage is explained briefly below.

1) Primary Search

Groupers are aquatic creatures that can move quickly through water by propelling themselves forward and adjusting their fin direction. They are primarily used for exploration during the search phase in nature-inspired optimization methods, where search agents traverse the search space at random. A successful prey search enhances the algorithm's exploring capabilities. Groupers use a zig-zag swimming style to traverse a broad region while tracking potential dangers. During this phase, they search for food and move in a zig-zag pattern, assigning each fish a random location before starting. This zig-zag motion occurs when no three consecutive places of movement are in a rising or decreasing sequence. Fig. 4 demonstrates several variations of the zig-zag motion by a grouper fish in the search space.

A zigzag motion happens when no three consecutive places of the movement are in a rising or decreasing sequence. In other words, if the motion array comprises three entries (h_i, h_{i+1}, h_{i+2}) such that $h_i < h_{i+1} < h_{i+2}$ or $h_i > h_{i+1} > h_{i+2}$, the motion is not zigzagged. Under the statement, P denotes the total number of iterations in the PS, ES, and AC.

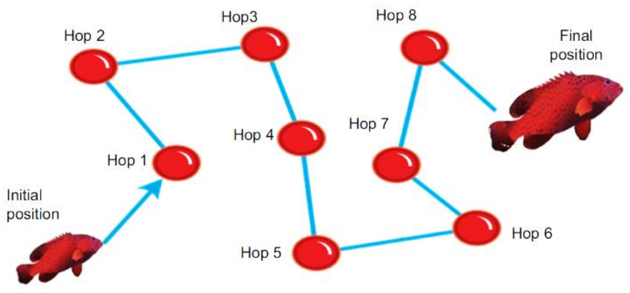

Figure 4. Zig-Zag Motion of Grouper Fish in the Search Space

So, the respective values of the number of iterations for PS, ES, and AC represented as P_{search}, P_{Enc}, and P_{AC}, which are determined by

$$P_{search} = \left\lfloor \frac{P}{3} \right\rfloor \tag{14}$$

$$P_{Enc} = \left\lfloor \frac{P}{3} \right\rfloor \tag{15}$$

$$P_{AC} = \left\lfloor P - 2 \times \frac{P}{3} \right\rfloor \tag{16}$$

Consider S is the cluster of N available search agents (i.e., groupers and moray eels), these are initially split into two equivalent clusters of groupers and eels. So, the number of agents in each cluster is determined by

$$n = \frac{N}{2} \tag{17}$$

The cluster of groupers and eels is represented by $G = \{g_1, g_2, \ldots, g_n\}$ and $E = \{e_1, e_2, \ldots, e_n\}$, respectively. The following process involves randomly distributing groupers and eels in the search space using (18), ensuring their positions align with the specified lower and upper bounds.

$$X_{ij}^{initial} = \mathrm{l}_j + rand\left(\mathrm{u}_j - \mathrm{l}_j\right), i = 1, \ldots, N; j = 1, \ldots, \tag{18}$$

In (18), $X_{ij}^{initial}$ represents the initial location of i^{th} search agent of j^{th} dimension, u_j and l_j denote the upper and lower margins of the search space, respectively, N is the number of search agents, D is the total amount of dimensions, and $rand$ is an arbitrary vector that pursues a uniform distribution, with values ranging from 0 to 1.

After that, the objective function (as defined in (13)) is computed for each grouper. The groupers start moving in a zigzag pattern to find the prey (best solution). The zigzag movement helps the groupers to explore new regions in the search space, hence increasing the algorithm's exploration potential. At the end of this phase, the optimal location for each grouper is determined using the objective function.

Consider $\vec{X}_{gmj}^{i} = \{X_{gm1}^{i}, X_{gm2}^{i}, \ldots, X_{gmQ}^{i}\}$ is the location vector of the m^{th} grouper in the i^{th} iteration, where $1 \le i \le p_{search}$, $1 \le j \le Q$, and $1 \le m \le n$, n is the total quantity of groupers, and p_{search} is the total amount of search iterations. During this stage, groupers update their location vectors after each hop, and their corresponding objective function is evaluated to determine their proximity to the possible prey. The updated position of a grouper is defined in (19), which depends on the number of hops (n), with even hops resulting in an arbitrary location greater than the current location but not exceeding the search space's maximum boundary, and odd hops resulting in an arbitrary location less than the current location but greater than or equal to the search space's minimum boundary. Thus, the optimal location of each grouper is determined by the location vector that yields the maximum value of the objective function compared to locations created across all hops of the iteration.

$$X_{gmj}^{hop+1} = \begin{cases} rand(X), & X_{gmj}^{hop} < X \le \max\left(X_{gmj}\right);\ if\ n\ is\ even \\ rand(X), & \min\left(X_{gmj}\right) \le X < X_{gmj}^{hop};\ if\ n\ is\ odd \end{cases} \tag{19}$$

2) Pair Association

Groupers are predators known for their quick bursts of speed, but their large size and unwieldy shape prevent them from catching animals hiding in narrow crevices. They use a novel strategy of seeking out moray eels to flush out their prey. Groupers and moray eels have different predatory abilities, with groupers using ambush tactics and moray eels maneuvering through coral reef openings. They work together to patrol a larger hunting area, operating in the upper water column and navigating the reef depths. When trapped, the prey has two options: remain hidden until the moray eel finds it or swim in the water, where the grouper's teeth await.

The search process involves a series of phases, starting with the PS phase, where search agents are randomly distributed within the search domain. The objective function values are calculated and recorded. The PA phase occurs after the PS, where cooperation between groupers and eels allows them to discover new areas. The grouper's intelligence and learning ability enable it to choose the best eel for hunting. Pair identification occurs between groupers and eels, using random associations, distance-based associations, and objective function associations. The ES phase follows, where search agents move towards their best position. This AC stage sees search agents converge towards the optimal solution.

3) Encircling Search

During the search phase, paired eels encircle the prey, moving independently to explore new areas. They use an underwater shimmying dance called the Grouper to eel Encircling Signal (GES) to signal their desire to hunt in groups. They may even perform an underwater headstand to show the prey's hiding place. If the eel ignores the signal, groupers aggressively approach the moray, forcing it to chase the fish out of its hiding place. The moray then attacks the prey, sharing the meal, benefiting from their associative hunting techniques.

The location vector of m^{th} grouper in Q-dimensional space is denoted as $\vec{X}_{gmj} = \{x_{gm1}, x_{gm2}, \ldots, x_{gmQ}\}$ and the location vector of m^{th} eel in Q-dimensional space is denoted as $\vec{X}_{emj} = \{x_{em1}, x_{em2}, \ldots, x_{emQ}\}$. The best solution for a prey is found in the region bounded by the grouper and the eel. In an n-dimensional space, (20) is utilized to compute the prey location according to the differences between the coordinates of a grouper and an eel in each dimension.

$$\Delta x_{mj} = X_{emj} - X_{gmj} \tag{20}$$

The distance between the grouper and the eel $\left(d_{grouper-eel}\right)$ is then calculated by

$$d_{grouper-eel} = \sqrt{\sum_{j=1}^{Q}\left(\Delta x_{mj}\right)^2} \tag{21}$$

Moreover, the coordinates of the prey in each dimension are determined as:

$$c_{mj} = X_{gmj} + \frac{L}{d_{grouper-eel}}\left(\Delta x_{mj}\right) \tag{22}$$

In (22), c_{mj} is the coordinates of m^{th} prey in each dimension, X_{gmj} is the location of a grouper, X_{emj} is the location of an eel, and L represents the distance between the grouper and prey. During the ES stage, groupers and eel move towards the prey in their own ways, as displayed in Fig. 5. The logarithmic spiral is the primary location update method for groupers. The distance between groupers and potential prey is calculated by

$$\vec{d} = \left|\vec{X}_{prey\_m}(i) - \vec{X}_{gm}(i)\right| \tag{23}$$

The location of a grouper is updated multiple times in an iteration based on the number of hops (h). The location with the best fitness value is the initial location in the second iteration, and so on until the determined number of iterations in the ES stage. The updated location depends on $\vec{d}$, constant k that measures the shape properties of the logarithmic spiral, the estimated position of the m^{th} prey at iteration $i\left(\vec{X}_{prey\_m}(i)\right)$, w, the number of encircling iterations $(P_{encricle})$, and h. This is represented by

$$\vec{X}_{gm}(h+1) = \vec{d} \times e^{kw}\cos(2\pi w) + \vec{X}_{prey\_m}(i) \tag{24}$$

$$w = 1 - \frac{2\times h}{P_{encricle}} \tag{25}$$

Sine waves, found in nature and easy to handle mathematically, are the primary position update mechanism for moray eels during the ES phase. They can create any arbitrary wave shape, making them a significant shape in nature. The eel moves in a sinusoidal wave, updating its position multiple times in iterations.

The distance between the eel and prey, is computed as:

$$\vec{\lambda}(i) = \left|\vec{X}_{em}(i) - \vec{X}_{prey\_m}(i)\right| \tag{26}$$

After that, the wave amplitude $\vec{\eta}(i)$ is calculated by

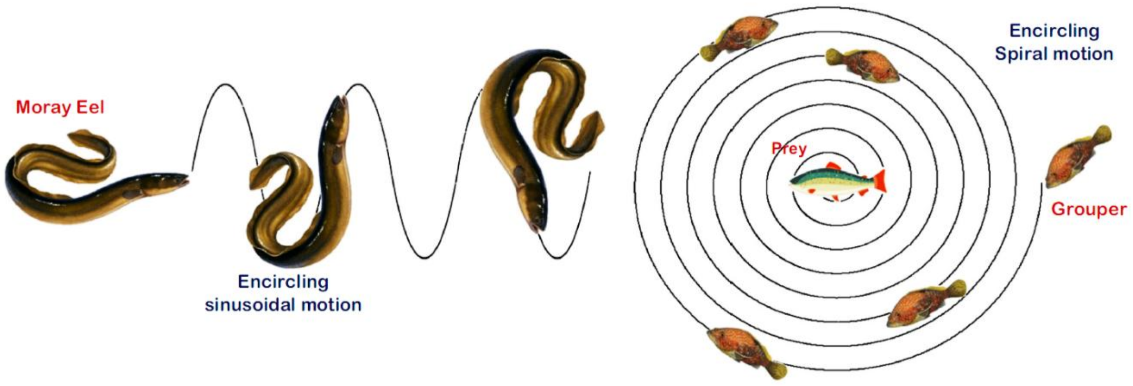

Figure 5. Encircling Strategy of Grouper and Eel

multiplying $\vec{\lambda}(i)$ by an arbitrary factor ξ, ranging from 0 to 1.

$$\vec{\eta}(i) = \vec{\lambda}(i) \times \xi \tag{27}$$

The distance between the eel and the prey is split into hops. So, the distance between the current and next hop is determined as:

$$d_{hops} = \frac{2\pi}{total\ number\ of\ hops} \tag{28}$$

The locations of the eels are updated as follows:

$$X_e^{i+1} = \alpha \times \vec{\eta}(i) \times \sin(g) + X_e^i \tag{29}$$

In (29), α is an arbitrary value ranging between 0 and 1. After the current iteration is completed, the best location of the eel from all hops is chosen as the initial location of the next iteration.

4) Attacking and Catching

At the AC phase, all search agents, whether groupers or eels, participate in the attack on the prey after accurately surrounding its location. In 2D space, the hypothesis relies on forming a circle around the prey, where the prey is located in the center of the circle.

The process of forming a circle around a predicted prey involves first assuming the location of the agent with the best fitness function as the prey's location. The distance between the prey and all other agents is then calculated. A circle is then formed, with the radius (R) representing the distance between the prey and the farthest agent. The radius of the next circle is determined by

$$R_{i+1} = (1-\mu) \times R_i \tag{30}$$

In (30), $R = \{1,2,\dots,AB-1\}$, where AB is the number of attack balls, and μ represents the shrinking ratio. The search agent will be dispersed at random within the second circle after its radius has been established. The search agents approach the target more closely with each repetition of the attack. The agents capture the prey in the final iteration of the procedure, which continues until the number of assaulting iterations is reached. The pseudocode for the GME optimization algorithm for RA in IoT-Fog-Cloud paradigm is given in Algorithm 2.

Algorithm 2: GME Optimization-Based RA Technique
Input: Tasks $\mathcal{T}$, including cloud and fog layer tasks, and resources R
Output: Best assignment of tasks to resources

```
1.  Begin
2.  for(each t_k ∈ T and r_j ∈ R)
3.    Calculate the execution time and delay using (8) & (12), respectively;
4.    Define the objective function f(x) by (13);
//GME Optimization Algorithm
5.    Initialize the set of N available search agents (groupers and eel);
6.    //PS stage
7.    Initialize the total number iterations P_search and the number of movements in the iteration (hop);
8.    for(i = 1: P_search)
9.      Split the search agents into two equivalent clusters of grouper and eels;
10.     Arbitrarily circulate the groupers across the search space;
11.     Determine the objective function f(x) for all groupers;
12.     Each grouper starts to travel in the form of zig-zag motion;
13.     Determine the new location for all groupers according to the motion in the iteration using (19);
14.     Compute the new objective function;
```

```
15.     Choose the ideal location of the grouper from every hop in the current iteration to be the initial location of next iteration;
16.     Allocate the ideal location for all groupers according to their fitness values;
17.     Descending order the groupers based on the highest fitness value;
18.     Arbitrarily circulate the moray eels in the search space and determine the fitness value for all eels;
19.   $\boldsymbol{end\ for}$
20.   //PA & ES Stage
21.   Initialize the set of available search agents of groupers and eels, total number of encircling iterations $P_{encircle}$, ideal location vector for all $j^{th}$ groupers from the PS stage, $h$ number of hops for the grouper's motion, and constant $k$;
22.   $\boldsymbol{for}(i = 1: P_{encircle})$
23.     Determine the position of the $m^{th}$ prey by (20)-(21);
24.     Compute the distance between the grouper and prey by (22);
25.     Determine the new location for all groupers and fitness value for the hop location;
26.     Choose the ideal location of the grouper from all hops in the current iteration to be the optimal location and initial location of next iteration;
27.     Update the ideal location for all groupers according to the fitness value;
28.     Determine the wavelength and wave amplitude using (26) & (27), respectively;
29.     Split the distance between the eel and prey based on the number of hops;
30.     Determine the new location for all eels using (28);
31.     Compute the fitness value for all locations;
32.     Choose the ideal location with the highest fitness value to be the initial location of the next iteration;
33.     Update the optimal location for every eel according to the fitness value;
34.   $\boldsymbol{end\ for}$
35.   //AC Stage
36.   Initialize the total number of attacking iterations $P_{AC}$, number of attack balls $AB$, where $AB = P_{AC}$, the ideal locations of groupers and eels from the ES stage, and shrinking ratio $\mu$;
37.   $\boldsymbol{for}(i = 1: P_{AC})$
38.     Compute the location of search agent, which has the optimal fitness value to be the position of the predicted prey;
39.     Determine the distance between the predicted prey and each other agent;
40.     Determine the distance between the prey and the farthest agent and assign the radius $R$ of the circle, where the prey is positioned in the circle centroid;
41.     Arbitrarily circulate the agents within the circle and compute the fitness value for all search agents;
42.     Choose the search agent with the best fitness value to be the prey in the next iteration;
43.     Calculate the radius of the next circles using (30);
44.     Form the circle and locate the prey in the circle centroid;
45.     Arbitrarily circulate the search agents within it;
46.     Determine the fitness value for the search agents;
47.     Choose the search agents with the optimal fitness value to be the prey (i.e., optimal assignment of tasks to resource);
48.   $\boldsymbol{end\ for}$
49. $\boldsymbol{end\ for}$
50. **Return** the best assignment of tasks to resources;
51. **End**
```

IV. Simulation Results

This section presents the experimental details and effectiveness of the suggested DNN-GME algorithm. The iFogSim simulator is used to implement the suggested approach. The traditional Java-based CloudSim simulator's features are expanded by the iFogSim simulator to enable the modeling of fog resources. The iFogSim Simulator has been widely used to simulate fog and cloud resources, applications, and a defined service deployment strategy that determines how the services are handled on the devices. A simulation-based performance assessment is carried out using the iFogSim simulator to compare the suggested algorithm with other algorithms, such as MOMFO [19], ERAM-EE [17], ANFIS-CHBA-OBL [22], and DELTa [24], to assess the performance of the suggested scheduling framework. The uniform deployment of IoT devices at the edge of the fog-cloud network is the basis for the uniform generation of user tasks in the simulation scenarios of the suggested and current algorithms. As shown in Table 2, processing speed, memory, and bandwidth are responsible for the processing resources in both the fog and cloud levels. By using the task requirements and resource capabilities, the classifier determines whether each job should be completed in the fog or on the cloud. The GME optimization technique generates optimal resource allocation while minimizing latency, execution time, and task failure rate.

Table II. Resource and Task Attributes

Resource type	***No. of nodes***	***CPU (MIPS)***	***Memory (GB)***	***Bandwidth (Mbps)***	***Latency (ms)***
Fog	50	500-2000	1-8	1-10	1-10
Cloud	10	5000-10000	16-32	50-100	20-50
Task type	***No. of tasks***	***CPU requirements (MIPS)***	***Memory requirements (MB)***	***Bandwidth requirements (Mbps)***	***Latency constraints (ms)***
Fog	1500	100-1000	300-500	2-8	2-8
Cloud	1500	2000-5000	500-1000	20-70	20-40

Table 3 shows the parameters for the DNN-GME method for resource allocation. At various periods, IoT devices create tasks that must be identified and allocated resources for execution. At each time interval, the number of tasks created in each category is raised by 500, while the system's performance is monitored. The devices create a total of 3000 tasks, which are then sent to the task categorizer in batches through the arrival queue.

Table III. Parameter Configurations for DNN-GME Algorithm

Algorithm	***Parameters***	***Values***
DNN	No. of input layer neurons	
	No. of hidden layers	2
	No. of neurons in hidden layer 1	32
	No. of neurons in hidden layer 2	64
	No. of output layer neurons	2
	Activation function	ReLU
	Learning cycle	500 epochs
	Loss function	Mean square error

	Optimizer (i.e., learning scheme)	Adam
	Batch size	100
GME Optimization	No. of agents	100
	Total no. of iterations	200
	No. of hops in the grouper's motion	3
	No. of hops in the eel's motion	3
	Constant k	Each agent has separate value
	Factor ξ	[0,1]
	Arbitrary value α	0.3
	Shrinking ratio μ	0.2

A. *Response Time*

It represents the time required for each task to be completed within the fog-cloud network. It is determined by

$$Response\ time = T + P \tag{31}$$

In (31), the amount of time required for data to be transmitted from edge devices to fog nodes is known as the transmission time (T). The time it takes for the fog nodes to process the data they have received is known as the processing time (P).

Fig. 6 depicts the performance evaluation of different scheduling algorithms in terms of response times for different numbers of tasks. The suggested DNN-GME has a lower response time, indicating its usefulness in dealing with varied workloads with minimal delay. For 500 tasks, it decreases response time by 66%, 57.5%, 43.33%, and 29.17% compared to the ERAM-EE, DELTa, MOMFO, and ANFIS-CHBA-OBL. Furthermore, when the task size is extended to 3000, the response time is lowered by 24.7%, 20.89%, 13.79%, and 8.76% compared to the same algorithms, respectively. As a result, the DNN-GME algorithm demonstrates its efficacy in delay-sensitive circumstances by regulating RA and task scheduling simultaneously as the number of tasks rises.

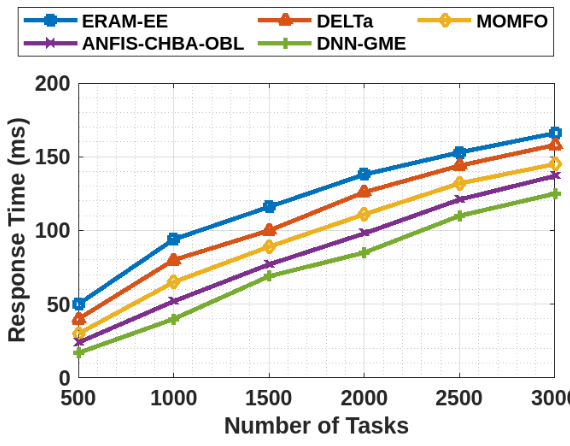

Figure 6. Response Time vs. No. of Tasks

B. *Latency*

It is the sum of processing and transmission latencies observed in the IoT-fog-cloud network. It is determined by (12). The delay for both suggested and current algorithms is shown in Fig. 7. Although the DNN-GME progressively increases the latency as the task load grows, it consistently has the lowest latency compared to other methods, indicating its adaptability and dependability to carry out time-sensitive tasks in fog-cloud computing environments

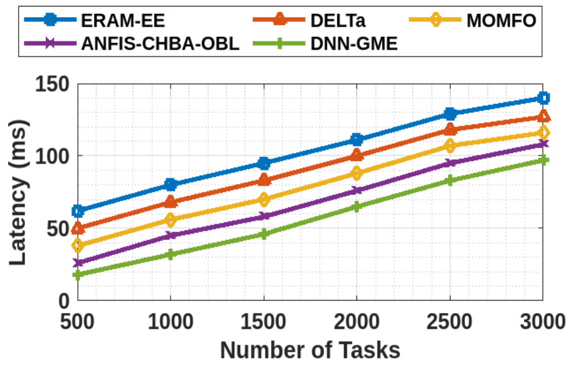

Figure 7. Latency vs. No. of Tasks

In comparison to the ERAM-EE, DELTa, MOMFO, and ANFIS-CHBA-OBL, it lowers latency by 70.97%, 64%, 52.63%, and 30.779% for 500 tasks, respectively. Also, compared to the identical algorithms, the latency is decreased by 30.71%, 23.62%, 16.38%, and 10.19% when the task load is increased to 3000. For applications requiring low latency in dispersed fog-cloud environments, the suggested DNN-GME may therefore be appropriate.

Fig. 8 shows the delay of 3000 tasks submitted to the system for execution. The number of tasks increased, as did their delay. It can be seen that the cloud only had the greatest rise in latency. This is the effect of increased network traffic between the channels. The fog-only scenario had the lowest latency when compared to the other deployment scenarios, including cloud-only and fog+cloud. However, as compared to other current scheduling methods, the suggested solution significantly reduces the system's total latency.

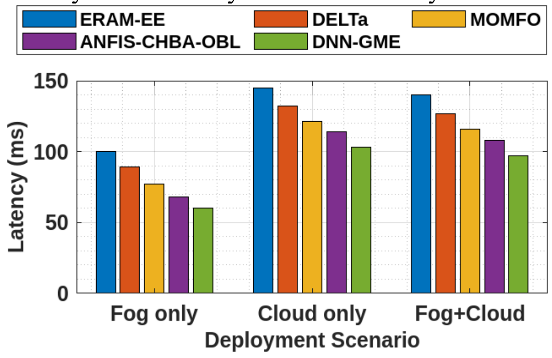

Figure 8. Latency for 3000 Tasks in Different Deployment Scenarios

C. *Load Imbalance*

It assesses the unequal allocation of tasks across fog and cloud resources.

Fig. 9 displays the load imbalance ratios between the proposed and current algorithms. The suggested DNN-GME maintains minimum load imbalances, which reach 2.5% at peak workload circumstances. The findings show that DNN-GME successfully distributes tasks among resources for all task counts. Under 500 tasks, the DNN-GME reduces load imbalance by 55.56%, 46.67%, 38.46%, and 20%, compared to the ERAM-EE, DELTa, MOMFO, and ANFIS-CHBA-OBL, respectively. Furthermore, increasing the task to 3000 reduces it by up to 28.57%, 21.88%, 16.67%, and 10.71% when compared to the same algorithms. So, the suggested DNN-GME is the ideal option for cases requiring fair resource allocation to avoid bottlenecks.

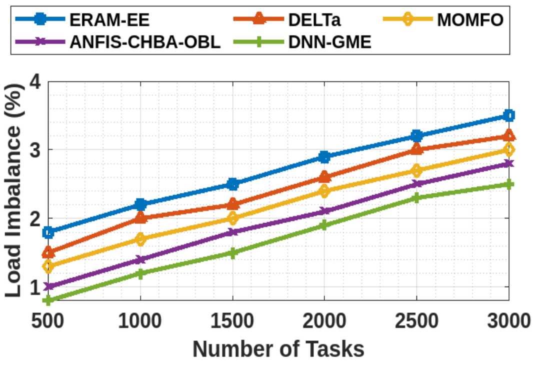

Figure 9. Load Imbalance vs. No. of Tasks

4.4 Task Failure Rate

It is the percentage of tasks that fail to perform efficiently due to resource and computational constraints. Fig. 10 shows how proposed and current algorithms handle failure rates when processing tasks ranging from 500 to 3000. The suggested DNN-GME outperforms all other algorithms tested in terms of task processing reliability. When operating with 2000 tasks, the DNN-GME decreases failure rates by 56%, 47.62%, 35.29%, and 21.43% compared to ERAM-EE, DELTa, MOMFO, and ANFIS-CHBA-OBL. For 3000 tasks, it is lowered up to 31.91%, 27.27%, 20%, and 11.11% when compared to the identical algorithms. The gains in these data demonstrate the usefulness of DNN-GME, which reduces reaction time and latency while balancing resource consumption and attaining high job completion rates in IoT-fog-cloud environments.

Task failures reported by the various algorithms for 3000 tasks in various conditions, including fog-only, cloud-only, and fog+cloud, are displayed in Fig. 11. Task failures recorded by all techniques have grown in tandem with the quantity of tasks.

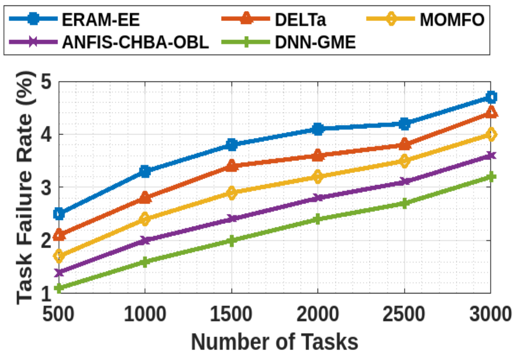

Figure 10. Task Failure Rate vs. No. of Tasks

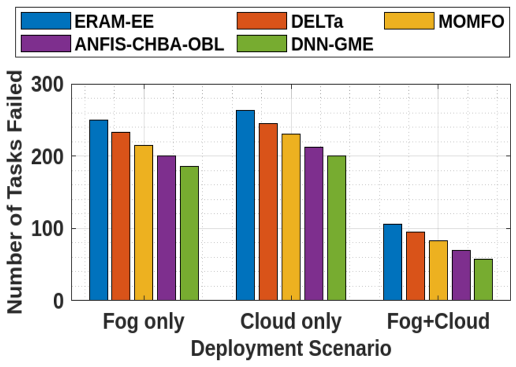

Figure 11. Number of Tasks Failed for 3000 Tasks in Different Deployment Scenarios

This demonstrates how increasing the number of tasks raises resource consumption and, consequently, task failure. It is evident that, across all deployment situations, the suggested algorithm saw the least increase in task failure compared to alternative algorithms. In particular, it performs better in fog+cloud situations than in fog-only and cloud-only situations. For example, compared to the fog-only and cloud-only cases, it lowers the number of tasks that fail by 68.82% and 71%, respectively.

4.5 Execution Time

It is calculated by (8). Fig. 12 depicts the execution time for 3000 tasks under various scheduling techniques and deployment situations. Both the proposed and existing methods showed considerable increases in execution timeThe rise is due to the increased demand for resources caused by the tasks. Compared to all cases, the fog-only exhibits the greatest increase in execution time, indicating that the increased number of tasks has overburdened the fog layer. According to the results, the suggested algorithm has the shortest execution

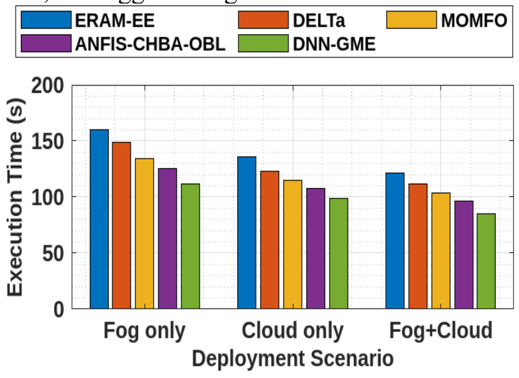

Figure 12. Execution Time for 3000 Tasks in Different Deployment Scenarios

time for the fog+cloud scenario compared to the other deployment situations and existing algorithms

V. Conclusion

This study presents an effective RA framework to employ cloud and fog resources to execute delay-sensitive operations and handle the massive amount of data created by end users. At first, using the GME optimization method, the tasks in the arrival queue were categorized according to the task guarantee ratio on the cloud and fog layers, and appropriate resources were assigned in the layers of their corresponding classes. Then, to categorize recently arriving tasks and match them with appropriate resources for execution in the layer of their respective classes, the DNN classifier was applied to prior allocation history data. Using classifiers to categorize the following tasks can help the system allocate resources more accurately when executing a massive quantity of tasks. Additionally, by using historical data to anticipate the layer and resource of newly incoming tasks, the system overhead caused by the optimal resource search time may be decreased. Finally, the simulation findings prove that the proposed DNN-GME algorithm outperformed earlier scheduling algorithms, such that the DNN-GME has 125 ms response time, 97 ms latency, 2.5% load imbalance, 3.2% task failure rate, and 85 s execution time for 3000 tasks. It is clear from the performance shown by the suggested algorithm that not all tasks can be completed with the fog alone without the cloud, and vice versa.

VI. References

[1] A. Choudhary, "Internet of things: a comprehensive overview, architectures, applications, simulation tools, challenges and future directions," Disc. Int of Things, vol. 4, no. 1, p. 31, 2024.

[2] D. Loconte, S. Ieva, A. Pinto, G. Loseto, F. Scioscia, and M. Ruta, "Expanding the cloud-to-edge continuum to the IoT in serverless federated learning," Future Gener. Comput. Syst., vol. 155, pp. 447–462, 2024.

[3] J. S. Yalli, M. H. Hasan, and A. Badawi, "Internet of things (IoT): origin, embedded technologies, smart applications and its growth in the last decade," IEEE Access, vol. 12, pp. 91357–91382, 2024.

[4] S. Z. Marshoodulla and G. Saha, "A survey of data mining methodologies in the environment of IoT and it's variants," J. Netw. Comput. Appl., p. 103907, 2024.

[5] H. Kuchuk and E. Malokhvii, "Integration of IoT with cloud, fog, and edge computing: a review," Adv. Inf. Syst., vol. 8, no. 2, pp. 65–78, 2024.

[6] A. K. Y. Yanamala, "Emerging challenges in cloud computing security: a comprehensive review," Int. J. Adv. Eng. Technol. Innov., vol. 1, no. 4, pp. 448–479, 2024.

[7] N. K. Rajpoot, P. D. Singh, B. Pant, and V. Tripathi, "The future of cloud computing: a paradigm shift with fog computing," in Integration of Cloud Computing and IoT, pp. 309–323.

[8] S. N. Srirama, "A decade of research in fog computing: relevance, challenges, and future directions," Softw. Pract. Exp., vol. 54, no. 1, pp. 3–23, 2024.

[9] T. Shwe and M. Aritsugi, "Optimizing data processing: a comparative study of big data platforms in edge, fog, and cloud layers," Appl. Sci., vol. 14, no. 1, p. 452, 2024.

[10] D. Alsadie, "Advancements in heuristic task scheduling for IoT applications in fog-cloud computing: challenges and prospects," PeerJ Comput. Sci., vol. 10, p. e2128, 2024.

[11] M. R. Rezaee, N. A. W. A. Hamid, M. Hussin, and Z. A. Zukarnain, "Fog offloading and task management in IoT-fog-cloud environment: review of algorithms, networks and SDN application," IEEE Access, vol. 12, pp. 39058–39080, 2024.

[12] E. Khezri, R. O. Yahya, H. Hassanzadeh, M. Mohaidat, S. Ahmadi, and M. Trik, "DLJSF: data-locality aware job scheduling IoT tasks in fog-cloud computing environments," Results Eng., vol. 21, p. 101780, 2024.

[13] A. Mahapatra, K. Mishra, R. Pradhan, and S. K. Majhi, "Next generation task offloading techniques in evolving computing paradigms: comparative analysis, current challenges, and future research perspectives," Arch. Comput. Methods Eng., vol. 31, no. 3, pp. 1405–1474, 2024.

[14] W. C. Chuan, S. Manickam, E. Ashraf, and S. Karuppayah, "Challenges and opportunities in fog computing scheduling: a literature review," IEEE Access, vol. 13, pp. 14702–14726, 2025.

[15] H. K. Apat, V. Goswami, B. Sahoo, R. K. Barik, and M. J. Saikia, "Fog service placement optimization: a survey of state-of-the-art strategies and techniques," Computers, vol. 14, no. 3, p. 99, 2025.

[16] I. Z. Yakubu and M. Murali, "An efficient meta-heuristic resource allocation with load balancing in IoT-fog-cloud computing environment," J. Ambient Intell. Humaniz. Comput., vol. 14, no. 3, pp. 2981–2992, 2023.

[17] P. Periasamy et al., "ERAM-EE: efficient resource allocation and management strategies with energy efficiency under fog–internet of things environments," Connection Sci., vol. 36, no. 1, p. 2350755, 2024.

[18] S. R. Hassan, A. U. Rehman, N. Alsharabi, S. Arain, A. Quddus, and H. Hamam, "Design of load-aware resource allocation for heterogeneous fog computing systems," PeerJ Comput. Sci., vol. 10, p. e1986, 2024.

[19] T. Salehnia, et al., "An optimal task scheduling method in IoT-fog-cloud network using multi-objective moth-flame algorithm," Multimed. Tools Appl., vol. 83, no. 12, pp. 34351–34372, 2024.

[20] M. B. Shaik, K. S. Reddy, K. Chokkanathan, S. A. A. Biabani, P. Shanmugaraja, and D. D. Brabin, "A hybrid particle swarm optimization and simulated annealing with load balancing mechanism for resource allocation in fog-cloud environments," IEEE Access, vol. 12, pp. 172439–172450, 2024.

[21] A. Mahapatra, S. K. Majhi, K. Mishra, R. Pradhan, D. C. Rao, and S. K. Panda, "An energy-aware task offloading and load balancing for latency-sensitive IoT applications in the fog-cloud continuum," IEEE Access, vol. 12, pp. 14334–14349, 2024.

[22] S. K. Srichandan, S. K. Majhi, S. Jena, K. Mishra, and R. Bhat, "A secure and distributed placement for quality of service-aware IoT requests in fog-cloud of things: a novel joint algorithmic approach," IEEE Access, vol. 12, pp. 56730–56748, 2024.

[23] M. A. Ala'anzy, R. Zhanuzak, R. Akhmedov, N. Mohamed, and J. Al-Jaroodi, "Dynamic load balancing for enhanced network performance in IoT-enabled smart healthcare with fog computing," IEEE Access, vol. 12, pp. 188957–188975, 2024.

[24] A. Mahapatra, R. Pradhan, S. K. Majhi, and K. Mishra, "DELTa: Dynamic Energy-and-Latency-aware Task scheduling for Fog-Cloud Paradigm," IEEE Access, vol. 13, pp. 74617–74633, 2025.

[25] V. C. Bharathi, S. S. Abuthahir, M. Ayyavaraiah, G. Arunkumar, U. Abdurrahman, and S. A. A. Biabani, "O2O-PLB: a one-to-one-based optimizer with priority and load balancing mechanism for resource allocation in fog-cloud environments," IEEE Access, vol. 13, pp. 22146–22155, 2025.

[26] N. A. Mansour, M. S. Saraya, and A. I. Saleh, "Groupers and moray eels (GME) optimization: a nature-inspired metaheuristic algorithm for solving complex engineering problems," Neural Comput. Appl., vol. 37, no. 1, pp. 63–90, 2025.