IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 4 February 2024, accepted 9 March 2024, date of publication 13 March 2024, date of current version 20 March 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3376670

RESEARCH ARTICLE

A Novel Offloading Mechanism Leveraging Fuzzy Logic and Deep Reinforcement Learning to Improve IoT Application Performance in a Three-Layer Architecture Within the Fog-Cloud Environment

DEZHEEN H. ABDULAZEEZ[1] **AND SHAVAN K. ASKAR**[2]
[1]Department of Computer Science, College of Science, Duhok University, Duhok 42001, Iraq
[2]Department of Information System Engineering, Erbil Technical Engineering College, Erbil Polytechnic University, Erbil 44001, Iraq
Corresponding author: Dezheen H. Abdulazeez (dezheen.abdulazeez@uod.ac)

ABSTRACT This paper presents a novel offloading technique designed to enhance the efficiency of Internet of Things (IoT) applications within a sophisticated three-layer architecture situated in a fog computing environment. The IoT layer contains various intelligent IoT devices that generate a large number of tasks, each characterized by distinct specifications such as size, computational demand, communication requirements, and latency constraints. owing to the limited storage and computing capacity of resource-constrained IoT devices, it is essential to offload these tasks to different layers to ensure effective processing while satisfying the required Quality of Service (QoS) goals. To address this challenge, a fuzzy logic-based task scheduler is employed to make informed offloading decisions, considering task attributes and determining the most suitable processing layers—whether locally at the IoT layer, on collaborative fog nodes, or in the cloud. Furthermore, the study leverages the Deep Q Network (DQN) method, a form of deep reinforcement learning, to identify the optimal fog node for offloading tasks and to maintain a balanced workload distribution across collaborative fog nodes. The experimental findings demonstrate that the proposed scheme outperforms state-of-the-art solutions in terms of latency, power consumption, network usage, throughput, and offloading rate in comparison with the Non-offload, First-Fit, GASDEO, and NAFITO-FLA methods.

INDEX TERMS Offloading, fuzzy logic, deep reinforcement learning, DQN, IoT applications, fog computing.

I. INTRODUCTION

The development of Internet of Things (IoT) technology has resulted in the emergence of various time-sensitive IoT applications, such as autonomous vehicles [1], augmented reality, and smart healthcare [2], [3]. These applications require significant computational resources with low latency for real-time processing, resulting in high energy consumption for resource-limited IoT devices. The fog computing paradigm has emerged as a viable alternative to overcome the resource constraints and intensive computing demands of IoT devices [4], [5].

Fog computing brings cloud-like services to the network edge to enhance the performance of IoT applications in terms of resource utilization, energy usage, service delay, and workload balancing [6]. With fog computing, time-sensitive data can be processed at the network edge or near the data source, minimizing the amount of data sent to the cloud [7],

The associate editor coordinating the review of this manuscript and approving it for publication was Daniel Augusto Ribeiro Chaves.

[8]. This not only optimizes the use of computational resources but also ensures that smart applications satisfy their time-sensitive requirements [9].

However, the implementation fog computing presents further challenges, and user task offloading is considered a major hurdle. The research in [10] and [11] comprehensively covers all issues related to offloading in fog computing, including decisions regarding whether tasks should be executed locally or transmitted to the cloud or fog. An additional challenge involves the distribution of the workload across various heterogeneous fog devices with different resources and computational capacities. The increasing number of requests exacerbates this challenge, potentially resulting in longer task queues for more powerful fog nodes. The extended waiting time within these queues may exceed the latency requirements of the time-sensitive applications.

In addition, many single-fog computing systems cannot effectively manage resource-intensive tasks owing to a lack of available resources, inadequate processing capacities, or the dynamic nature of task demands. Consequently, vertical offloading is employed to transfer the task to a remote cloud server when the processing demands of a task surpass the capabilities of fog platforms. Alternatively, horizontal offloading is another viable option, involving the allocation of user tasks to the most suitable surrogate fog nodes [12].

To address the aforementioned challenges, it is essential to have a task scheduler that identifies which tasks will be executed by the local, fog, or cloud layers to meet the targeted objectives of IoT applications [13], [14]. Moreover, effective resource allocation becomes vital in this situation, ensuring the distribution of user tasks among different fog nodes while satisfying diverse quality of service requirements [15]. Accordingly, this study introduces a novel task offloading strategy based on task scheduling and load balancing to enhance the performance of IoT applications within a three-layer architecture in a fog-cloud computing environment. Initially, IoT devices generate a large number of heterogeneous tasks. Due to the limited storage and computational capabilities of IoT devices, these tasks must be transferred into different layers for effective processing. To achieve this, a fuzzy logic strategy is implemented, which serves as a task scheduler to select the target processing layers (e.g., locally at the IoT layer, collaborative fog node, or cloud) while considering the different characteristics of each task.

In the process of task offloading, if a fog node in a fog layer is unable to handle a task because of processing requirements, the system checks whether the task can be sent to a different fog node within the same cluster. The task is redirected to the alternative fog node if the response is positive; otherwise, it is moved to the cloud layer. This mechanism ensures that when a task is assigned to one fog node in a cluster during task placement, other nodes within that cluster can also serve the task if necessary. This approach results in reduced energy consumption and latency because it provides quick access to the task within the same cluster without relying on cloud access. Furthermore, this leads to increased task allocation to the fog layer during implementation, thereby reducing the number of requests directed to the cloud layer and consequently lowering the energy consumption of the layer. To achieve this, the implementation involves employing a DQN algorithm, which identifies the optimal fog node for offloading user tasks and ensures a balanced load distribution among collaborative fog nodes within each cluster.

The proposed scheme considers several important aspects of the implementation of a novel offloading strategy. These aspects include: a) making optimal decisions on whether to offload tasks locally or remotely, b) selecting the most suitable fog nodes, c) inter-fog collaboration, d) heterogeneity of task and computational nodes, and e) the capable distribution load among computational nodes. In this regard, extensive simulation results are provided to demonstrate the efficiency of the proposed mechanism. The primary contributions of this study are as follows.

- Implement a novel offloading strategy to enhance the performance of latency-sensitive applications within the three-layer architecture of a fog-cloud environment.
- The proposed strategy considers several important aspects, such as making optimal decisions on whether to offload tasks locally or remotely, selecting the most suitable fog node, encouraging inter-fog collaboration, addressing the heterogeneity of task requirements and computational nodes, implementing task scheduling, and facilitating load-sharing.
- The fuzzy logic strategy was implemented as a task scheduler to determine the target processing layers for heterogeneous tasks generated from IoT devices, considering the different characteristics of each task. (e.g., local fog node, collaborative fog nodes, and cloud)
- Presented a Deep Q learning (DQN) method to determine the most suitable fog node for offloading user tasks and ensure balanced load distribution among collaborative fog nodes within each cluster.
- Finally, we assessed the effectiveness of the proposed offloading schema using an iFogSim simulator. The experimental results demonstrate that the proposed scheme outperforms the other approaches in terms of latency, power consumption, network usage throughput, and offloading rate.

The rest of the paper is structured as follows. Section II discusses the current state-of-the-art task offloading strategies for fog computing. Section III outlines the proposed novel offloading strategy, focusing on task scheduling and load balancing within a three-layered fog-cloud computing architecture. The algorithms employed in the proposed architecture are described in Section IV. Section V evaluates the performance of the proposed offloading schema by comparing it with benchmark schemes. Finally, Section VI concludes the study and outlines potential future research directions.

II. RELATED WORK

Task offloading is a well-established concept widely used in cloud computing. Offloading involves transferring tasks from resource-constrained IoT devices to another resource-rich device in order to improve the performance of time-critical IoT applications. Devices belonging to users are strategically positioned at the network's edge, facilitating the offloading of computationally intensive tasks to fog and cloud nodes through 4G/5G or Wireless Local Area Networks (WLAN) Networks. In instances where a single fog node is inadequate to handle ever-increasing workloads, additional fog or cloud nodes are available to assist in handling such tasks. This practical solution effectively supports IoT applications by sending computationally intensive tasks to resource-rich servers within the fog-cloud system. Researchers [10] systematically and comprehensively analyzed the utilization of RL or DRL algorithms to address offloading-related challenges in fog computing. This study elaborates on the offloading process from various prespectives, covering aspects such as offloading decisions, offloading metrics, offloading directions, and offloading modes.

This section presents an overview of the most relevant studies that are closely related to our research. For instance, the study [16] presented an offloading approach to identify the optimal location for executing modules. A MAPE control loop was implemented on the intelligent gateway to determine whether the task should be processed locally or transferred to the fog or cloud. The proposed method involved two stages: first, a greedy technique was employed to locally evaluate Fog Devices (FDs), considering sibling nodes and the parent, second, the optimal location for module execution was determined using a Deep Reinforcement Learning (DRL) algorithm. The evaluation of this method is based on the following criteria: time delay, execution cost, energy consumption, learning rate, network resource consumption, and time interval of the offloading operation. The results indicate that the proposed schema exhibits superior performance compared to other existing algorithms. However, the allocation of tasks to their respective destinations depends on the sequence of fog nodes, or could. Moreover, their offloading design lacks a predefined task scheduling mechanism to identify the tasks that should be performed at each layer. The authors also did not consider the heterogeneity of the workloads and fog devices.

Additionally, the study described in [17] involved a four-tier architecture intended for workload balancing and delay-aware scheduling within fog computing. The first tier, Tier-1, includes the IoT devices. The following tier categorizes workloads into Low Priority (LP) and High Priority (HP) through a router, utilizing the Dual Fuzzy Logic method. The Fuzzifier processes the following input metrics: minimum execution time, maximum completion time, task size, and arrival time. Tasks assigned high priority were directed to the fog tier. Within Tier 3, a new fog computing paradigm, specifically an artificial fractal, was proposed as an alternative to centralized or distributed fog environments. Each fog node integrates a communication component, load monitor, and fog scheduler. The communication component functions as a messenger that sends heartbeat messages to the seed node to address fog node failures. An Artificial Neural Network (ANN) was deployed to predict the current usage of fog nodes, which were regularly updated. When an IoT device suffers from a lack of available resources, the request is sent to the cloud tier. Nevertheless, their offloading architecture failed to select the most optimal fog node for task offloading within the fog tier.

A context-aware computation offloading decision maker was proposed by the authors of this research [18]. Their approach introduced a MAPE loop to solve the offloading decision-making process. The proposed architecture covers the monitoring, analysis, planning, and execution phases. During the monitoring phase, contexts were collected, and in the analysis phase, these contexts were examined. Subsequently, the planning phase formulates the offloading instructions, and finally, the execution phase implements these instructions. The proposed context-aware approach then determines the appropriate offloading decision. The evaluation of this approach incorporates metrics such as execution cost, network usage, power consumption, module size, mobile types, delay, and time interval of offloading operations. However, the authors did not observe heterogeneity in the task and fog devices. They also did not consider inter-fog collaboration.

Another study was conducted to tackle task prioritization and offloading policies using fuzzy logic [19]. The primary goal was to minimize the incurred delay and average waiting time. Additionally, the proposed approach considers the priority of each task and assign them to the respective queues for scheduling by compatible nodes. Concurrent execution of different task types across various target layers. Furthermore, a binary elitism-based multi-population Jaya algorithm was employed to schedule diverse tasks, with the aim of achieving optimal mapping. The proposed algorithm evaluates both task and machine heterogeneity to assess its effectiveness. The experimental findings demonstrates the superiority of the proposed method over existing methods. However, their offloading architecture proved inadequate in choosing the most appropriate fog node for task offloading, and it could not distribute the workload evenly among processing nodes.

The research outlined in [20] introduced an innovative method for task offloading, aiming to reduce the total delay for time-critical applications. This method incorporates a fuzzy logic technique, taking into account various application characteristics such as network demand, CPU demand, and delay sensitivity, in addition to considering resource utilization and heterogeneity. However, their offloading architecture proved inadequate in choosing the most appropriate fog node for task offloading, and could not distribute the workload evenly among the processing nodes.

At study conducted by researchers [21] addressed the challenge of task offloading within a software-defined access network. IoT devices communicate with fog computing

TABLE 1. Summary of current works on offloading strategies.

Existing works	Optimal offloading decision on layer	Optimal fog node selection	Heterogeneity of tasks	Heterogeneity of computational nodes	inter-fog collaboration	Load balancing	Task Scheduling
[16]	✓	✓	×	×	×	×	×
[17]	✓	x	×	×	✓	✓	✓
[18]	✓	✓	×	×	×	×	×
[19]	✓	×	✓	✓	✓	×	✓
[20]	✓	×	✓	✓	×	×	✓
[21]	×	✓	×	×	✓	×	×
[22]	×	✓	×	×	✓	×	×
[23]	✓	✓	×	×	✓	✓	×
[24]	✓	×	✓	✓	×	×	✓
[26]	×	✓	×	×	✓	✓	×
[27]	×	✓	×	×	×	×	✓
[28]	×	✓	×	×	×	×	✓
[29]	×	×	×	×	×	✓	✓
[25]	×	×	×	✓	×	✓	×
Proposed schema	✓	✓	✓	✓	✓	✓	✓

nodes in a network via multi-hop IoT access points (APs). The key considerations of the proposed methodology include: a) best decision-making for local or remote task processing, b) selecting the most appropriate fog node, and c) determining the ideal path for offloading. The experimental results demonstrated a substantial reduction in the average delay and energy consumption compared to existing methods. Nevertheless, the offloading architecture requires a proficient pre-planned task scheduling mechanism to designate the appropriate layer for processing each task. Additionally, it is incapable of distributing the workload among processing nodes.

Another research outlined in [22] introduced an effective decision-making strategy, endowing fog nodes with intelligence to determine a suitable method for data processing. Devote, developed through reinforcement learning algorithms, demonstrated adaptability to a dynamic IoT environment. The selection of an algorithm relies on the nature of the data, and is categorized as too critical, critical, or normal. In addition, to select a suitable fog node to offload critical data, an online secretary-based algorithm was proposed. This algorithm effectively manages the trade-off between processing delays and efficient data service. Numerical analysis showed that Devote incurred minimal service delays while achieving heightened user satisfaction. However, this study did not address the aspects of device heterogeneity and mobility.

Furthermore, the authors [23] introduced a novel strategy for task offloading, aiming to make optimal decisions regarding when and where to offload a task, whether to a fog node or the cloud server. The problem was formulated as a Markov decision process (MDP). The introduced MDP involves two decision-makers: IoT users, responsible for selecting the fog node to transfer their tasks, and fog nodes, which may select to transfer specific tasks to other alternative fog devices or to the remote cloud to distribute the workload fairly across fog nodes. To overcome the challenges associated with large state and action spaces, a Q-learning approach was developed to derive an ideal policy. Simulation results demonstrate that the proposed schema reduces the delay time and improves load balancing compared to alternative methods. However, their offloading schema did not have an efficient predetermined task scheduling to choose the appropriate layer for processing each task.

Many task offloading strategies have been implemented in the literature that consider different fog architectures and evaluate different performance metrics [16], [17], [18], [19], [20], [21], [22], [23], [24], [25]. However, a significant portion of these methods designed for dedicated computing situations in fog environments and some important aspects have not been considered in their offloading strategies: a) making optimal decisions on whether to perform local or remote task computation; b) selecting the most appropriate fog node; c) inter-fog collaboration; e) heterogeneity of task and computational nodes; f) task scheduling; and g) load-sharing. Addressing these aspects contributes to enhanced performance metrics in fog-cloud computing. Addressing these aspects contributes to enhanced performance metrics in fog-cloud computing. Therefore, we propose a novel offloading scheme for three-layered fog computing to optimize the performance of IoT applications while considering the aforementioned aspects. A summary of the current work on offloading techniques is presented in TABLE 1.

III. PROPOSED SYSTEM ARCHITECTURE

As mentioned earlier, the main challenge in fog computing is to determine whether tasks should be performed locally or transferred to fog or cloud. Another challenge involves effectively distributing tasks among available resources to reduce latency and power consumption. Despite the preceding section covering different solutions to the highlighted challenges, there is still room for future improvements in lowering delays and energy usage in fog-cloud systems.

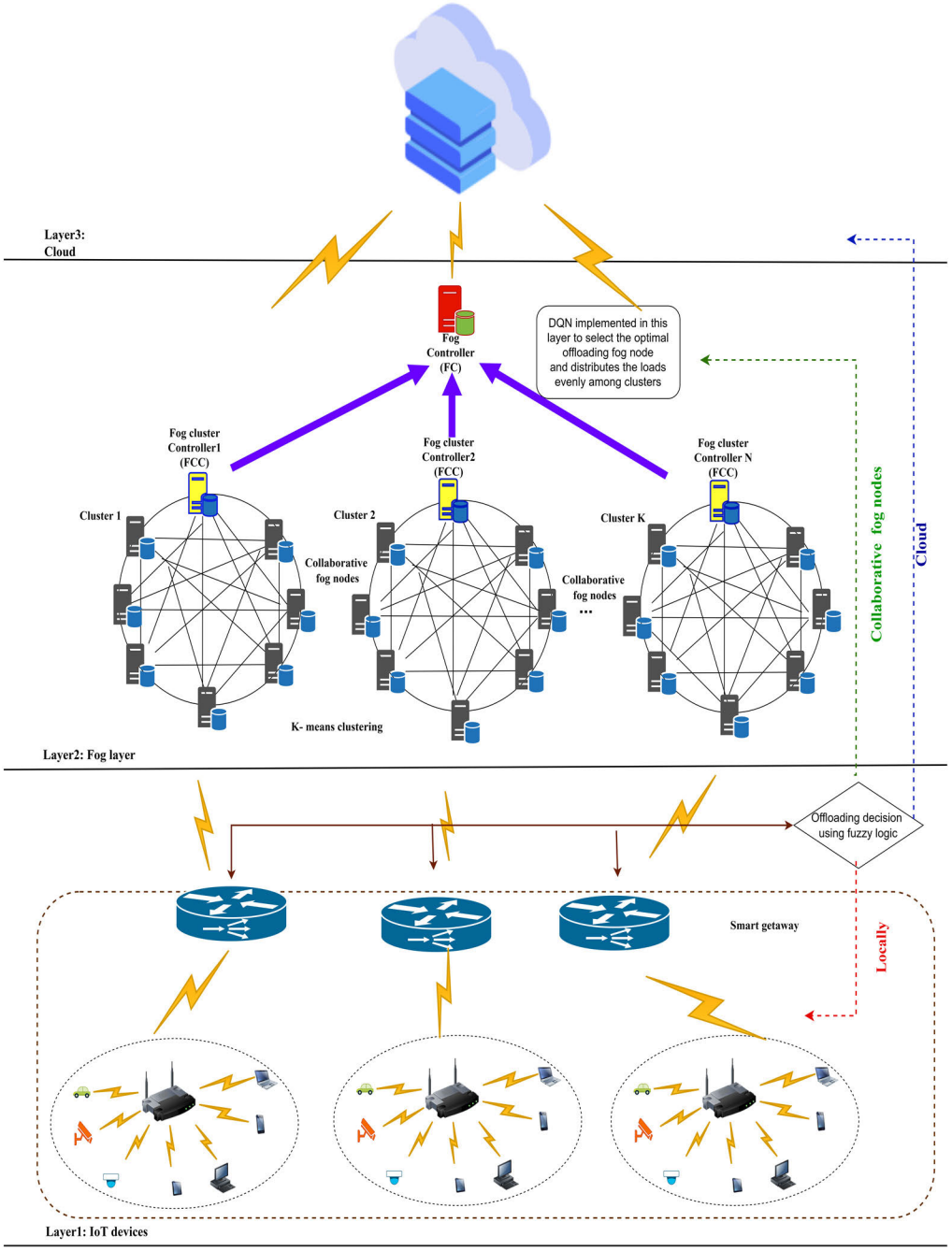

FIGURE 1. Proposed system model.

This section introduces our innovative offloading strategy, engaging task scheduling, and workload balancing aimed at enhancing the performance of IoT applications within a three-layer architecture in a fog environment, as illustrated in FIGURE 1.

The hierarchical architecture of the proposed approach consists of three layers: the IoT layer (Layer 1), fog layer (Layer 2), and cloud layer (Layer 3). This architecture includes IoT devices in the IoT layer and intermediate nodes responsible for task routing to the appropriate fog nodes or cloud. The Fog layer contained a collection of geographically distributed fog nodes. In addition, a cloud datacenter was deployed in the third layer of the hierarchical architecture. The operational characteristics of each layer are as follows:

Initially, the IoT layer contained many heterogeneous intelligent devices that produced a substantial volume of data through actuators (e.g., Motors and valves), sensors (e.g., temperature sensors, humidity sensors, and motion sensors), and Intelligent IoT Devices(e.g., Smart cameras, wearable devices, and environmental monitoring devices). Additionally, this layer incorporates intermediate nodes such as switches, and routers, which are referred to as intelligent

gateways. Because of the restricted processing and storage capacities of IoT devices, these intelligent gateways transfer these tasks from IoT devices to either the fog or the cloud layer. Moreover, this layer employs a fuzzy logic technique to evaluate and categorize user tasks according to their specifications in order to identify suitable layers for offloading (e.g., locally at the IoT layer, collaborative fog node, or cloud).

Next, the Fog Layer contains many geographically distributed computing nodes, also known as fog nodes. In this phase, the K-means strategy is used to cluster all fog nodes based on their geographical distance. Each cluster contains a combination of heterogeneous and homogeneous resources to achieve the QoS goals. Every fog node in a cluster communicates with a fog cluster controller (FCC). The fog controller (FC) is responsible for linking all FCCs of the Fog layer. The FC includes a load balancer that efficiently allocates workload to each cluster. A DQN algorithm is implemented in this layer to find the optimal fog node to offload user tasks and create a balanced load among collaborative fog nodes in each cluster.

Finally, the cloud layer is responsible for executing intensive tasks that do not have stringent delay constraints, aiming to minimize the service time and fulfill general deadline requirements. This layer includes centralized, high-performance virtual machines (VMs), which serve as computing nodes

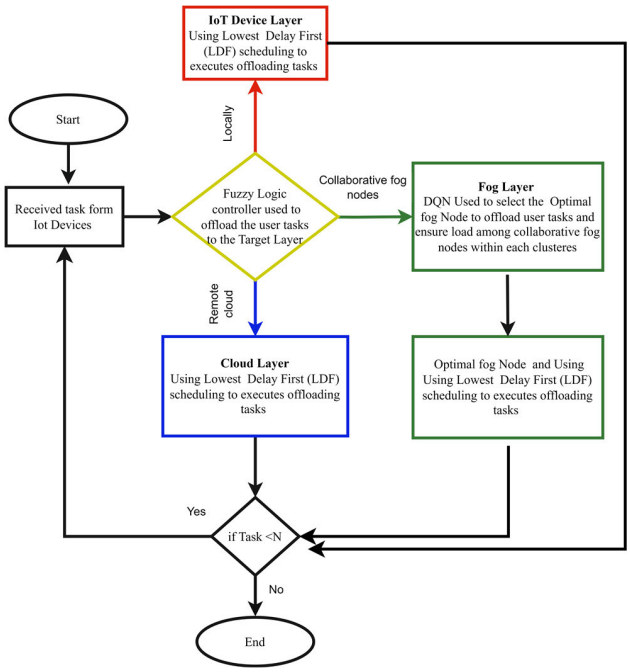

FIGURE 2. The flow chart of the proposed offloading approach.

The next section describes the details of the operational characteristics of each layer in the proposed architecture, along with the algorithms used for task scheduling and load balancing to address offloading issues. FIGURE 2 depicts a flowchart of all the stages of the proposed approach. The main research motivations and algorithms used in this study are outlined in *TABLE 2*. The symbols used in this study are listed in *TABLE 3*.

TABLE 2. Main research motivations.

Layers	Operational characteristics of each layer	Algorithm used
Layer-1 (IoT devices)	1. Data acquisition 2. Optimal decision on: A. Locally B. Collaborative fog node C. Cloud	Fuzzy Logic Algorithm
Layer-2 (Fog Layer)	1. Fog nodes clustering 2. Optimal fog node selection and load balancing	K-means++ clustering DQN Algorithm
Layer-3 (Cloud)	It is responsible for processing executed intensive tasks that do not have stringent delay constraints.	---

TABLE 3. Definitions of the symbols used in the paper.

Symbol	Definition
IoT	Internet of Thing
QoS	Quality of service
DQN	Deep Q learning or Deep Q network
DRL	Deep reinforcement learning
FDs	Fog Devices
Π	Learned policy
LP	Low priority
HP	High priority
ANN	Artificial neural network
AP	Access points
FCC	Fog cluster controller
FC	Fog controller
VMs	Virtual machines
FLA	Fuzzy logic architecture
MSE	Mean squared error
NAFITO-FLA	Greedy auto-scaling deep reinforcement learning-based offloadin
GASDEO	A novel approach for IoT tasks offloading basded fuzzy logic algorithem
Qd	Queuing delay
Td	Transmission delay
$Prod$	Propagation delay
Pd	Processing delay
EC	Energy consumption
Tn	Current time
Tlu	The last utilization update time
Ph	The host power at the last utilization update
Bps	Bits per second
$Nspt$	The total number of tasks that have been successfully transmitted
T i	The time taken for the transfer tasks
Tf	Total number of tasks that have been successfully offloaded or transferred
N	The total number of tasks including both local and offloaded tasks

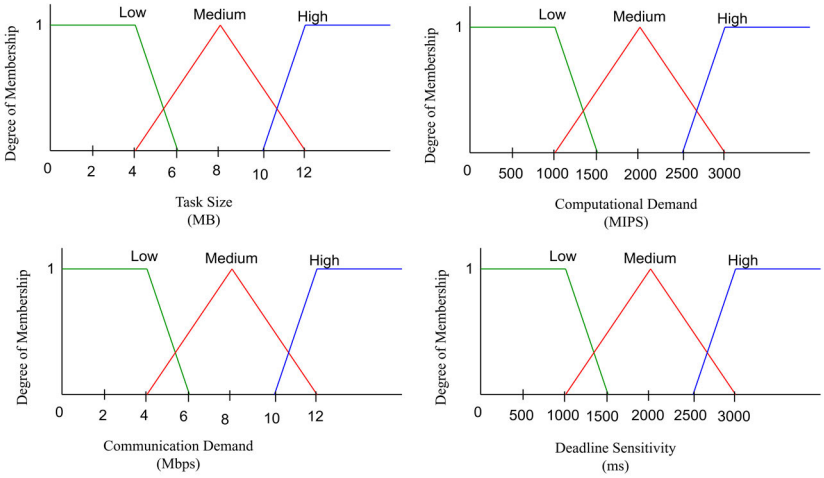

FIGURE 3. Offloading decisions using fuzzy logic architecture.

IV. ALGORITHMS USED IN THE PROPOSED ARCHITECTURE

This section provides a detailed explanation of the operational characteristics of each layer, building on the overview previously presented. In addition, it outlines the algorithms implemented in each layer for task scheduling and load balancing to address offloading issues.

A. IoT LAYER: DATA ACQUISITION

The IoT layer generates a substantial volume of data originating from heterogeneous intelligent IoT devices. The data are received with varying specifications, including size, computational demand, communication demand, and delay constraints.

The pseudocode for application preparation is presented as follows: To execute the proposed method, the following elements are created: IoT Devices, Smart gateways, heterogeneous fog devices, remote cloud data centers, IoT applications, and network topology, as illustrated in Algorithm 1. Subsequently, the simulator was initiated. Each task generates an IoT application in each time slot using different parameters (size, computational demand, communication demand, and tolerance delay constraints). In the final stage of this algorithm, the optimal computational resources are selected to offload user tasks, as introduced in Algorithm 2.

Additionally, the proposed approach gathers information about offloading tasks to determine the appropriate location for transfering the task (e.g., locally at the IoT layer, collaborative fog nodes, or the cloud), as depicted in Figure 3. The process for scheduling and allocating tasks is as follows: During this phase, the fuzzy logic controller method was employed to evaluate and categorize the tasks based on their specifications, with the aim of selecting suitable processing layers. This classification categorizes user tasks into three classes: the local class (class 1), collaborative fog nodes class (class 2), and cloud class (class 3).

Algorithm 1 Application Preparation

1: *Create a Set of IoT devices*
2: *Create smart gateways*
3: *Create the heterogeneous fog node with different capacity (storage, CPU, Processing)*
4: *Create remote Cloud data center*
5: *Create an IoT application that contains module and task*
6: *Create Network Topology*
7: *For each task that generates in IoT application-specific time slot do*
8: *Create a Task with parameters (Task Size, computational demand, communication demand, and tolerate delay constraint) and utilize fuzzy logic technique to determine the target computational resource for offloading By Algorithm 2*
9: *end for*

The local class contains high-priority tasks that have strict delays and do not demand many computational and communication resources to be serviced. Therefore, it will be placed at locally at the IoT layer (labeled as $Q1$). The collaborative fog node class involves tasks characterized by moderate delay, requiring intermediate levels of computational and communication resources for efficient service. Thus, this class involves a combination of heterogeneous and homogeneous nodes to be processed. Consequently, collaborative fog nodes (labeled as $Q2$) are appropriate for executing such tasks. Lastly, the cloud class covers all tasks that necessitate extensive computational and communication resources with a tolerated delay for processing. Consequently, tasks belonging to this category are handled by the cloud (labeled as $Q3$). In this context, the fuzzy logic procedure is utilized to categorize user requests, assigning each user request to the respective queues (Q1, Q2, and Q3).

The Fuzzy Logic Architecture (FLA) consists of three main parts: (1) fuzzy inputs, (2) fuzzification, and (3)

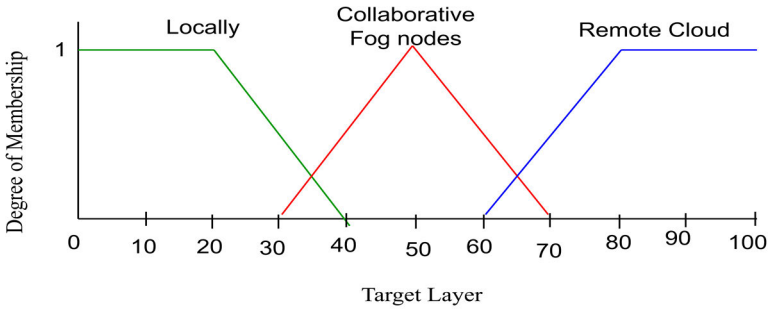

FIGURE 4. The four membership functions.

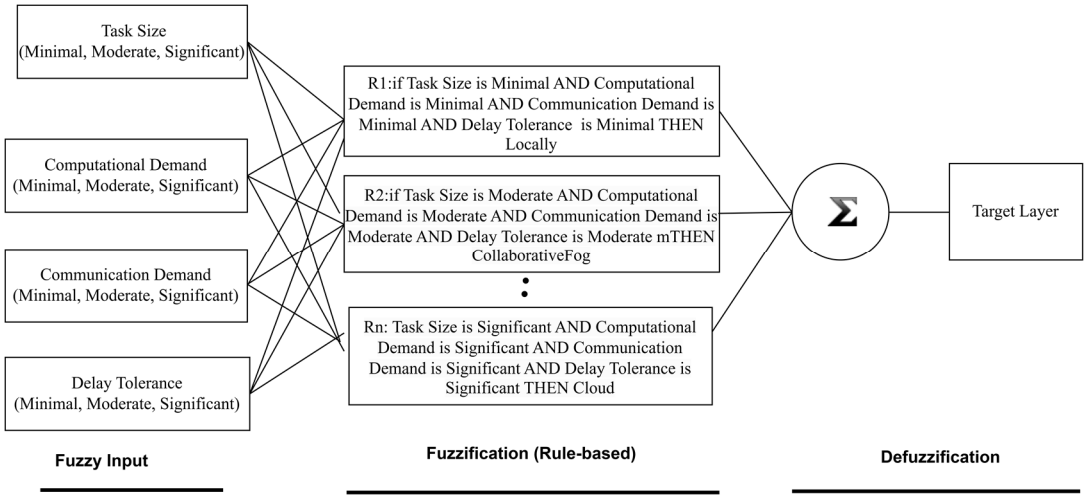

FIGURE 5. The output membership function of the fuzzy logic system.

defuzzification, as shown in Figure 3. Fuzzy inputs served as the primary parameters for the fuzzification process. Four input parameters are considered based on the requirements of the tasks: (1) Task Size, (2) Computational Intensity, (3) Communication Intensity, and (4) delay tolerance of the task. These parameters are considered lexical variables that are categorized as significant, moderate, and minimal. During the Fuzzification process, the fuzzifier consideres all input parameters and evaluates them using the Fuzzy Membership Functions specified in the Fuzzy Knowledge Base (FKB). The linguistic values for each fuzzy input are identified using membership functions. As shown in Figure 4, four membership functions were created based on three specifications (significant, moderate, and minimal) and four task requirements.

In addition, the input parameters werw processed via an Inference Engine. This engine creates fuzzy rules that comprise a sequence of if-else conditions containing all potential system probabilities and application requirements [30]. For example, a fuzzy rule in inferences can be stated as follows. If the size of the task, computational demand, communication demand, and deadline sensitivity are all minimal, the task should be processed locally. The total number of fuzzy rules is $n=3^4 = 81$ based on four membership functions, each consisting of three linguistic terms. Some examples of fuzzy rules are given in Table 4. Subsequently, Defuzzification was executed to convert the fuzzy rules into appropriate values according to the membership functions, as depicted in Figure 5. Further explanations of how fuzzy logic operates in decision-making can be found in [24] and [31].

Algorithm 2 delineates the steps for identifying the most efficient layer for offloading the user tasks. Initially, data generated tasks from the IoT application are collected, including the size of the task, computational task demand, communication demand, and deadline requirements. The algorithm employs a fuzzy logic function to convert each fuzzy variable into a quantifiable linguistic term. Subsequently, a fuzzy rules-based approach is utilized to identify the optimal target layer for each computation task, whether it is locally at the IoT layer, collaborative fog node, or cloud server.

B. FOG LAYER: DQN TO SELECT OPTIMAL OFFLOADING FOG NODE

This Layer processes tasks that have moderate computing and communication capacities with moderate delays and require moderate resources and computing capacity for servicing. At this stage, the proposed approach uses the K-means

TABLE 4. Fuzzy logic rules.

Fuzzy inputs				Offloading decisions
Task Size	Computational Intensity	Communication Intensity	Delay tolerance	Target Layer
Minimal	Minimal	Minimal	Minimal	locally
Minimal	Minimal	Minimal	Moderate	locally
Minimal	Minimal	Moderate	Significant	locally
Minimal	Minimal	Moderate	Minimal	locally
Minimal	Moderate	Significant	Moderate	Collaborative fog nodes
Minimal	Moderate	Significant	Significant	Collaborative fog nodes
Minimal	Moderate	Minimal	Minimal	Collaborative fog nodes
Minimal	Moderate	Minimal	Moderate	Collaborative fog nodes
Moderate	Significant	Moderate	Significant	Cloud
Moderate	Significant	Moderate	Minimal	Collaborative fog nodes
Moderate	Significant	Significant	Moderate	Collaborative fog nodes
Moderate	Significant	Significant	Significant	Cloud
Moderate	Minimal	Minimal	Minimal	locally
Moderate	Minimal	Minimal	Moderate	locally
Moderate	Minimal	Moderate	Significant	Collaborative fog nodes
Moderate	Minimal	Moderate	Minimal	Collaborative fog nodes
Significant	Moderate	Significant	Moderate	Collaborative fog nodes
Significant	Moderate	Significant	Significant	Cloud
Significant	Moderate	Minimal	Minimal	Collaborative fog nodes
Significant	Moderate	Minimal	Moderate	Collaborative fog nodes
Significant	Significant	Moderate	Significant	Cloud
Significant	Significant	Moderate	Minimal	Collaborative fog nodes
Significant	Significant	Significant	Moderate	Cloud
Significant	Significant	Significant	Significant	Cloud

clustering algorithm to cluster fog nodes into different clusters. For example, the distance between two fog nodes (x1, x2) and f(y1,y2) is computed by Euclidean distance as in(1):

$$\sqrt{(x2-x1)^2+(y2-y1)^2} \tag{1}$$

According to [32], the K-means clustering process can be described as follows:

Step 1: Select the number of clusters (k)

Step 2: Randomly choose k data points to serve as initial centroids for the clusters.

Step 3: Assign each data point to the cluster with the closest centroid.

Step 4: Calculate new centroids for each cluster based on the assigned data points.

Step 5: Repeat steps 3 and 4 until convergence, meaning that the centroids no longer change.

Each cluster has one fog cluster controller (represented in yellow color) that is responsible for managing resources across the fog cluster and effectively distributing the workload across the cluster. The fog cluster controller is also responsible for determining which fog node can process the incoming request. Every fog node in a cluster communicates with a fog cluster controller (FCC). The fog controller (FC) is responsible for linking all FCCs of the Fog layer. In the process of task offloading, if a particular cluster in a fog layer is unable to handle a task because of processing requirements, the fog controller checks whether the task can be sent to a different cluster. The task is then forwarded to the alternative cluster if the response is positive; otherwise, it moves to the cloud layer.

Algorithm 2 Offloading Decisions Using Fuzzy Logic Algorithm

Input: *Application' Task T_i with parameters (Task Size, Computational Demand, Communication Demand and Delay Constraint), number of computing nodes CN_i*
Output: *Optimal Offloading decision (Locally, Collaborative fog, Cloud)*

```
For all Tasks in Ti do
F=FuzzyLogicSystem(TaskSize,
    ComputationalDemand,
    CommunicationDemand, and delayConstraint)
    If F≤ FLow then
        Allocate Ti on IoT layer
    else
    If F≤ Fmedium then
        Allocate Ti on collaborative fog
        Start Algorithm 4
    else
        Allocate Ti on cloud
        End if
    End if
End for
```

The use of DQN can help in making intelligent decisions for task offloading and balancing the workload, considering factors such as the processing capabilities of fog servers, network conditions, and task requirements. A deep Q-network (DQN) is a widely employed reinforcement learning technique that extends the Q-learning paradigm by leveraging neural networks to approximate Q-values. This method has been used to solve several scheduling and allocation problems in fog computing environment [33]. Instead of a Q-table, DQN utilizes a Q network and a target network to enhance learning stability [34].

According to [35] and [36], the steps of DQN involve: (1) setting up the environment by defining the states, actions, and rewards. In this case, the state can represent information regarding the current task and current fog node status. The action can represent the allocation of a task to the best fog node among collaborative fog nodes, and the reward can represent the performance metric (e.g., minimizing

processing time) associated with task execution. (2) Initialize the DQN agent with random weights for both the main neural network (Q-network) and target network. (3) interact with the environment to collect data on the current state, available tasks, and capabilities of each fog server. These data can be used to calculate the Q-values for each state-action pair and to store experiences in an experience replay memory. (4) The collected data were used to estimate the Q-values for each state-action pair in the Q-network.

(5) The target network is updated by copying weights from the main neural network. (6) Calculate the loss function based on the difference between the target Q-values and the Q-values estimated from the Q-network. (7) trains the Q-network to minimize the loss function by adjusting its weight. This can be achived using a gradient descent algorithem or other optimization algorithms. (8) Based on the Q-values estimated by the Q-network, the best action (i.e., the most powerful fog server) is selected to execute, incorporating an epsilon-greedy policy for exploration and exploitation. (9) Execute the selected action, and update the environmental state accordingly. (10) Repeat steps 3-9 until the desired level of performance is achieved or the environment reaches a stable state. The dataset, in this context, is generated through the agent's interactions with the environment during the training process. It consists of the experiences (state, action, reward, next state) stored in the experience replay buffer.

Throughout this process, the DQN agent learns to choose the most powerful fog server and achieves load distribution among the fog servers by minimizing the loss function and updating the Q-network. This approach can help optimize task placement and resource allocation in fog computing environments, by considering factors such as the processing capabilities of fog servers, network conditions, and task requirements.

In the context of a DQN, the update mechanism of Q-learning relies on the Bellman equation, seeking to minimize the temporal difference error. The Q-value for a state-action pair is updated towards the target Q-value, which is a combination of the immediate reward and maximum estimated Q-value for the next state. The Bellman equation is as follows:

$$Qtarget(s, a) = r + \gamma maxa' Q\left(s', a'; \theta-\right) \tag{2}$$

Here, Qtarget (s, a) refers to the target Q-value. The variable r signifies the immediate reward received after acting in states, γ is the discount factor, indicating the significance of future rewards (usually between 0 and 1), s′ is the resulting state, a′ is the action in the subsequent state, and maxa′ Q (s′,a′;$\theta-$) represents the maximum estimated Q-value for the next state over all possible actions a′, where $\theta-$ denotes the weights of the target Q-network.

The loss function used to update the Q-network is typically the Mean Squared Error (MSE) loss between the predicted Q-values Q(s,a;θ)) and target Q-values Qtarget(s,a). The overall training objective was to reduce the MSE loss by modifying the weights (θ) of the Q-network. The loss function used for training the Q-network in a DQN is defined as:

$$MSE\ Loss = \frac{1}{N}\sum_{i=1}^{N}\left(Q\left(s_i, a_i; \theta\right) - Qtarget\left(s_i, a_i\right)\right)^2 \tag{3}$$

Here, N is the batch size, and (s_i, a_i, r_i,r_i') represents the state, action, reward, and next state in the i-th experience sampled from the replay buffer.

Action selection is facilitated through an epsilon-greedy policy that offeres a nuanced balance between exploration and exploitation. Utilizing an ε-greedy policy, the selection of the action a_t is defined as follows:

With a probability of ϵ, a random action is chosen.

Otherwise, a_t is determined as

$$argmax_a Q(\phi(st), a; \theta) \tag{4}$$

Once at is established, the agent executes the action in the environment, observes the resulting reward Rt, and transitions to the subsequent state S_{t+1}. This experience tuple (S_t, A_t, R_t, S_{t+1}) is diligently stored in the experience replay buffer, which is a crucial storage space for the historical Q-value data used to update the Q-network from the target Q-network. This meticulous process enhances the stability and efficiency, allowing the agent to make informed decisions in dynamic environments.

The pseudo-code of the algorithm for optimal fog node selection, using the Deep Q-Network (DQN) method involves a step-by-step approach in training and testing for efficient task offloading in a fog computing environment, as presented in Algorithm 3. Initially, tasks and fog nodes are organized based on their features, laying the groundwork for prioritization and efficient allocation. The DQN parameters and networks were set up, and a replay buffer was created to store the experience. In the training phase, the algorithm continuously explores and performs actions, adjusting Q-network weights according to observed rewards, and periodically syncing a target network. This continual refinement enables the algorithm to make better decisions as it learns. During testing, the algorithm used the acquired Q-values to optimally assign tasks to fog nodes, showcasing its learned insights. The distinct training and testing phases ensure that the algorithm learns from the environment and then applies the acquired knowledge for efficient fog node assignment. The trained agent is evaluated on our offloading scenarios to assess its ability to make effective task-offloading decisions, which integrate exploration, experience replay, and target network synchronization to enhance decision-making in dynamic fog computing scenarios.

Finally, if the task is not executed locally or by a collaborative fog node owing to limited resource capacity, the task will be offloaded to the powerful computational resource layer 3 which is the Cloud Layer.

C. CLOUD LAYER

In Layer 3, if the task is not executed locally or the collaborative fog node is due to the input load, it decides to

Algorithm 3 Optimal Fog Node Selection Using DQN Method

Input: *Set of Task T_i and Set of Fog Node F_i*
Output*: Assignment of Tasks Ti to Optimal Fog Devices Fi based on learned Q-values*
Initialization
Sort Task in ascending order based on Minimum Required Delay
Sort Fog Node in descending order based on computational capacity
Initialize DQN parameters (numEpisodes, learningRate, discountFactor, epsilon, etc.)
Initialize Q-network with random weights (W_Q) and Target network with weights (W_Target = W_Q)
Initialize experience replay buffer to store tuples (state, action, reward, next state)
Set update_interval for updating the Target network weights periodically
Set update_counter = 0
Steps for Training Phase
For episode = 1 to numEpisodes do
  For each incoming Task in Ti do
    Initialize the current state S
    For state = 1 to LastState do
      Put the current state S into the target network and obtain the Q values for all fog nodes: Q_values = Q_target(S; θ_target)
      Select the best action (a) using epsilon-greedy policy based on Q-values from Q-network for the current state (S) with probability epsilonTrain, or choose randomly with probability 1 – epsilonTrain
      Execute actionA_t in the environment, observe reward R_t, and transition to the new state S_{t+1}
      Save the experience (S_t,A_t ,R_t ,S_{t+1})}) in the replay memory D
      Sample a random batch from the experience replay buffer
      Update Q-network weights (W_Q) to minimize the Mean Squared Error loss
      Increment update_counter
      If update_counter is divisible by update_interval, update Target network weights (W_Target) by copying *the Q-network weights (W_Q)*
      Set current state S to the next state S'
    End for
  End for
End for
Testing Phase
Initialize task assignments: TaskAssignments = []
Initialize Q_values for all tasks: Q_values_all_tasks = []
For each Task in Ti do
    Put the current state S into the Q network and obtain the learned Q-values for all fog nodes Q_values = Q_target(S; θ_target), Append Q_values to Q_values_all_tasks
    Select the fog node with the highest Q-value as the optimal destination for the task: optimal_fog_node, Add (OptimalTasks, FogNodes) to TaskAssignments
    Transfer current state S to the next state S'
End for
Return TaskAssignments list

offload the computational task to the remote cloud. All tasks in cloud computing are executed using the Lowest Delay First Scheduling algorithm, prioritizing tasks based on minimizing their expected delays. This approach ensures efficient task processing with a focus on meeting latency requirements and optimizing overall system performance.

V. PERFORMANCE EVALUATION

This section evaluates the performance of the proposed offloading schema by comparing it with benchmark schemes. The evaluation considers performance metrics including (1) average reduction delay time, (2) average energy consumption, (3) network usage, (4) throughput, and (5) offloading rate.

A. CASE STUDY

The scenario under study involves multiple IoT devices, such as smart CCTV systems, self-driving cars, and smartphones. These devices have various applications in the fields of transportation, manufacturing, healthcare, and security. The requirements of such applications include massive data processing and low delays. Within each application, various tasks exist. For instance, smart CCTV applications involve tasks such as motion detection and facial recognition. These

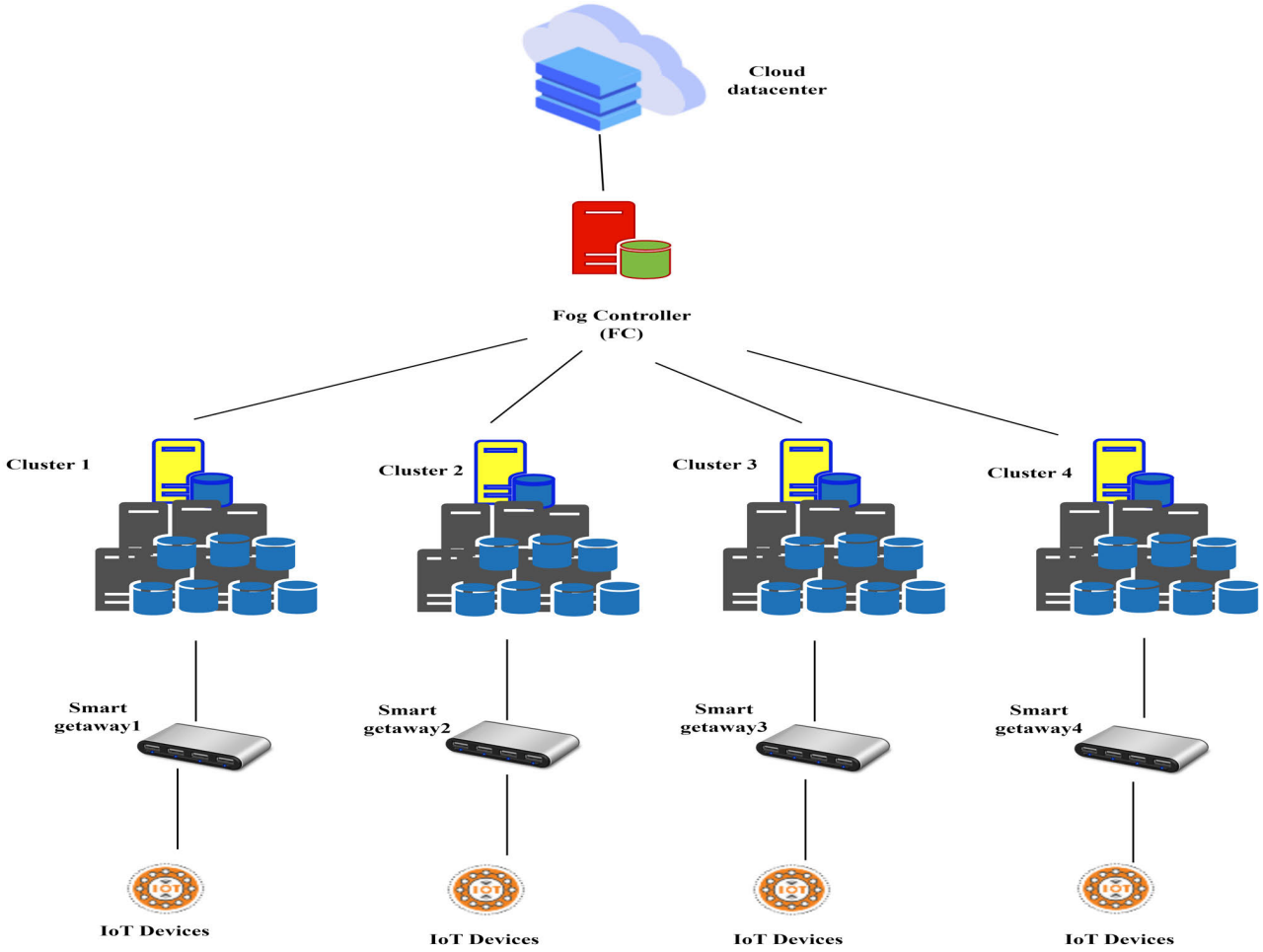

FIGURE 6. Ifogsim network topology.

tasks differ significantly in their specifications, including size, computational demands, communication requirements, and delay. Given these task-specific characteristics and the availability of resources, they can be deployed and executed across various computing resources such as locally, collaborative fog nodes or the cloud. Further details regarding the proposed architectural modes are described in Section III.

B. SIMULATION SETTINGS

The effectiveness of the proposed strategy assessed using the Ifogsim simulator. The experiments were conducted on an Intel i7 2.4 GHz PC with 16 GB RAM, running Windows. The parameters considered for the experiments are listed in TABLE 5, 6,7,8,9, and 10. The iFogSim network topology is illustrated in Figure 6.

TABLE 5. System configuration.

Parameter	Value
Simulation tool	iFogSim
OS	Windows 11 (64-bit)
CPU	an Intel i7 2.40 GHz
Memory	16 GB RAM
Language	Java
IDE	NetBeans 18
Development kit	JDK-20.0.2

TABLE 6. Simulation parameters.

Parameter	Value
Cloud data center	1
Number of fog nodes	32 (8*4)
Number of smart gateways (intermediate nodes)	4 nodes
Number of tasks	100

C. BENCHMARK SCHEMES

To demonstrate the efficacy of the proposed strategy, we compare it with the following baseline approaches:

1. NON-OFFLOAD: In this approach, the execution of user tasks occurs locally. Thus, the tasks are not offloaded to either the fog nodes or cloud.

TABLE 7. Configurations of computational resources.

Property	Cloud	Fog node Controller	8 fog nodes in each cluster	Smart gateway
MIPS	44,800	2800	1500-2000	750-1400
RAM	40,000	4000	1500-3000	750-1400
Upload Bandwidth (Mbps)	10,000	3000	1500-2500	750-1200
Download Bandwidth (Mbps)	10,000	3000	1500-2500	750-1200
Level	0	1	2	3

TABLE 8. Configurations of hosts.

Parameter	Value
OS	Linux
Architecture	×86
BW	10,000 B/S
Storage	1,000,000 B
Time zone	10
VM model	Xen
Cost	3
Cost per storage	0.01
Cost per memory	0.05

TABLE 9. Tasks' property.

Tasks' Property	Minimal	Moderate	Significant
Task size	0-5000	5000-10000	(10000-15000
Computational demand (MIPS)	500-2000	2000-3000	3000-500
Communication demand (Mbps)	10- 20	20-30	30-50
delay	1ms-50ms	50ms-200ms	200ms-500ms

2. FIRST FIT (FF): In this method, the determination of the destination of the task relies on the order of the fog nodes or cloud in the network [37]. The selection process involved checking the suitability of the first fog node for offloading; if it was considered appropriate, it was chosen. Otherwise, the algorithm evaluates the second fog node, and this sequential evaluation continues until the final fog node is considered, followed by the cloud.
3. GASDEO (GREEDY AUTO-SCALING DEEP REINFORCEMENT LEARNING-BASED OFFLOADIN): In this approach, a Deep Reinforcement Learning (DRL) algorithm is employed to select the suitable destination for task execution. In the GASDEO strategy, an offload decision is made to minimize the average delay. The proposed method involves the initial local evaluation of fog nodes using a greedy technique, prioritizing sibling nodes before considering the parent node. In the next step, the best destination for task execution was determined using Deep Reinforcement Learning (DRL), as detailed [16].
4. NAFITO-FLA (A NOVEL APPROACH FOR IOT TASKS OFFLOADING BASED FUZZY LOGIC ALGORITHEM): In this method, fuzzy logic has been used to identify the suitable location of the offloading tasks. For more details, please refer to the method described in [20].

TABLE 10. DQN 's parameters.

Parameter	Description	Value
numEpisodes	Maximum number of episodes for training	100
learning Rate	Learning rate for the optimizer	0.2
Discount Factor	Discount factor for future rewards	0.99
update_interval	Frequency of updating the Target network weights	20
update_counter	Counter to track when to update the Target network weights	(Initialized to 0)
W_Q	Weights of the Q-network	(Randomly initialized)
W_Target	Weights of the Target network (initialized with W_Q)	(Initialized with W_Q)
Ther size of Experience replay memory Replay	Size of the experience replay buffer	1000
Epsilon	Exploration-exploitation trade-off parameter (probability of choosing a random action)	0.1
epsilonTrain	Probability for choosing the best action during training using epsilon-greedy policy	0.8
Batch Size	Number of experiences sampled from the replay buffer in each update step	30
Loss Function	Loss function used to train the Q-network	Mean Squared Error

D. PERFORMANCE METRICS

In this section, performance parameters and their mathematical formulations are defined.

1) DELAY

A delay or latency occurs when a task is sent to a corresponding layer for processing. Thus, the delay rate differs for each task type based on its specific characteristics. For example, the time required to process and offload a task locally differs from that required to process and execute tasks remotely. The delay can be estimated in milliseconds and is crucial for evaluating the performance of fog computing systems. Various studies have focused on minimizing delay in fog computing environments, particularly in the context of latency-sensitive applications. The delayl evel associated with a task offloaded by an IoT device is influenced by

four factors: queuing delay (Q_d), transmission delay (T_d), propagation delay (Pro_d), and processing delay (P_d). The total delay was calculated using Equation (5).

$$Delay = Q_d + T_d + Pro_d + P_d \tag{5}$$

2) ENERGY CONSUMPTION

The energy consumption is computed for the entire network topology using equation (5), which considers the current energy consumption (E_C), current time (T_n), last utilization update time (, T_{lu}), and host power at the last utilization update (P_h). Various studies have focused on developing energy-efficient models for fog computing in the context of the Internet of Things (IoT). These models aim to reduce electric energy consumption of the nodes in the IoT and address the challenges of power consumption and delay in fog computing networks.

$$E = E_C + (T_n - T_{lu}) * P_h \tag{6}$$

3) NETWORK USAGE

The utilization of network usage depends on the size of the transferred task at a specific time, multiplied by latency. The network usage is calculated using Equation (7).

$$Network\ usage = Latency * tupleSize \tag{7}$$

4) THROUGHPUT

Throughput generally refers to the amount of data or information that can be processed within a given period. It is measured in bits per second (bps). This can be mathematically written as follows:

$$TH = \frac{N_{spt}}{T_i} \tag{8}$$

where, N_{spt} is the total number of tasks that were successfully transmitted. And T_i is the time taken for the transfer tasks.

5) OFFLOADING RATE

It is defined as the rate at which tasks or workloads are offloaded or transferred from a local device to a fog node, cloud, or another collaborative fog node. The primary objective of offloading rate is to optimize resource utilization and reduce the computational burden on local devices by transferring tasks to more powerful or available resources in a network. To mathematically represent the offloading rate, you can use the following formula:

$$OFR = \frac{T_f}{N} * 100 \tag{9}$$

where:, T_f is the total number of tasks that have been successfully offloaded or transferred and N is the total number of tasks including both local and offloaded tasks.

E. RESULTS AND DISCUSSION

In this section, the result and discussion of the proposed schema to evaluate the performance parameters are discussed.

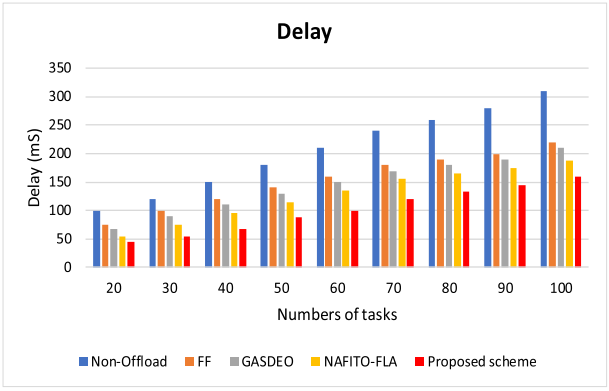

FIGURE 7. Delay vs tasks.

1) DELAY

Firstly, we consider the average delay per task that occurs as a result of task offloading. Figure 7 illustrates the performance of the proposed approach in comparison to benchmark schemes. The figure indicates that with a growth in the number of tasks, there is a corresponding increase in the average delay. This outcome is expected as the increase in offloaded tasks creates more load on the fog nodes, resulting in queuing delay and increased demand for network bandwidth.

Furthermore, we note that the proposed technique demonstrates superior performance compared to the benchmarks in each case. Specifically, the proposed system can decrease the latency by 53%, 34%, 32%, and 23 %, compared to the Non-Offload FF, GASDEO, and NAFITO-FLA schemes, respectively. The baseline scheme does not consider the optimal fog node to process tasks which leads to decreased service latency. Nevertheless, the Non-Offload approach experiences the highest delay because of the local processing tasks.

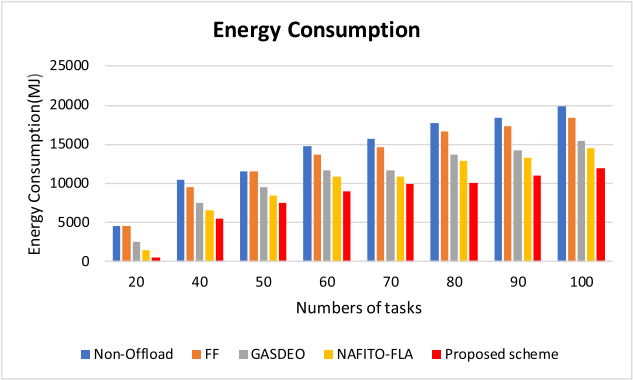

FIGURE 8. Energy consumption vs tasks.

2) ENERGY CONSUMPTION

Figure 8 illustrates the efficiency of the proposed approach in comparison to the existing approaches, specifically in terms of the average energy consumption during task offloading.

The figure indicates that energy consumption increases almost linearly with an increase in the number of tasks. In particular, the suggested method can decrease energy usage by 45%, 37%, 21%, and 12%, compared to Non-Offload FF, GASDEO, and NAFITO-FLA schemes, respectively.

This is because our suggested architecture employs an innovative method for job execution and task scheduling. An optimal scheduling method is necessary to minimize energy consumption. In addition, it offloads tasks to more powerful fog nodes when necessary. This can reduce energy consumption on less capable nodes and improve overall efficiency. Moreover, effective task scheduling and computation offloading strategies can help minimize energy consumption by determining the most efficient processing location for tasks. However, the highest level of energy consumption can be observed in the non-offload method due to local processing tasks. Moreover, The FF scheme chooses the destination of the offloading tasks by considering the sequence of fog nodes or cloud in the network and thus, suffers from increased energy. Then, the NAFITO-FLA scheme based fuzzy logic chooses to offload tasks based only on task characteristics without considering the best destination for devices for offloading. Therefore, GASDEO suffers from higher energy consumption as a result of inadequate task scheduling.

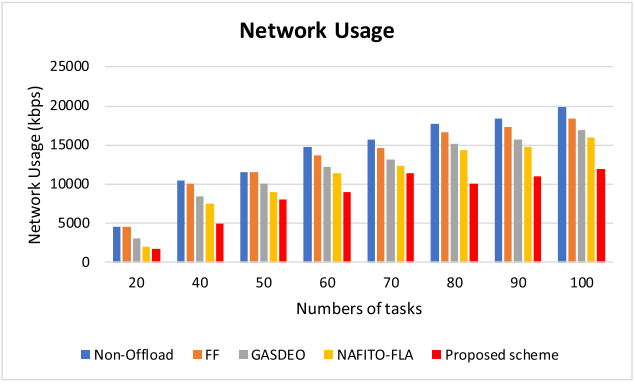

FIGURE 9. Network usage vs tasks.

3) NETWORK USAGE

When the size is an increase in traffic on the cloud server, only cloud resources are utilized. Heightened traffic on the cloud server contributes to improved network usage. When it comes to servers that are geographically distributed, each fog node is allocated to a specific geographical area to handle requests from that region. Consequently, network usage decreases in such scenarios. Figure 9 illustrates the outcomes of the network usage comparison. The Non-Offload approach fails to utilize network resources. However, the FF and GASDEO methods had almost similar results to NAFITO-FLA. Consequently, the proposed approach outperformed the other three strategies in terms of the network usage. An increase in network size will require more resources, but, the proposed strategy remains more effective than the alternatives. Our Smart computation offloading strategies can help minimize network usage by determining the most efficient processing location for tasks, whether at the edge, in the fog, or in the cloud. Moreover, efficient load-balancing techniques can distribute tasks and data processing across fog nodes, thereby preventing network congestion and reducing network usage.

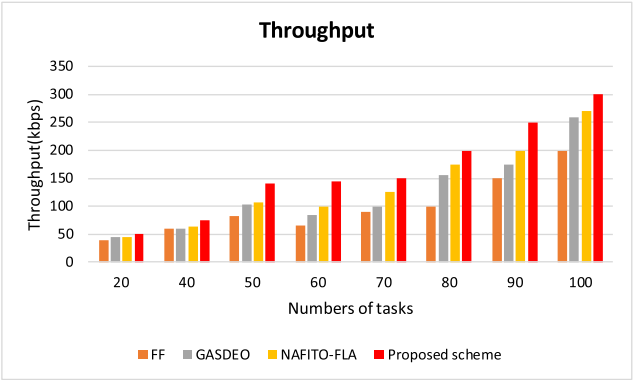

FIGURE 10. Throughput vs tasks.

F. THROUGHPUT

Figure 10 shows the performance of the average throughput versus the number of tasks. It can be seen that the proposed schema performs best in terms of the average throughput compared with the other methods. This is because the proposed approach distributes more tasks across multiple nodes and servers. Therefore. The proposed method allows multiple tasks can be executed simultaneously on different resources by determining the most efficient processing location for tasks. This can lead to a faster task execution and improved throughput. Furthermore, efficient load balancing techniques can distribute tasks and data processing across fog nodes, thereby preventing network congestion and increasing throughput. NAFITO-FLA outperforms GASDEO and FF in terms of average throughput because it makes better task offloading decisions to improve the average throughput.

G. OFFLOADING RATE

It is defined as the rate at which tasks or workloads are offloaded or transferred from a local device to a fog node, cloud, or other collaborative fog node. The primary objective of the offloading rate is to optimize resource utilization and reduce the computational burden on local devices by transferring tasks to more powerful available resources in a network. Figure 11 shows the performance of the average offloading rate versus several tasks. An offloading rate can be considered to provide better performance than other approaches. The reason is that the proposed algorithm efficiently utilizes the resources. When tasks are offloaded from local devices to more powerful remote resources (e.g., fog nodes or cloud servers), better resource utilization can be achieved. This, in turn, can result in improved performance, as tasks are executed on resources that can handle them more efficiently. Implement efficient scheduling algorithms

that can quickly prioritize and allocate tasks to fog nodes. This ensures that tasks are offloaded promptly, increasing the offloading rate. A high offloading rate means that a significant proportion of computational tasks are offloaded from a local device to more powerful resources in the network. This can lead to an improved performance on the local device, reduced energy consumption, and efficient use of resources.

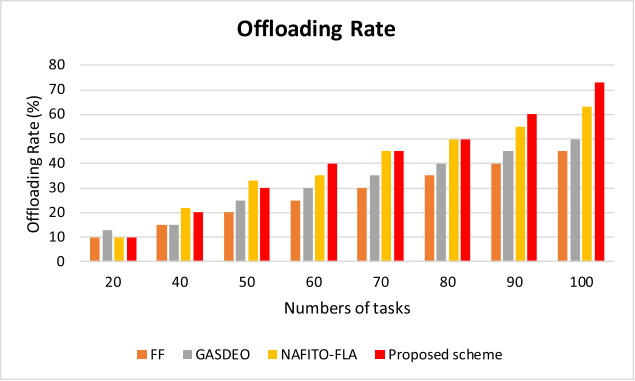

FIGURE 11. Offloading rate vs tasks.

H. LIMITATIONS

This section clarifies the constraints associated with the proposed fog computing methodology, particularly in the realms of static topology, security, and privacy. These limitations are crucial considerations that underscore the challenges and scope of our approach within the context of a fixed network topology.

1) STAIC TOPOLOGY

One limitation of our offloading methodology is its reliance on a static topology, assuming fixed positions for IoT devices and fog nodes. It's worth noting that real-world IoT scenarios often feature mobile nodes in operation, thus introducing a dynamic element that our study does not explicitly account for.

2) SECURITY AND PRIVACY

Fog-cloud systems involve the transmission and processing of sensitive data, raising concerns regarding the security and privacy of information. Inadequate security measures can lead to unauthorized access, data breaches, privacy violations, eroding user trust and system reliability. To mitigate these risks, strong encryption protocols should be implemented for data transmission to protect against eavesdropping and unauthorized access. Robust access control mechanisms must be in place to ensure that only authorized entities can access and modify critical system components. Privacy-preserving techniques, such as differential privacy, should be explored to safeguard user data during processing. Compliance with data protection regulations and standards is crucial to address the legal and ethical aspects of data handling. Integrating security measures throughout the entire architecture, involving regular security audits, and engaging stakeholders are essential components of a comprehensive security strategy.

VI. CONCLUSION

In this study, we propose a novel offloading strategy to enhance IoT application performance in fog environments. The main idea of the proposed schema involves the processing of sensed data generated from IoT devices in different layers of the fog computing architecture. Fuzzy logic algorithms were used to make optimal decisions regarding task offloading to the target processing layer, whether at local fog nodes, collaborative fog nodes, or the cloud while considering four different task requirements. Subsequently, DQN method-based deep reinforcement learning was used to select the optimal fog node among the collaborative fog nodes to offload tasks. Subsequently, we proposed a lowest deadline-first scheduling method to schedule tasks for all computational resources. Finally, we compared the simulation outcomes across several performance metrics and demonstrated that our proposed offloading strategy in fog architecture outperforms other existing methods.

In future work, we will focus on task offloading without considering security parameters. For example, what will happen if we offload a task to a compromised fog node? We are currently considering this feature for future implementation. Additionally, the study employed a static topology, assuming fixed positions for IoT devices and fog nodes. Real-world Internet of Things scenarios may involve mobile nodes in operation. As a result, we aim to further investigate the impact of dynamic topology on the efficiency of the offloading scheme in the next stage of this research.

REFERENCES

[1] A. Ahmed, S. Abdullah, S. Iftikhar, I. Ahmad, S. Ajmal, and Q. Hussain, "A novel blockchain based secured and QoS aware IoT vehicular network in edge cloud computing," *IEEE Access*, vol. 10, pp. 77707–77722, 2022, doi: 10.1109/ACCESS.2022.3192111.

[2] Y.-L. Jiang, Y.-S. Chen, S.-W. Yang, and C.-H. Wu, "Energy-efficient task offloading for time-sensitive applications in fog computing," *IEEE Syst. J.*, vol. 13, no. 3, pp. 2930–2941, Sep. 2019, doi: 10.1109/JSYST.2018.2877850.

[3] M. Izhar, S. A. A. Naqvi, A. Ahmed, S. Abdullah, N. Alturki, and L. Jamel, "Enhancing healthcare efficacy through IoT-edge fusion: A novel approach for smart health monitoring and diagnosis," *IEEE Access*, vol. 11, pp. 136456–136467, 2023, doi: 10.1109/ACCESS.2023.3337092.

[4] Q. D. La, M. V. Ngo, T. Q. Dinh, T. Q. S. Quek, and H. Shin, "Enabling intelligence in fog computing to achieve energy and latency reduction," *Digit. Commun. Netw.*, vol. 5, no. 1, pp. 3–9, Feb. 2019, doi: 10.1016/j.dcan.2018.10.008.

[5] N. K. Suryadevara, "Energy and latency reductions at the fog gateway using a machine learning classifier," *Sustain. Comput., Informat. Syst.*, vol. 31, Sep. 2021, Art. no. 100582, doi: 10.1016/j.suscom.2021.100582.

[6] K. Gasmi, S. Dilek, S. Tosun, and S. Ozdemir, "A survey on computation offloading and service placement in fog computing-based IoT," *J. Supercomput.*, vol. 78, no. 2, pp. 1983–2014, Feb. 2022, doi: 10.1007/s11227-021-03941-y.

[7] S. Bebortta, S. S. Tripathy, U. M. Modibbo, and I. Ali, "An optimal fog-cloud offloading framework for big data optimization in heterogeneous IoT networks," *Decis. Anal. J.*, vol. 8, Sep. 2023, Art. no. 100295, doi: 10.1016/j.dajour.2023.100295.

[8] A. Hazra, P. Rana, M. Adhikari, and T. Amgoth, "Fog computing for next-generation Internet of Things: Fundamental, state-of-the-art and research challenges," *Comput. Sci. Rev.*, vol. 48, May 2023, Art. no. 100549, doi: 10.1016/j.cosrev.2023.100549.
[9] E. Gomes, F. Costa, C. De Rolt, P. Plentz, and M. Dantas, "A survey from real-time to near real-time applications in fog computing environments," *Telecom*, vol. 2, no. 4, pp. 489–517, Dec. 2021, doi: 10.3390/telecom2040028.
[10] D. H. Abdulazeez and S. K. Askar, "Offloading mechanisms based on reinforcement learning and deep learning algorithms in the fog computing environment," *IEEE Access*, vol. 11, pp. 12555–12586, 2023, doi: 10.1109/ACCESS.2023.3241881.
[11] M. Sheikh Sofla, M. Haghi Kashani, E. Mahdipour, and R. Faghih Mirzaee, "Towards effective offloading mechanisms in fog computing," *Multimedia Tools Appl.*, vol. 81, no. 2, pp. 1997–2042, Jan. 2022, doi: 10.1007/s11042-021-11423-9.
[12] M. Kaur and R. Aron, "A systematic study of load balancing approaches in the fog computing environment," *J. Supercomputing*, vol. 77, no. 8, pp. 9202–9247, 2021, doi: 10.1007/s11227-020-03600-8.
[13] N. M. Dhanya, G. Kousalya, P. Balarksihnan, and P. Raj, "Fuzzy-logic-based decision engine for offloading IoT application using fog computing," in *Handbook of Research on Cloud and Fog Computing Infrastructures for Data Science*. Hershey, PA, USA: IGI Global, 2018, pp. 175–194, doi: 10.4018/978-1-5225-5972-6.ch009.
[14] J. C. Guevara and N. L. S. da Fonseca, "Task scheduling in cloud-fog computing systems," *Peer-Peer Netw. Appl.*, vol. 14, no. 2, pp. 962–977, Mar. 2021, doi: 10.1007/s12083-020-01051-9.
[15] F. M. Talaat, M. S. Saraya, A. I. Saleh, H. A. Ali, and S. H. Ali, "A load balancing and optimization strategy (LBOS) using reinforcement learning in fog computing environment," *J. Ambient Intell. Humanized Comput.*, vol. 11, no. 11, pp. 4951–4966, Nov. 2020, doi: 10.1007/s12652-020-01768-8.
[16] F. Jazayeri, A. Shahidinejad, and M. Ghobaei-Arani, "Autonomous computation offloading and auto-scaling the in the mobile fog computing: A deep reinforcement learning-based approach," *J. Ambient Intell. Humanized Comput.*, vol. 12, no. 8, pp. 8265–8284, Aug. 2021, doi: 10.1007/s12652-020-02561-3.
[17] S. Sharma and H. Saini, "A novel four-tier architecture for delay aware scheduling and load balancing in fog environment," *Sustain. Comput., Informat. Syst.*, vol. 24, Dec. 2019, Art. no. 100355, doi: 10.1016/j.suscom.2019.100355.
[18] F. Farahbakhsh, A. Shahidinejad, and M. Ghobaei-Arani, "Context-aware computation offloading for mobile edge computing," *J. Ambient Intell. Humanized Comput.*, vol. 14, no. 5, pp. 5123–5135, May 2023, doi: 10.1007/s12652-021-03030-1.
[19] C. Chakraborty, K. Mishra, S. K. Majhi, and H. K. Bhuyan, "Intelligent latency-aware tasks prioritization and offloading strategy in distributed fog-cloud of things," *IEEE Trans. Ind. Informat.*, vol. 19, no. 2, pp. 2099–2106, Feb. 2023, doi: 10.1109/TII.2022.3173899.
[20] J. Almutairi and M. Aldossary, "A novel approach for IoT tasks offloading in edge-cloud environments," *J. Cloud Comput.*, vol. 10, no. 1, pp. 10–28, Apr. 2021, doi: 10.1186/s13677-021-00243-9.
[21] S. Misra and N. Saha, "Detour: Dynamic task offloading in software-defined fog for IoT applications," *IEEE J. Sel. Areas Commun.*, vol. 37, no. 5, pp. 1159–1166, May 2019, doi: 10.1109/JSAC.2019.2906793.
[22] M. Tiwari, S. Misra, P. K. Bishoyi, and L. T. Yang, "Devote: Criticality-aware federated service provisioning in fog-based IoT environments," *IEEE Internet Things J.*, vol. 8, no. 13, pp. 10631–10638, Jul. 2021, doi: 10.1109/JIOT.2021.3049326.
[23] S. Aljanabi and A. Chalechale, "Improving IoT services using a hybrid fog-cloud offloading," *IEEE Access*, vol. 9, pp. 13775–13788, 2021, doi: 10.1109/ACCESS.2021.3052458.
[24] V. Nguyen, T. T. Khanh, T. D. T. Nguyen, C. S. Hong, and E.-N. Huh, "Flexible computation offloading in a fuzzy-based mobile edge orchestrator for IoT applications," *J. Cloud Comput.*, vol. 9, no. 1, pp. 1–18, Dec. 2020, doi: 10.1186/s13677-020-00211-9.
[25] R. Beraldi, C. Canali, R. Lancellotti, and G. P. Mattia, "Distributed load balancing for heterogeneous fog computing infrastructures in smart cities," *Pervas. Mobile Comput.*, vol. 67, Sep. 2020, Art. no. 101221, doi: 10.1016/j.pmcj.2020.101221.
[26] M. Al-khafajiy, T. Baker, H. Al-Libawy, Z. Maamar, M. Aloqaily, and Y. Jararweh, "Improving fog computing performance via fog-2-fog collaboration," *Future Gener. Comput. Syst.*, vol. 100, pp. 266–280, Nov. 2019, doi: 10.1016/j.future.2019.05.015.
[27] M. Adhikari and H. Gianey, "Energy efficient offloading strategy in fog-cloud environment for IoT applications," *Internet Things*, vol. 6, Jun. 2019, Art. no. 100053, doi: 10.1016/j.iot.2019.100053.
[28] A. Kishor and C. Chakarbarty, "Task offloading in fog computing for using smart ant colony optimization," *Wireless Pers. Commun.*, vol. 127, no. 2, pp. 1683–1704, Nov. 2022, doi: 10.1007/s11277-021-08714-7.
[29] M. K. Hussein and M. H. Mousa, "Efficient task offloading for IoT-based applications in fog computing using ant colony optimization," *IEEE Access*, vol. 8, pp. 37191–37201, 2020, doi: 10.1109/ACCESS.2020.2975741.
[30] J. M. Mendel, "Fuzzy-logic systems for engineering," *Proc. IEEE*, vol. 83, no. 9, p. 1293, Jul. 1995.
[31] C. Sonmez, A. Ozgovde, and C. Ersoy, "Fuzzy workload orchestration for edge computing," *IEEE Trans. Netw. Service Manage.*, vol. 16, no. 2, pp. 769–782, Jun. 2019, doi: 10.1109/TNSM.2019.2901346.
[32] T. M. Ghazal, M. Z. Hussain, R. A. Said, A. Nadeem, M. K. Hasan, M. Ahmad, M. A. Khan, and M. T. Naseem, "Performances of K-means clustering algorithm with different distance metrics," *Intell. Autom. Soft Comput.*, vol. 29, no. 3, pp. 735–742, 2021, doi: 10.32604/iasc.2021.019067.
[33] S. Park, Y. Yoo, and C.-W. Pyo, "Applying DQN solutions in fog-based vehicular networks: Scheduling, caching, and collision control," *Veh. Commun.*, vol. 33, Jan. 2022, Art. no. 100397, doi: 10.1016/j.vehcom.2021.100397.
[34] Z. Alexander and B. Brandon, *Deep Reinforcement Learning in Action*. New York, NY, USA: Manning Publications, 2020.
[35] S. Gupta, G. Singal, and D. Garg, "Deep reinforcement learning techniques in diversified domains: A survey," *Arch. Comput. Methods Eng.*, vol. 28, no. 7, pp. 4715–4754, Dec. 2021, doi: 10.1007/s11831-021-09552-3.
[36] S. S. Mousavi, M. Schukat, and E. Howley, "Deep reinforcement learning: An overview," in *Proc. SAI Intell. Syst. Conf. (IntelliSys)*, vol. 2, Sep. 2018, pp. 426–440, doi: 10.1007/978-3-319-56991-8_32.
[37] Q.-H. Nguyen and T.-A. Truong Pham, "Studying and developing a resource allocation algorithm in fog computing," in *Proc. Int. Conf. Adv. Comput. Appl. (ACOMP)*, Nov. 2018, pp. 76–82, doi: 10.1109/ACOMP.2018.00020.

DEZHEEN H. ABDULAZEEZ received the B.S. degree in computer science from Duhok University, in 2011, and the M.S. degree in web application and services from Leicester University, U.K., in 2015. She is currently pursuing the Ph.D. degree in fog computing with the College of Science, Duhok University. Her research interests include fog and cloud computing, the Internet of Things (IoT), semantic web, and web applications and services.

SHAVAN K. ASKAR received the B.Sc. and M.Sc. degrees from the Control and Systems Engineering Department, Baghdad, in 2001 and 2003, respectively, and the Ph.D. degree in electronic systems engineering from the University of Essex, U.K., in 2012. He is currently a CEO with Arcella Telecom. He is also a Full Professor of computer networks with the College of Technical Engineering, Erbil Polytechnic University. He works in the field of networks, that includes the Internet of Things, software-defined networks, optical networks, and 5G information systems engineering. His research interests include 5G, IoT SDN, network virtualization, and fog and cloud computing.