Enhanced Grey Wolf Optimization Dimension Learning for Energy-Efficient Task Scheduling in Edge Computing Environment

Jafar Aminu[1], Rohaya Latip[2], Zurina Mohd Hanapi[3], Shafinah Kamarudin[4], Mustapha Abubakar Giro[5]
Faculty of Computer Science and Information Technology, Universiti Putra Malaysia, Selangor, 43300, Malaysia[1, 2, 3, 4]
Department of Computer Science, Kebbi State University of Science and Technology, Aliero, 1144 Nigeria[1, 5]

***Abstract*—The development of edge computing has facilitated the development of numerous applications with diverse characteristics and stringent quality of service (QoS) requirements; these applications demand significant computational power and have strict time-sensitive constraints. While cloud computing offers seemingly unlimited computational resources, it often fails to meet the real-time demands of certain applications because of the latency introduced by the distance between edge devices and cloud data centers. Edge computing enables computational services closer to edge devices, better fulfilling these time-sensitive demands. Task scheduling that tries to share tasks among diverse virtual machines in an optimum manner concerning overall system performance metrics, such as minimal execution time or reduced energy consumption, is one of the key challenges of this heterogeneous computing environment. Task scheduling is an NP-complete problem. Therefore, metaheuristic algorithms are usually applied to obtain near-optimal solutions. The study presents an enhanced grey wolf optimization hybridized by a dimension learning-based strategy, EGWODLB, for optimizing QoS objectives focusing on execution time and energy consumption. The experimental results reflect that EGWODLB outperforms the benchmark algorithms by achieving significant improvements in both execution time, energy consumption, and VM utilization.**

Keywords*—Edge computing; energy consumption; execution time; task Scheduling; grey wolf optimization dimension learning

I. Introduction

With the penetration of intelligent edge devices, the 21st century turns computationally intensive with several broad applications ranging from online gaming, video conferencing, and 3D modeling. Such resource-intensive computing applications are normally restricted in edge devices due to the limited battery life and processing power, resulting in poor user experiences for resource-intensive operations [1][2]. The concept of Edge Computing, therefore, came to alleviate this, offloading expensive computation into local EC servers, hence making performance great and saving energy and execution time, the important role of EC in building the spaces where the demands of modern applications and the limited abilities of edge devices have equipped it to become a key technological component for time-critical applications. For instance, autonomous vehicles, smart cities, and healthcare [3]. Despite the considerable advantages of Edge Computing (EC), the task scheduling problem in these contexts is far from trivial. Huge search spaces can randomly generate satisfactory task assignments [4]. Scheduling algorithms take a considerable amount of time to converge, which can lead to the underutilization of virtual machines. This inefficiency circumscribes the maximum potential of edge computing systems and negatively affects the timely execution of tasks, especially in IoT and real-time data processing[5].

To address these challenges, EC operates within an environment that aims to optimize resource allocation and performance by distributing tasks between edge devices and EC servers. This distribution is based on various factors such as task requirements, network conditions, and resource availability. However, inefficiencies in scheduling algorithms continue to limit this potential, as they may not fully exploit the capabilities of available resources [6]. Effective task scheduling is pivotal in ensuring that EC systems can operate efficiently. This involves the intelligent allocation of tasks to appropriate resources, considering multiple optimization goals, such as minimizing energy consumption, execution time, and completion delay, while ensuring that the Quality of Service (QoS) standards are met [7]. However, the EC environment needs to manage resources efficiently, which requires effective task scheduling. It involves the intelligent allocation of resource tasks, taking into account various optimization objectives such as energy consumption, execution time, and completion delay, all to maintain the Quality of Service (QoS) requirement [8].

Many current algorithms, such as traditional heuristic and metaheuristic approaches, struggle with the flexibility needed to effectively manage the dynamic and constantly evolving conditions of edge environments. These algorithms have gained popularity for task scheduling in EC, particularly for NP-complete problems. The general idea of such algorithms is that they can reasonably perform in limited amounts of time to obtain near-optimal solutions for complex problems.[9]. These include (PSO, GA, FOA, and ACO); all these algorithms have their goal of reaching as near an optimal solution as possible to the complex problem at hand within reasonable time limits[10], [11]. However, these methods are suffering from shortcomings such as a slow convergence rate, inherent randomness tied to global search capability, and trapping in local optima that usually result in suboptimal solutions[12]. o far, the biggest challenge in applying such metaheuristic approaches to real-time edge computing is how to find a good balance between global exploration and local exploitation

[13],[14]. Recently, many challenges have been overcome by using the algorithm called Grey Wolf Optimizer. GWO has some special advantages in comparison with other metaheuristic algorithms. For example, memory is lower than that of PSO since only one position vector is used, and GWO avoids convergence to local optima because of the use of the top three solutions, while the PSO algorithm focuses on one best solution[15]. Besides, GWO has less complexity and fewer parameters than GA, implying faster computation time and lower energy consumption. It also shows better adaptation to the dynamic resource availability inherent in EC environments [16].

The GWO takes inspiration from grey wolves, always living in a structured social hierarchy, and their natural behavior and hunting strategy. More recently, the grey wolf optimizer has gained considerable attention over other metaheuristic algorithms due to efficient convergence performance, low complexity, minimum utilization of energy, and easy implementation converter [17]. The GWO has been widely used in many fields to solve a set of problems, such as optimization problems, classification problems, economic and power dispatch, and Capacitated Vehicle Routing problems [18]. Recently, with its attractive characteristics, this algorithm has gained specific attention from researchers in multi-objective problems of task scheduling, especially in edge computing, given the particular features of EC servers [19]. The major influencing factors include execution time and energy consumption, the most important among them all. Though scheduling tasks on EC servers reduces execution time through the exploitation of proximity to resources, it could lead to higher energy consumption. Overall, the rise in energy consumption can be attributed to several factors, including the added computational overhead required for efficient task management and mapping, frequent scheduling in dynamic environments, and the ongoing monitoring and adjustment of resource utilization to accommodate fluctuating demands.[20]. Moreover, most edge devices are limited in computation capability and power supply, and high energy consumption within an edge server increases not only operation costs but also shortens equipment life. Consequently, energy consumption reduction becomes critical to edge networks. Most of the work in improving energy efficiency in edge servers has focused either on developing energy-efficient hardware for edge computing or strategies of offloading jobs from the cloud to an edge server. Workload scheduling and task assignment to an appropriate virtual machine, though critical aspects that can significantly enhance EC server execution time and energy efficiency, have not yet been clearly addressed [21].

To address the identified limitations, this study proposes an enhanced version of the GWO, designed to improve its efficiency and effectiveness. The enhancement incorporates a novel search mechanism known as the dimension learning-based Hunting (DLH) strategy, inspired by the individual hunting behaviour of wolves in nature. The DLH strategy expands the global search capability by leveraging multi-neighbour learning, allowing the algorithm to explore the search space more thoroughly. In each iteration, the (EGWO) generates two potential solutions for each wolf: one based on the (DLH) strategy and another from the standard GWO approach. The algorithm then applies a selection and update process to determine the best candidate solution, ensuring that the wolves move to more optimal positions in subsequent iterations. Furthermore, a network performance model is integrated to assess critical performance metrics. This model minimizes delays by effectively assigning tasks to the most suitable virtual machines, resulting in improved system performance and efficiency.

The main contributions of this study are as follows:

- The existing research and challenges related to task scheduling, particularly those related to energy consumption and execution time, are examined as multi-objective optimization problems.
- A mathematical framework optimizes energy consumption, execution time, and VM utilization by allocating tasks to virtual machines.
- Introducing an enhanced EGWODLB strategy to improve task scheduling, aiming to reduce both execution time and energy consumption in edge computing environments
- Through simulations, the proposed EGWODLB algorithm significantly outperforms other methods in minimizing energy consumption, execution time, VM utilization, and running time.

This study is organized as follows: The relevant research is reviewed in Section II. The system model is examined in Section III. Section IV describes the suggested approach in depth. Section V provides an explanation of the experimental setting, and Section VI discusses the findings. The study is finally concluded in Section VII with a review of the results and recommendations for additional research.

II. Related Works

Edge computing has emerged as a promising approach to complement cloud computing by improving resource management and enhancing performance[22]. However, the inherently dynamic and varied nature of edge environments creates significant challenges, particularly in areas such as task offloading, network performance monitoring, and the handling of task dependencies [23]. Task offloading within edge computing has become a critical concern, attracting considerable research interest. A range of metaheuristic algorithms has been investigated to tackle this issue, with each method offering distinct advantages and drawbacks. These algorithms play an essential role in optimizing task management decisions, whether on edge devices or EC servers, across various computing settings, due to their capacity to find near-optimal solutions in complex and evolving environments [21].

The Particle Swarm Optimization (PSO) algorithm has become a popular choice for task scheduling in edge computing environments due to its ability to efficiently find optimal solutions for complex problems [24]. Over time, various modifications and enhancements have been introduced to enhance the Particle Swarm Optimization (PSO) algorithm's effectiveness in task scheduling. Drawing inspiration from the

social behaviours of animals such as birds and fish, these advancements have enabled the algorithm to adapt to the dynamic and flexible demands of edge computing. As a result, PSO has become a cost-effective and energy-efficient solution that meets strict Quality of Service (QoS) requirements. Recently, a two-level scheduling framework has been developed to tackle optimization challenges in edge computing environments. At the upper level, the Inertia Weight Particle Swarm Optimization (IW-PSO) method addresses edge users' complex, non-convex, and nonlinear power allocation problem, focusing on minimizing transmission energy consumption. A binary PSO algorithm is utilized at the lower level to handle the joint task offloading and resource allocation problem, modelled as a Mixed-Integer Nonlinear Programming (MINP) problem. This hierarchical framework achieves an optimal scheduling solution, reducing energy consumption while enhancing response times and overall system efficiency [25]. This study introduces a hybrid method integrating Genetic Algorithms (GA) and Particle Swarm Optimization (PSO) to optimize multi-objective job scheduling in fog computing environments. Combining both algorithms' strengths, the hybrid approach outperforms traditional single-algorithm techniques. It optimises between exploring and exploiting the search space, resulting in more effective solutions. Experimental findings reveal that the proposed hybrid algorithm significantly enhances execution time and reduces energy consumption, demonstrating its efficiency and practicality [26]. This study proposes a Particle Swarm Optimization (PSO)-based approach aimed at reducing energy costs and minimizing the time required to identify viable solutions. Experimental results indicate that the energy-efficient PSO metaheuristic effectively leverages its capabilities to achieve significant performance improvements [27]. A centralized orchestrator layer is proposed to implement a two-tier cooperative scheduling mechanism. At the first level, tasks are scheduled locally on EC servers, ensuring efficient handling of resources at the edge. The second level operates within the orchestrator, where tasks are distributed to either the cloud or a nearby base station. Task prioritization is determined based on factors such as required throughput and acceptable delay, enabling the system to optimize resource allocation and performance [28]. This research presents a heuristic Particle Swarm Optimization (PSO) algorithm, termed LPSO, built upon the Lyapunov framework. The algorithm is designed to stabilize queues and complete tasks within defined timeframes while balancing computational energy usage at IoT nodes, transmission energy requirements, and energy consumption in the fog computing environment. By fine-tuning these parameters, the proposed LPSO algorithm effectively reduces total energy consumption and promotes efficient task execution. [29]. The main goal of scheduling is to minimize task completion time and lower energy consumption on edge devices. The study introduces two optimization techniques grounded in slow-motion particle swarm optimization to address this NP-hard challenge. Notably, a position-based mapping strategy is developed to convert particle positions into practical and efficient scheduling solutions [30]. The Particle Swarm Optimization (PSO) algorithm is highly effective for solving continuous optimization problems. It is renowned for its straightforward implementation and fast convergence speed. However, it is prone to premature convergence, which can result in suboptimal solutions as it becomes trapped in local optima. Consequently, additional enhancements are required to improve its overall performance.

Genetic Algorithms (GAs) have demonstrated their effectiveness in addressing complex task scheduling problems across diverse computing environments, including cloud computing, Edge Computing (EC), and fog computing [31]. Their primary strength is their capacity to produce excellent results quickly, even for NP-complete problems like job scheduling, which makes them an invaluable tool in these situations [32]. In order to minimize the total energy consumption of fog nodes (FNs) while ensuring that the Quality of Service (QoS) requirements for Internet of Things tasks are met, the research presents a mathematical model for the task scheduling problem. [33]. This study introduces an enhanced task scheduling strategy, IGA-TSPA, based on an improved genetic algorithm. The approach refines the initialization and mutation processes of the genetic algorithm, effectively narrowing the initial solution space and accelerating convergence toward optimal solutions [34]. A scheduling system was created to break down applications into discrete jobs, which are then handled using a task queue. An auction-based bidding approach was used to find the most appropriate server for each application [35]. The Genetic Algorithm (GA) employs a weighted probabilistic crossover method to optimize work scheduling, considering the fitness of Virtual Machines (VMs).

The Fruit Fly Optimization Algorithm (FOA) has been widely used for job scheduling challenges in many fields, significantly improving efficiency and optimization. [36]. In recent research, an improved Fruit Fly Optimization (IFFO) algorithm was designed to optimize the scheduling of multiple workflows in a cloud computing environment by minimizing both makespan and associated costs [37]. A Fruit Fly-inspired Simulated Annealing Optimization System (FSAOS) was proposed as a potential solution. To address the issue of premature convergence in both global and local searches, the framework integrates simulated annealing into the optimization process. Additionally, a trade-off factor is introduced, allowing application owners to select the most suitable service quality that minimizes execution costs [38]. The Firefly Algorithm (FA) excels in solving multimodal optimization challenges and effectively avoids getting trapped in local optima. However, its effectiveness in dynamic environments largely relies on precise tuning of key parameters, such as the smell coefficient and attractiveness function, to ensure optimal performance.

The Ant Colony Optimization (ACO) algorithm has proven to be highly effective in addressing task offloading challenges across various fields, including edge computing and construction project management. Drawing inspiration from the natural behavior of ants, the algorithm mimics their method of leaving pheromone trails to guide others [39]. These pheromone trails enable ants to navigate the solution space, marking their routes and assessing the quality of potential solutions, to converge on the optimal one. Recently, there has been research based on an ant colony. For resource and task selection criteria in clusters, the ACO technique was

proposed in [40]. To attain the required quality for scheduling workflows, users define QoS restrictions. [41] presents a workflow scheduling method based on the ant colony system (ACS) that has been improved with new features. The idea of a knowledge matrix combined with the ACO algorithm is covered in [42]. Researchers employed the ACO technique to assess the historical attractiveness of distributing jobs to the same physical machine. An energy-efficient scheduling system based on ant colonies was invented. Pheromone systems have been revised and proven in [43] and [44]. Due to the balance between exploration and exploitation, it has demonstrated encouraging outcomes in several optimization situations. To evaluate its performance in dynamic situations like edge computing. However, more research is necessary as its applicability in these settings is yet mostly unexplored.

In the realm of task scheduling for edge computing, traditional optimization techniques and artificial ontologies have historically played a central role in research [45]. While these strategies have proven efficient in specific situations, their use is frequently limited by significant computational complexity, resulting in high resource usage and increased energy needs. These constraints represent important issues for EC systems, which operate under strict criteria for minimizing energy usage and execution time to support real-time applications. To address these constraints, researchers have increasingly adopted metaheuristic approaches such as Particle Swarm Optimization (PSO), Genetic Algorithms (GA), the Fruit Fly Optimization Algorithm (FOA), and Ant Colony Optimization (ACO). These techniques stand out for their ability to deliver high-quality solutions with lower complexity than classical methods, enabling more energy-efficient task scheduling. However, their dependency on multiple parameters to achieve optimal results can offset these advantages by increasing computational overhead and energy usage. Consequently, such approaches often fall short of meeting the rigorous performance demands of EC applications, especially in Internet of Things (IoT)-driven environments where rapid response times and power efficiency are paramount. This work proposes the Enhanced Grey Wolf Optimization algorithm integrated with a Dimension Learning-Based strategy (EGWODLB) to overcome these challenges. This novel mechanism addresses the dual objectives of reducing energy consumption and minimizing task execution time in EC systems. By effectively balancing key parameters with minimal input requirements, EGWODLB demonstrates its ability to manage large-scale IoT workloads with remarkable efficiency. Moreover, the algorithm ensures optimized task scheduling, faster response times, and improved system performance, aligning seamlessly with modern EC applications' energy and latency demands. Table I provides a consolidated overview of recent studies on task scheduling in edge computing environments, highlighting both key findings and existing limitations. This summary captures the current state of research, offering insights into the strengths and gaps of various approaches used to address task scheduling challenges in dynamic and resource-constrained edge computing environments.

TABLE I. SUMMARY OF LITERATURE ON TASK SCHEDULING

Algorithm	Strategy Enhancement	Main Goals	Benefits	Limitation	Ref
PSO	Standard PSO	Efficient task scheduling in edge environments	Fast convergence, simple implementation, energy-efficient, but prone to premature convergence	Prone to local optima, lacks robustness in dynamic environments	[24]
IW-PSO + Binary PSO	Two-level scheduling: IW-PSO for power allocation, Binary PSO for offloading/resource allocation	Minimize transmission energy, improve system efficiency	Jointly reduces energy and enhances response time	Increased complexity due to dual-layer structure	[25]
GA-PSO Hybrid	Genetic Algorithm + PSO	Multi-objective job scheduling	Improves execution time and reduces energy compared to single algorithms	inherits the drawbacks of both GA and PSO	[26]
PSO-based Heuristic	Cost-oriented PSO variant	Reduce energy costs and decision time	Achieves significant performance improvements	limited handling of task dependencies or dynamics	[27]
Cooperative PSO + Orchestrator	Local edge scheduling + orchestrated cloud/Base Station tasks	Task prioritization by throughput and delay	Optimizes resource use in two-tier environments	Lacks fine-grained control of edge tasks	[28]
LPSO	PSO with Lyapunov framework	Stabilize queues, balance energy in IoT/Fog	Reduces energy while ensuring timely task completion	Requires accurate modeling of system dynamics	[29]
Slow-Motion PSO	Mapping particle positions to tasks	Handle NP-hard scheduling, minimize makespan	Converts abstract positions into feasible schedules	not scale with heterogeneous environments	[30]
GA	Classic GA with probabilistic crossover	General task scheduling in MEC/Fog/Cloud	Fast for NP-complete problems, highly adaptable	limited exploitation	[31]
IGA-TSPA	Improved GA with refined initialization and mutation	Speed convergence, improve job matching	Narrows the solution space, faster convergence	Still risks local optima in large task sets	[34]
GA + Auction Model	GA with task queue and bidding	Efficient job-server pairing	Dynamic and fair server assignment	Bidding-based methods may introduce latency	[35]
FOA	Standard Fruit Fly Optimization	Improve workflow scheduling in the cloud	Enhances efficiency and optimization	less efficient in constrained edge environments	[36]

IFFO	Improved FOA minimizes the makespan and cost	Multi-workflow scheduling	Achieves cost and time efficiency	Multiple objectives can reduce	[37]
FSAOS	FOA + Simulated Annealing	Avoid local optima	Enables balanced QoS and cost	Annealing can slow convergence	[38]
FA (Firefly Algorithm)	Tuning the smell coefficient and attractiveness	Solve multimodal problems	Avoids local optima, needs careful tuning		
ACO	Pheromone trail modeling	Workflow scheduling, resource matching	Balances exploration-exploitation, energy efficient	Scalability issues in large	[40]

A review of recent literature reveals substantial advancements in task scheduling strategies for edge computing, particularly through metaheuristic and hybrid optimization methods. Nevertheless, critical limitations persist. Many approaches, such as standard PSO and GA variants, exhibit premature convergence, limited scalability, and inadequate performance in handling dynamic and heterogeneous workloads, especially under real-time constraints. Others, including hybrid and cooperative algorithms, introduce high computational overheads and often lack effective mechanisms. Moreover, several studies fail to balance key objectives such as energy consumption, execution time, and VM utilization simultaneously, frequently sacrificing one to optimize another. The need for manual parameter tuning, inability to adapt to fluctuating edge environments, and poor convergence in multi-objective contexts further hinder practical deployment. These issues underscore the necessity for a more adaptive multi-objective scheduling solution. In response, this study proposes an Enhanced Grey Wolf Optimizer integrated with a Dimension Learning-Based strategy (EGWODLB) that is specifically designed to improve convergence speed, task scheduling accuracy, and optimization across diverse edge environments.

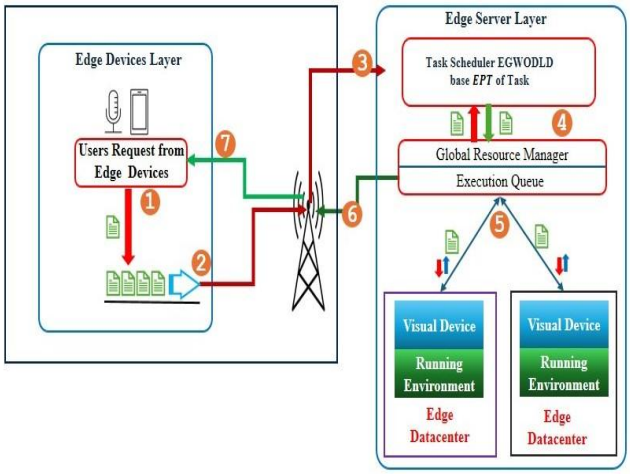

Fig. 1. System model.

Fig. 1 shows a model of the system for a multiuser, multi-server context where computational tasks originate from various edge devices. These tasks are managed by a sophisticated scheduling system designed to allocate resources efficiently. The system calculates the estimated processing time (EPT) for every task running on each VM via the following equation $ETP_{ij}=\frac{\mathrm{LT}_i}{PSV_{(machin)j}}$, where LT_i is the task length in millions of instructions (MI) and $PSV_{(machine)j}$ indicates the processing power of the VM. It leverages a more enhanced strategy known as EGWODLB for task-VM assignments in pursuit of optimality to optimise task assignments. This intelligent strategy can explore the huge solution space for finding the optimal task-to-VM assignments since there can be thousands of ways to configure them. It does the runtime monitoring of QoS metrics like execution time and energy consumption once the optimal task assignments are done to VMs. The system dynamically readjusts the resources and allocations to maintain peak performance under intensive resource utilization. QoS monitoring will offer real-time feedback, which permits adjustment and adaptations to variant conditions in line with the highest level of service quality. It encompasses integrating tasks generated from edge devices, EPTs calculation, optimization by EGWODLB, and task allocation to VMs for efficient task allocation for resource management in a multiuser, multiple-server edge computing environment.

III. Problem Description

The section offers a thorough explanation of the mathematical model that was applied to solve the edge computing environment's task scheduling challenge. that includes multiple users and servers. The proposed model considers EC servers, each with unique specifications such as RAM, storage capacity, CPU cores, and network bandwidth. These servers are capable of scaling to meet different Quality of Service (QoS) requirements. Consider $V_{machine}=\{V_{mashine1,\ldots}\, V_{machine2,\ldots}\, V_{machinev}\}$ as a set of virtual machines within an EC server. Each virtual machine's processing capacity is expressed in milliseconds (M). Let $T=(t_{1,\ldots}\, t_{2,\ldots}\, t_n)$ represent the set of tasks submitted by edge devices for execution on these virtual machines. Given the large number of tasks and virtual machines, the task scheduling problem becomes highly complex, involving a vast search space with numerous possible task-to-VM assignments. The task length LT_i of each task t_i is determined in millions of instructions. The Estimated Processing Time (EPT) is utilized to track the time needed to execute a specific task across different virtual machines.[46] EPT_{ij} refers to the EPT of t_i on $V_{machinej}$ computed as Eq. (1):

$$EPT_{(ij)}=\frac{\mathrm{LT}_i}{V_{machinej}}, 1\le i\le n, 1\le j\le v \qquad (1)$$

where, LT indicates the task's length i and where $V_{machinej}$ denotes the processing power of the virtual machine. We aim to reduce the execution time and energy consumption to increase the quality of service. To address the extensive search space, the method uses an enhanced GWO-integrated DLB strategy called the EGWODLB. This approach leverages dimension learning to explore the large combinatorial space efficiently, thus improving the effectiveness and efficiency of the scheduling process. Accurate task assignments are crucial for reducing execution time and energy consumption. The

execution time (ET) is defined as the maximum total duration required to complete all tasks as in Eq. (2).

$$ET=max_{j\in(1,2,\ldots,v)}\ \sum_{i=1}^{n} x_{ij}\times ET_{ij} \quad (2)$$

Energy consumption is a key factor in the quality of service and includes both the active and idle states of virtual machines. Research indicates that a virtual machine typically consumes more of its total energy while idle. Therefore, the energy consumption $EV_{(machine)j}$ for a specific $V_{machine\,j}$ can be calculated as Eq. (3):

$$E_{vmj}^{total}=E_{j}^{total}\ \times a_j{+}{+}(ET\text{-}ET_{j}^{total})\times Cenjg_j \quad (3)$$

where, E_{vmj}^{total} indicates the total time VM is actively executing the tasks of $Ceng_j$energy consumption rate of VM during the idle state, and a_j indicates the energy consumed in its active state. Consequently, the following equation can be used to determine an edge system's overall energy consumption:

VM Utilization is a performance metric that quantifies the effective computational capacity of a virtual machine (VM) used over a given period. It is defined as the ratio of the time a VM is actively engaged in executing tasks to the total time it is available during a simulation. The overall VM Utilization across all VMs is formulated as Eq. (4):

$$U_j=\frac{\sum_{i=1}^{n} x_{ij}\times EPT_{ij}}{ET} \quad (4)$$

where, $x_{ij}$is a binary decision variable indicating whether the task $t_i$is assigned to a virtual machine j, EPT_{ij} is the estimated processing time of the task t_i on VM j, and ETdenotes the overall execution time (makespan) of the system as defined in Eq. (2).

To evaluate the overall utilization of the edge computing system, the average VM utilization across all virtual machines is computed as Eq. (5):

$$U_{avg}=\frac{1}{m}\sum_{j=1}^{m} U_j\times 100 \quad (5)$$

where, $m$is the total number of virtual machines. Utilization is expressed as a percentage to reflect the efficiency of resource use across the system.

A. Network Model

The network connection model in edge computing is designed to optimize data flow and interaction between edge devices and edge servers [47]. This model plays a critical role in enhancing the performance of real-time applications by ensuring efficient data transmission while balancing local processing capabilities with the extensive resources available at the edge. In this model, edge devices such as smartphones and IoT sensors connect to edge computing through dedicated communication links. This setup enables low-latency data processing, which is essential for meeting the stringent Quality of Service (QoS) requirements of real-time applications. Tasks requiring higher computational power or additional storage are seamlessly transferred from edge devices to one or more edge servers, leveraging their superior processing capabilities. The hierarchical structure of the model ensures efficient task and data flow between these devices and servers, optimizing local resource usage while taking full advantage of Edge Computing (EC) infrastructure [48]. Moreover, interconnections between edge servers allow tasks to be offloaded from one edge server and results retrieved from another. This collaborative mechanism enhances resource utilization, minimizes latency, and upholds QoS standards for a wide range of applications. As a result, this network connection model significantly improves the overall performance, efficiency, and reliability of the edge computing environment. Let $M=(MD_1, MD_2, \ldots, MD_n)$ Consider the collection of edge devices and $S=(MS_1, MS_2, \ldots, MS_P)$ represent the set of edge servers in the environment, the following guidelines are used to calculate the network model. Every edge device needs to be linked to a single edge server [Eq. (6)].

$$\sum_{J\in M} kij=1\ \forall MD_i\in M \quad (6)$$

where, k_{ij} is a binary variable that indicates whether the edge server MS_j is linked to an edge device MD_i (1 if true, 0 otherwise). Multiple edge servers can be linked to each edge device [Eq. (7)].

$$\sum_{J\in S} hij=0\ \forall MS_i\in S \quad (7)$$

where, the binary variable h_{ij} indicates if the edge server MS_i and order the edge server MS_j are connected (1 if true, 0 otherwise). From the source to the destination, the data flow needs to be preserved [Eq. (8)].

$$\sum_{z:(z,i)\in L} f_{zi}\text{-}\sum_{z:(z,i)\in L} f_{ij}=q_i 0,\forall_i\in M\cup S \quad (8)$$

where, L is the set of network links, z is an index for servers from which data is flowing into the server i, q_i is the flow demand at the server i, and f_{ij} is the data flow from server i to server j.

B. Fitness Function

The Enhanced Grey Wolf Optimization (EGWO) is a metaheuristic algorithm that schedules tasks based on a fitness function to identify the most suitable computing resources. The fitness function plays a crucial role in evaluating solutions by aligning them with the objectives of minimizing energy consumption and execution time. In this study, the fitness function is specifically designed to prioritize solutions that achieve these goals effectively. The performance of the proposed algorithm is significantly influenced by the value of the fitness function. In the context of Multi-objective Optimization (MOP), it is essential to address various conflicting objectives simultaneously. However, the fitness function cannot directly compare the generated solutions against each other, highlighting the complexity of achieving a balanced optimization [Eq. (9)].

$$F(Y)=(1\text{-}a)\ \times ET{+}a\ \times E_{vmj}^{total} \quad (9)$$

where, ETrepresents the overall execution time (makespan), E_{vm}^{total} denotes the total energy consumption of all virtual machines, and $\alpha\in[0,1]$is a weighting parameter that controls the trade-off between the two objectives.

The parameter α plays a critical role in balancing the optimization priorities. When α approaches 0, the model

prioritizes minimizing execution time, which is suitable for latency-sensitive applications. Conversely, when αapproaches 1, the model emphasizes energy efficiency, making it more appropriate for energy-constrained environments. Intermediate values of αprovide a balanced trade-off between performance and energy consumption.

To evaluate the robustness of the proposed model, a sensitivity analysis was conducted by varying αacross different values within the interval $[0, 1]$. The results indicate that moderate values of α(e.g., $\alpha = 0.4$ to 0.6) achieve a good balance between execution time and energy consumption, while extreme values tend to bias the optimization toward a single objective. Based on this observation, a balanced value of $\alpha$is selected in this study to ensure effective multi-objective optimization.

IV. Proposed Approach

The enhanced grey wolf optimizer (EGWODLB) enhances the original GWO algorithm and integrates a dimension learning-based hunting (DLB) strategy to achieve numerous goals. This represents the first application of the EGWODLB method in the context of edge computing. GWO is inspired by the natural social hierarchy and hunting behavior of grey wolves. Starting locations for the remaining ω wolves are then nudged toward positions corresponding to the best three fitness values to locate the global optimum. The three top wolves are the best candidates in the GWO technique, known as α, β, and δ (Fig. 2). Wolf hunting consists of three major stages: circling, chasing, and attacking the prey.

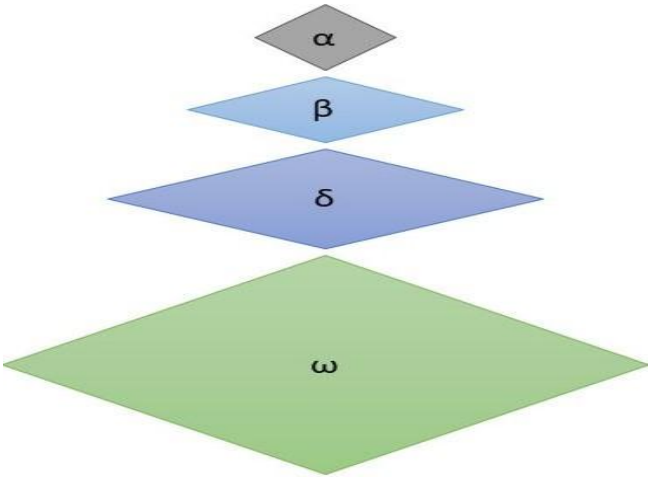

Fig. 2. Grey wolf hierarchy (dominance diminishes from the top down).

Encircling: Provide a mathematical representation of the actions of grey wolves as they circle their prey via Eq. (10) and Eq. (11).

$$D_{\vec{()}}=C_{\text{vec}} \cdot M_p(t)\text{-}M(t) \quad (10)$$

$$M(t+1)= M_p(t)\text{-}D_{(\text{vec})} \cdot A \quad (11)$$

where, $D_{\vec{()}}$ represents the computed distance between the grey wolf and its prey, whereas $C_{\vec{()}}$ is the coefficient vector. In this model, $m(t)$ denotes the position of the grey wolf at a specific time t, and M_p represents the prey's location at the same time. The coefficient vectors C and A are determined via Eq. (12) and Eq. (13):

$$A=2 \cdot a \cdot r_1\text{-}a \quad (12)$$

$$C_{\vec{()}}=2 \cdot (rand\ 0,1) \quad (13)$$

Here, r_1 and r_2 are random vectors that appear in the range of [0, 1], and according to Eq. (14), the components of the vector decrease linearly from 2 to 0 during the repetitions.

$$a=2\text{-}(2 \cdot \frac{t}{maxiteration}) \quad (14)$$

where, t indicates the iteration that is currently in use, whereas t_{max} represents the maximum number of iterations.

Hunting: To describe wolves' hunting behavior quantitatively, α, β, and δ require greater knowledge of the prey's position. As a result, the other wolves (ω) are driven to follow the three best solutions (α, β, and δ). Eq. (15)-(16) characterize the hunting behavior as follows:

$$\begin{aligned} D_\alpha&=acs\big(C_1 \cdot Y_\alpha(t) \cdot M(t)\big) \\ D_\beta&=acs(C_2 \cdot Y_\beta(t) \cdot M(t)) \\ D_\delta&=acs(C_3 \cdot Y_\delta(t) \cdot M(t)) \end{aligned} \quad (15)$$

$$\begin{aligned} Y_1(t)&=Y_\alpha(t)\text{-}A_1 \cdot D_\alpha(t), \\ Y_2(t)&=Y_\beta(t)\text{-}A_2 \cdot D_\beta(t), \\ Y_3(t)&=Y_\delta(t)\text{-}A_3 \cdot D_\delta(t) \end{aligned} \quad (16)$$

where, $Y_{\alpha,} Y_{\beta,} \wedge Y_\delta$ represents the top three solutions at iteration t. The coefficients $A_{1,} A_{2,} A_3$ are computed via Eq. (12), and $D_\alpha, D_\beta, D_\delta$ are defined via Eq. (15).

$$Y(t+1)=\frac{Y_1(t)+Y_2(t)+Y_3(t)}{3} \quad (17)$$

Attacking: the hunting phase ends when the wolves stop moving, and the wolves attack the prey. The whole procedure is controlled through a parameter$a$, which is linearly decreased from iteration to iteration to balance the exploration and exploitation. As can be seen from Eq. (14), a is updated in each iteration, decreasing from the value 2 to 0 according to [49]. As pointed out, during the first iterations the algorithm emphasizes exploration, while the latter ones belong to the exploitation. During this step, wolves move randomly to a position somewhere between their current position and the position of prey.

A. Enhanced Grey Wolf Optimizer (EGWO)

The leading wolves α, β, and δ in the grey wolf optimizer (GWO) algorithm direct the remaining wolves (ω) to investigate advantageous areas of the search space to identify the optimal solution. This strategy, however, may occasionally cause the algorithm to become trapped in local optima, reducing population diversity and ultimately causing premature convergence. We introduce an enhanced grey wolf optimizer (EGWO) to address these problems. This enhanced version incorporates a new search strategy, along with revised selection and updating processes. The EGWO algorithm progresses through three key stages: initialization, movement, selection, and updating, which are explained.

Initialization stage**:** N wolves are distributed at random over the search space in this phase, within the given range [li, uj], via Eq. (18).

$$Y_{ij}=L_j+rand_j(0,1)\cdot\left(U_j\text{-}L_j\right),i\in(1,N),j\in(1,D) \quad (18)$$

The position of the i-th wolf in the t-th iteration is represented as a vector of real values, $Y_i(t)$ = { Y_{i1}, Y_{i2}, …, Y_{iD}), where D is the problem's dimensionality. The entire wolf population is organized in a matrix, Population, with N rows and D columns. The fitness value of $Y_i(t)$ is determined via the fitness function $f(Y_i(t))$.

Movement stage: In addition to individual and group hunting, another important habit observed in grey wolves has inspired further improvements to the GWO algorithm. The enhanced grey wolf optimizer (E-GWO) incorporates a novel movement technique called the dimension learning-based hunting (DLB) search strategy. This approach allows each wolf to learn from its peers to discover a new potential position Yi(t). Below is a comparison of how the traditional GWO and DLB search strategies generate different candidate positions.

Canonical GWO Search Strategy: Using the standard grey wolf optimizer (GWO) algorithm, the top three wolves in the population are identified and given the roles of $\alpha, \beta,$ and δ. Eq. (12)-(14) are used to compute the linearly decreasing coefficient a, as well as the coefficients A and C_{vec}. To determine how to encircle the prey, the positions Y_α, Y_β, and Y_δ are enumerated via Eq. (15) and (16). Finally, the initial candidate for the new position of wolf $Y_i(t)$ is computed via Eq. (17).

Dimension Learning-Based (DLB) Search Strategy**:** In the traditional GWO algorithm, the three leader wolves guide each wolf's new position. However, this approach can result in slow convergence, reduced population diversity, and local optima entrapment. The DLB strategy aims to address these issues by incorporating additional learning mechanisms. To address these issues, the proposed DLB search strategy incorporates individual hunting, where wolves learn from their neighbors. In the DLB search strategy, each component of the new position for wolf $Y_i(t)$ is calculated via Eq. (12). This strategy allows the wolf to learn from its various neighbors and a randomly selected wolf from the population. In addition to the candidate position $Y_{(i\text{-}gwo)}(t+1)$, the DLB search strategy produces another potential candidate for the wolf's new position, labeled $Y_{(i\text{-}dlb)}(t+1)$. The process begins by determining a radius $RS_i(t)$, which is calculated on the basis of the Euclidean distance between the current position of $Y_i(t)$ and the candidate position $Y_{(i\text{-}gwo)}(t+1)$, as specified in Eq. (19).

$$RS_i(t)=\left\|Y_i(t)\text{-}Y_{(i\text{-}gwo)}(t+1)\right\| \quad (19)$$

Next, the neighbors of $Y_i(t)$, denoted as $N_i(t)$, are determined via Eq. (20) on the basis of the radius $RS_i(t)$, where D_i represents the Euclidean distance between $Y_i(t) \wedge Y_j(t)$.

$$N_i(t)=\left\{Y_i(t)|D_i(Y_i(t),Y_j(t)\le RS_i(t),Y_j(t)\in population\right\} \quad (20)$$

After the neighborhood of $Y_i(t)$ is constructed, multineighbor learning is carried out via Eq. (20). In this process, the d-dimension of $Y_{(i\text{-}dlb)}(t+1)$ is computed via the d-th dimension of a randomly chosen neighbor $Y_{n,k}(t)$ from $N_i(t)$ and a randomly selected wolf $Y_{r,k}(t)$ from the population [Eq. (21)].

$$Y_{(i\text{-}dlb),k}(t+1)=Y_{(ik)}(t+1)+rand(0,1)\cdot(Y_{(n,k)}(t+1)\text{-}Y_{(rk)}(t+1)) \quad (21)$$

Selection and updating stage: During this stage, the fitness values of the two candidates are compared to determine whether one has a higher value, $Y_{(i\text{-}gwo)}(t+1)$ and $Y_{(i\text{-}dlb)}(t+1)$, via Eq. (22).

$$Y_i(t+1)=\begin{cases}Y_{(i\text{-}dlb)}(t+1), if\ f\left(Y_{(i\text{-}gwo)}\right)<f\left(Y_{i\text{-}dlb)}\right)\\ ortherwi\\ Y_{(i\text{-}dlb)}(t+1)\end{cases} \quad (22)$$

Next, to update the position of $Y_i(t+1)$, if the fitness value of the chosen candidate is better than that of $Y_i(t+1)$, it is updated with the selected candidate. If not, $Y_i(t)$ remains unchanged in the population. Finally, after this process is completed, the search continues until the predetermined maximum number of iterations (Maxiter) is achieved, at which point the iteration counter (iter) is increased by one for everyone. The EGWODLB algorithm is shown in Algorithm 1

Algorithm 1: EGWODLB

Input: N, D, $Maxiter$
Output: $Y_{best} \wedge F_{best}$
Begin
1. Initializing population of N solution i=(1,2, …, N) using Eq. (14)
2. Calculate the fitness value $F(Y)$ for each wolf using the objective function using Eq. (8)
3. Identify the top three wolf α , β, $\wedge$ δ on their fitness value
4. **For** $iter$ = 2 to $Maxiter$ **do**
5. **For** each wolf in in the population **do**
6. Grey wolf optimizer strategy
7. Updated the position $Y(t+1)$ by following $\alpha, \beta \wedge \delta$ using Eq. (14), to (16)
8. Dimension learning Strategies
9. Generate and alternative candidate position $Y_{(i\text{-}dlb)}(t+1)$ using Eq. (20)
10. Selection and updating
11. *Calculate the fitness* $F(Y(t+1)$ *and* $F(Y_{(i\text{-}dlb)}(t+1)$
12. **If** $F(Y_{(i\text{-}dlb)}(t+1)<F(Y(t+1))$ **then**
13. Update $Y(t+1)=Y_{(i\text{-}dlb)}(t+1)$
14. **End if**
15. **End for**
16. Update $\alpha, \beta, \wedge \delta$ if new better solution is found
17. **End for**
18. Set $Y_{(best)}$= the best solution found in the population
19. Set $F_{(best)}$= fitness value
24. **Return** $Y_{(best)}$ and $F_{(best)}$
25. **END**

The computational complexity of the proposed algorithm is $O(N \times MaxIter \times n)$, where $n$is the number of tasks.

V. Experimental Setup

The experiments were performed using the same simulation settings as outlined in the baseline study. In the experiment scenario, several edge devices generate different applications. Every application consists of several tasks that are required to be executed by edge resources. The proposed method considers ten EC servers, each connected with several edge devices. In the simulated edge environment, the computing capability of the server VM is set to 100 and 1000MIPS. Every edge server can have one or more processors, and every task has a random length ranging from 10 to 510 MIPS. The simulations are based on four tasks with total workloads of 40, 80, 120, and 160, and the maximum iterations are 500, respectively. To make the length of each task have inherent unpredictability, the tasks are generated randomly. For the hardware setup, the experiments were conducted on a high-performance system equipped with an Intel Core i7 processor (2.80 GHz, 8 cores), 8 GB of RAM, and an NVIDIA GTX 1080 GPU, providing the necessary resources for parallel processing. Edge Computing (EC) servers and edge devices were emulated using virtual machines configured to reflect typical edge processor specifications. The software environment was based on Ubuntu 20.04 LTS, utilizing Python 3.12 and essential libraries for optimization and performance evaluation. Both synthetic and real-world datasets were employed to test the algorithm under various EC load conditions. The synthetic datasets generated task requests with diverse parameters, such as task size and computational requirements. Evaluation scenarios included varying network bandwidth, latency, and EC server loads to simulate various operational environments. The parameters used for the simulation are presented in Table II.

TABLE II. Parameters and Their Descriptions

Parameters	Description
N	Number of wolves in the population during optimization.
D	Number of dimensions in the solution space.
VM	Set of VMs available for task execution.
$TL_{i,j}$	Length of task i, measured in millions of instructions (MI).
ET	Maximum duration required to complete all tasks
E_{total}	Total energy consumed by VM j during task execution.
EPT_{ij}	Estimated processing time for task i on VM j.
FY	Fitness function.
L	Set of network links between edge devices and servers.
$r_1 \wedge r_2$	Random values used in GWO to ensure diversity in the population.
Y_i	The current position of the i-th wolf in the solution space.
$PSV_{machine}$	Processing capability of the j-th virtual machine (VM).

Table III outlines the key simulation parameters employed to evaluate the performance of the proposed EGWODLB algorithm within a multi-user, multi-server edge computing environment. The experimental setup includes 10 heterogeneous edge servers with processing capacities ranging from 1000 to 2800 MIPS, and tasks with lengths varying from 2000 to 5600 million instructions (MI). Task data sizes span 400 to 600 MB, and the total number of tasks ranges from 40 to 160 to reflect dynamic workload scenarios. The system bandwidth is fixed at 1 Gbps to simulate realistic network constraints. Additionally, a population size of 50 is adopted for the optimization process, and a weight coefficient parameter a∈[0,1] is used to balance multiple objectives. These configurations are designed to simulate diverse edge conditions and workloads, enabling a comprehensive assessment of the EGWODLB algorithm execution efficiency and energy consumption performance under varying computational demands.

TABLE III. Simulation Parameters

Parameters	Values
No of Server	10
CPU Power	1000-2800MS
Bandwidth capacity	1Gbs
Task length	2000-5600MI
Data Size	400-600mb
No of Task	[40,80,120,160]
a	[0,1]
Population size	50

VI. Result and Discussion

This section comprehensively analyses the findings obtained from the experimental evaluations, comparing the proposed EGWODLB algorithm against the benchmark algorithm using the evaluation metrics, which are presented in the following subsections

A. Execution Time Result Analysis

Execution time remains a critical performance indicator in task scheduling for edge computing environments, particularly in latency-sensitive applications. Fig. 3 and Table IV present a comparative evaluation of the proposed Enhanced Grey Wolf Optimization Integrated Dimension Learning-Based (EGWODLB) strategy against the benchmarks QPSO and EADFPSO algorithm across varying workloads of 40, 80, 120, and 160 tasks. The experimental findings demonstrate that the EGWODLB algorithm consistently outperforms EADFPSO by achieving significantly lower execution times across all task scenarios. For instance, under the highest workload of 160 tasks, EGWODLB recorded an execution time of 39.40 seconds, whereas QPSO and EADFPSO required 41.19 and 40.29 seconds. Although this margin may appear modest in absolute terms, it represents the execution time by average 4.71% reduction. More notably, the EGWODLB algorithm demonstrated robust stability in execution time as task load increased. While EADFPSO exhibited noticeable time degradation beyond 120 tasks, EGWODLB sustained relatively smoother performance. This is attributed to the integration of the Dimension Learning-Based (DLB) strategy, which enhances the search diversity of the population and mitigates the risk of premature convergence. By learning from neighboring solutions and generating diversified candidates,

the algorithm ensures that task-to-VM mappings are optimized even under high computational stress.

TABLE IV. EXECUTION TIME BASED ON INCREASING NUMBER OF TASKS

Number of Tasks	QPSO Execution time(S)	EADFPSO Execution time(S)	EGWODLB Execution time(S)	Improvement (%)
40	20.98	20.91	20.11	3.82
80	29.66	28.81	27.20	5.58
120	33.56	32.94	30.55	7.25
160	41.19	40.29	39.40	2.20

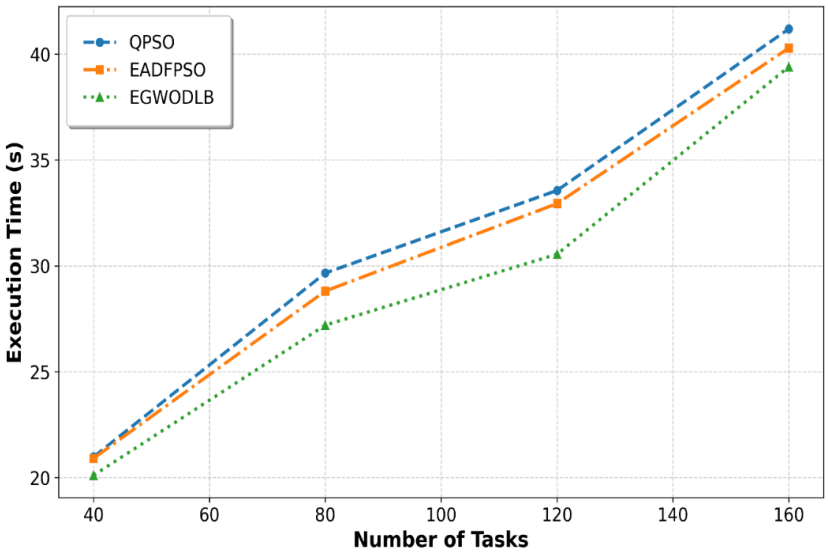

Fig. 3. Execution time based on increasing number of tasks.

As shown in Fig. 3 and Table IV, it proves that EGWODLB generally performs the lowest execution time throughout the various tasks evaluated. For example, at a workload of 160 tasks, the execution time recorded by EGWODLB was 39.40, while QPSO and EADPSO had execution times of 41.19 and 40.29, respectively. Therefore, this large execution time reduction logically means that the algorithm is better in terms of capability regarding the scheduling of tasks, even when it comes to increasing the workloads.

B. Energy Consumption Result Analysis

Energy consumption is a crucial metric in the evaluation of task scheduling algorithms for edge computing, as it directly affects the overall QoS and the environmental sustainability of edge computing systems. In resource-constrained edge environments where power availability is often limited, minimizing energy consumption while maintaining responsiveness is a complex yet essential objective. Table V and Fig. 4 illustrate the comparative energy consumption performance of the proposed EGWODLB algorithm and the benchmarks QPSO and EADFPSO across various task loads, 40, 80, 120, and 160 tasks. The experimental results clearly show that EGWODLB achieves consistently lower energy consumption than benchmarks under all evaluated scenarios. This is enabled by its integration of the Dimension Learning-Based (DLB) strategy, which enhances the task scheduling process by promoting more energy-efficient task-to-VM assignments. The strategy reduces redundant computations, thereby limiting energy waste due to underutilized and overburdened virtual machines. Most importantly, the total cumulative energy consumption achieved by EGWODLB across all workloads is 12.09%, which represents a significant reduction in overall energy consumption compared to the benchmark method.

TABLE V. ENERGY CONSUMPTION WITH DIFFERENT NUMBER OF TASKS

Number of Tasks	QPSO Energy (J)	EADFPSO Energy(J)	EGWODLB Energy(J)	Improvement (%)
40	3.44	3.26	2.80	14.11
80	4.60	4.58	3.99	12.88
120	4.97	4.89	4.20	14.11
160	5.77	5.50	5.10	7.27

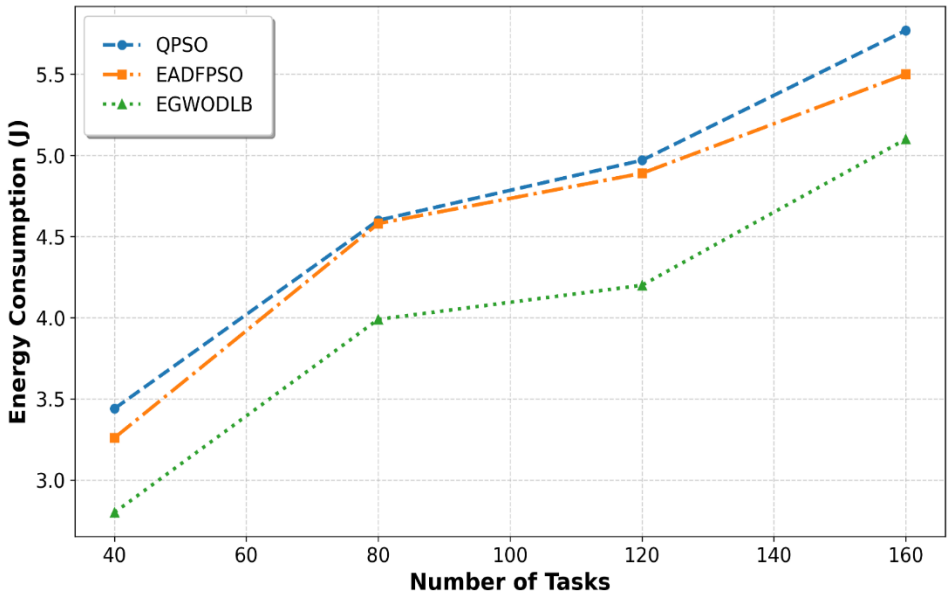

Fig. 4. Energy consumption with different tasks.

As shown in Fig. 4 and Table IV, the energy consumption performance of the algorithms under varying workloads. It is obvious that the EGWODLB algorithm always has the lowest energy consumption in all scenarios. When faced with a maximum workload of 160 tasks, EGWODLB consumes 5.10 energy consumption (J), significantly less than QPSO and EADFPSO, which consume 5.77 and 5.50 energy consumption (J), respectively. The result indicates a significant decrease in energy consumption, demonstrating the efficiency of the proposed EGWODLB in reducing energy consumption in the edge computing environment.

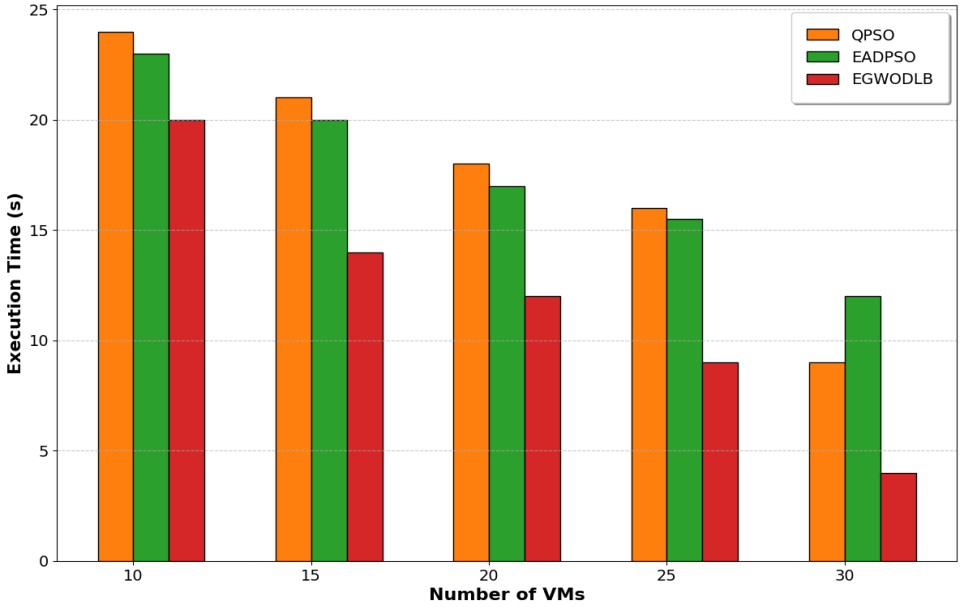

Fig. 5. VMs against execution time.

Fig. 5 illustrates the performance of the proposed EGWODLB scheduling algorithm. The evaluation involved executing a fixed set of 40 tasks across a varying number of Virtual Machines (VMs), which were incrementally increased from 10 to 30. The execution time was measured for each scenario. As depicted in Fig. 5, the execution time decreased significantly as the number of VMs increased, regardless of the algorithm used. This improvement is attributed to the enhanced parallel processing capabilities provided by additional VM resources in the edge environment, which directly reduces task completion times. The results highlight that the proposed EGWODLB algorithm consistently achieved the shortest execution times, outperforming traditional methods. For instance, with 30 VMs, the EGWODLB algorithm completed all 40 tasks in just 4.22 seconds, while the benchmark algorithms required 12.35 seconds, more than double the time.

This superior performance is attributed to the iterative optimization of scheduling solutions using the EGWODLB method.

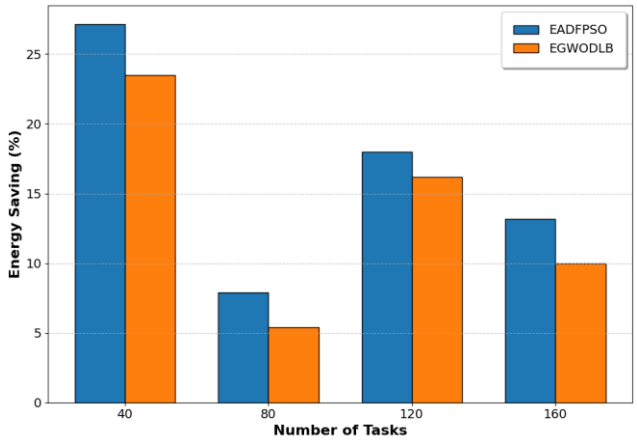

Fig. 6. Energy saving for EGWODLB with EADFPSO.

As shown in Fig. 6, the analysis evaluates task groups ranging from 40 to 160 tasks. The results reveal that the EGWODLB algorithm achieves substantial energy savings compared to the EADFPSO algorithm: 23.5% for 40 tasks, 5.4% for 80 tasks, 16.2% for 120 tasks, and 10.7% for 160 tasks. These findings highlight the superior energy efficiency of the EGWODLB algorithm across varying task loads.

C. Virtual Machine (VM) Utilization Results

Virtual Machine (VM) Utilization quantifies the efficiency of task allocation by measuring the percentage of computational resources actively engaged in task execution. It serves as a critical metric for evaluating how well an algorithm distributes workload. Higher VM utilization reflects better exploitation of system resources, reduced idle time, and more effective parallel processing. VM utilization is analyzed for both the EADFPSO and the proposed EGWODLB algorithms across varying task sizes. As shown in Table VI and Fig. 7, the EGWODLB algorithm consistently demonstrates superior VM utilization, indicating a more balanced and optimized task allocation strategy. The improvement in utilization, although modest, indicates that EGWODLB executes tasks with better efficiency, reflecting enhanced workload.

TABLE VI. SUMMARY OF VM UTILIZATION FOR BOTH EADFPSO AND EGWODLB

Number of Tasks	EADFPSO Execution Time (s)	EGWODLB Execution Time (s)	EADFPSO VM Utilization (%)	EGWODLB Utilization VM (%)
40	20.34	17.11	39.33	46.76
80	29.42	25.01	54.38	63.97
120	30.15	27.58	79.60	87.02
160	40.58	34.98	78.86	91.48

The results in Fig. 7 provide a clear comparative analysis of VM utilization achieved by the EADFPSO and EGWODLB algorithms across increasing workload sizes (40 to 160 tasks). The EGWODLB algorithm demonstrates significantly higher VM utilization than the baseline EADFPSO in all scenarios. For instance, with a task load of 40, EGWODLB achieves a VM utilization of 46.76%, compared to 39.33% for EADFPSO. The efficiency gap widens as the task load increases. At 160 tasks, EGWODLB reaches a utilization level of 91.48%, while EADFPSO achieves only 78.86%. This performance difference is attributed to EGWODLB's superior task distribution strategy, which minimizes task queuing delays and ensures that more VMs remain actively engaged throughout the simulation period. By reducing idle time across the VM. Underutilized VMs not only waste energy but also lead to suboptimal task execution times. Therefore, the ability of EGWODLB to maintain utilization rates of 91.48% under peak loads makes it particularly suitable for real-time and large-scale edge-based applications.

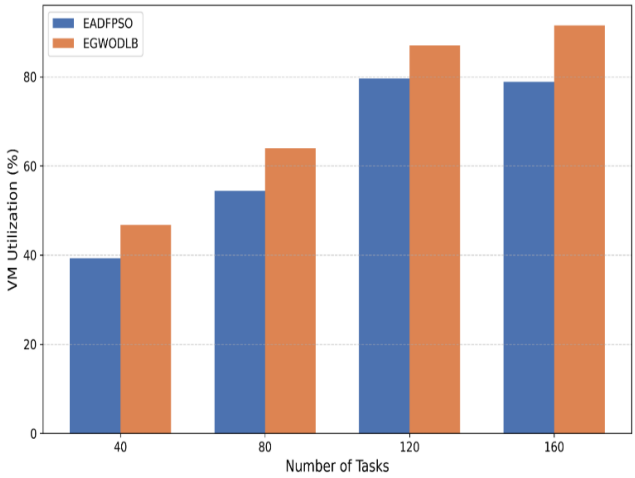

Fig. 7. VM utilization for EADFPSO and EGWODLB.

VII. CONCLUSION

This work proposed an advanced EGWODLB approach to scheduling tasks in EC. The algorithm, in this work, outperforms the benchmark algorithms in execution time and energy consumption under various kinds of workloads. By incorporating the dimension learning-based hunting strategy, the algorithm will enormously improve its exploration and exploitation capability in the search space, leading to a more efficient scheduling of tasks. These results emphasize that, because of its notable features in execution time, energy consumption, VM utilization, and running time, EGWODLB can function as a highly effective mechanism for performance optimization concerning the edge computing model. To better reflect realistic edge computing scenarios, the experimental setup is extended to include larger and more diverse workloads. with a proportional increase in virtual machines to simulate heterogeneous environments. This allows evaluation of scalability in terms of execution time, energy consumption, and resource utilization. The results show that the proposed EGWODLB approach maintains stable performance and scales effectively with increasing workload size, demonstrating its suitability for dynamic real-world edge computing applications.

Feature work studies may be presented on how the EGWODLB algorithm may be utilized in dynamic and

heterogeneous edge computing scenarios, and/or the need to incorporate with machine learning to improve its adaptability and scalability.

AUTHORS' CONTRIBUTION

Jafar Aminu: Data analysis, methodology development, manuscript writing; Rohaya Latip: Conceptualization, supervision, manuscript reviewing; Zurina Mohd Hanafi: supervision, manuscript editing; Shafinah Kamarudin: supervision, critical revisions; Mustapha Abubakar Giro: manuscript reviewing and editing.

CONFLICT OF INTEREST

There is no conflict of interest. All authors have agreed to its submission.

ACKNOWLEDGMENT

This study is supported by the Universiti Putra Malaysia and the Ministry of Higher Education Malaysia under grant Number: (FRGS/1/2023/ICT11/UPM/02/3). We are sincerely grateful for the facilities and funding provided, which were essential for the completion and publication of this study.

REFERENCES

[1] X. Qiao, P. Ren, S. Dustdar, L. Liu, H. Ma, and J. Chen, "Web AR: A Promising Future for Mobile Augmented Reality-State of the Art, Challenges, and Insights," Proc. IEEE, vol. 107, no. 4, pp. 651–666, 2019, doi: 10.1109/JPROC.2019.2895105.

[2] J. Aminu, R. Latip, Z. M. Hanafi, S. Kamarudin, and D. Gabi, Systematic review of metaheuristic-based task scheduling strategies in edge computing environments. 2025.

[3] J. Qadir, B. Sainz-De-Abajo, A. Khan, B. García-Zapirain, I. De La Torre-Díez, and H. Mahmood, "Towards Mobile Edge Computing: Taxonomy, Challenges, Applications and Future Realms," IEEE Access, vol. 8, pp. 189129–189162, 2020, doi: 10.1109/ACCESS.2020.3026938.

[4] L. T. Hsieh, H. Liu, Y. Guo, and R. Gazda, "Deep Reinforcement Learning-Based Task Assignment for Cooperative Mobile Edge Computing," IEEE Trans. Mob. Comput., vol. 23, no. 4, pp. 3156–3171, 2024, doi: 10.1109/TMC.2023.3270242.

[5] X. Zhang, Z. Cao, and W. Dong, "Overview of Edge Computing in the Agricultural Internet of Things: Key Technologies, Applications, Challenges," IEEE Access, vol. 8, pp. 141748–141761, 2020, doi: 10.1109/ACCESS.2020.3013005.

[6] M. Asim, Y. Wang, K. Wang, and P. Q. Huang, "A Review on Computational Intelligence Techniques in Cloud and Edge Computing," IEEE Trans. Emerg. Top. Comput. Intell., vol. 4, no. 6, pp. 742–763, 2020, doi: 10.1109/TETCI.2020.3007905.

[7] L. Chen, K. Guo, G. Fan, C. Wang, and S. Song, "Resource Constrained Profit Optimization Method for Task Scheduling in Edge Cloud," IEEE Access, vol. 8, pp. 118638–118652, 2020, doi: 10.1109/ACCESS.2020.3000985.

[8] P. Varshney and Y. Simmhan, "Characterizing application scheduling on edge, fog, and cloud computing resources," Softw. - Pract. Exp., vol. 50, no. 5, pp. 558–595, 2020, doi: 10.1002/spe.2699.

[9] L. Muwafaq, N. K. Noordin, M. Othman, A. Ismail, and F. Hashim, "A Survey on Cloudlet Computation Optimization in the Mobile Edge Computing Environment," Int. J. Adv. Comput. Sci. Appl., vol. 14, no. 1, pp. 520–532, 2023, doi: 10.14569/IJACSA.2023.0140157.

[10] J. Aminu, R. Latip, Z. M. Hanafi, S. Kamarudin, and D. Gabi, "Efficient Task Allocation for Energy and Execution Time Trade-Off in Edge Computing Using Multi-Objective IPSO," 2025, doi: 10.32604/cmc.2025.062451.

[11] M. Akbari, H. Rashidi, and S. H. Alizadeh, "An enhanced genetic algorithm with new operators for task scheduling in heterogeneous computing systems," Eng. Appl. Artif. Intell., vol. 61, no. March, pp. 35–46, 2017, doi: 10.1016/j.engappai.2017.02.013.

[12] Z. Deng, Z. Yan, H. Huang, and H. Shen, "Energy-aware task scheduling on heterogeneous computing systems with time constraint," IEEE Access, vol. 8, pp. 23936–23950, 2020, doi: 10.1109/ACCESS.2020.2970166.

[13] R. Latip, J. Aminu, Z. Mohd, H. Shafinah, and K. Danlami, "Metaheuristic task offloading approaches for minimization of energy consumption on edge computing: a systematic review," Discov. Internet Things, 2024, doi: 10.1007/s43926-024-00089-y.

[14] P. Paknejad, R. Khorsand, and M. Ramezanpour, "Chaotic improved PICEA-g-based multi-objective optimization for workflow scheduling in cloud environment," Futur. Gener. Comput. Syst., vol. 117, pp. 12–28, 2021, doi: 10.1016/j.future.2020.11.002.

[15] H. Faris, I. Aljarah, M. A. Al-Betar, and S. Mirjalili, "Grey wolf optimizer: a review of recent variants and applications," Neural Comput. Appl., vol. 30, no. 2, pp. 413–435, 2018, doi: 10.1007/s00521-017-3272-5.

[16] D. Alsadie, "TSMGWo: Optimizing task schedule using multi-objectives grey Wolf optimizer for cloud data centers," IEEE Access, vol. 9, pp. 37707–37725, 2021, doi: 10.1109/ACCESS.2021.3063723.

[17] Y. Yang, B. Yang, S. Wang, T. Jin, and S. Li, "An enhanced multi-objective grey wolf optimizer for service composition in cloud manufacturing," Appl. Soft Comput. J., vol. 87, 2020, doi: 10.1016/j.asoc.2019.106003.

[18] G. Huang, Y. Qi, Y. Cai, Y. Luo, and H. Huang, "A Grey Wolf Optimizer Algorithm for Multi-Objective Cumulative Capacitated Vehicle Routing Problem Considering Operation Time," Biomimetics, vol. 9, no. 6, 2024, doi: 10.3390/biomimetics9060331.

[19] J. Zhang, Z. Ning, M. Waqas, H. Alasmary, S. Tu, and S. Chen, "Hybrid Edge-Cloud Collaborator Resource Scheduling Approach Based on Deep Reinforcement Learning and Multiobjective Optimization," IEEE Trans. Comput., vol. 73, no. 1, pp. 192–205, 2024, doi: 10.1109/TC.2023.3326977.

[20] S. Wang, B. Luo, W. Shi, and D. Tiwari, "Application configuration selection for energy-efficient execution on multicore systems," J. Parallel Distrib. Comput., vol. 87, pp. 43–54, 2016, doi: 10.1016/j.jpdc.2015.09.003.

[21] M. E. C. Networks, "A Survey of Energy Optimization Approaches for Computational Task Offloading and Resource Allocation in," 2023.

[22] A. Mijuskovic, A. Chiumento, R. Bemthuis, A. Aldea, and P. Havinga, "Resource management techniques for cloud/fog and edge computing: An evaluation framework and classification," Sensors, vol. 21, no. 5, pp. 1–23, 2021, doi: 10.3390/s21051832.

[23] S. U. Jamil, M. A. Khan, and S. U. Rehman, "Resource Allocation and Task Off-Loading for 6G Enabled Smart Edge Environments," IEEE Access, vol. 10, no. August, pp. 93542–93563, 2022, doi: 10.1109/ACCESS.2022.3203711.

[24] S. Nabi, M. Ahmad, M. Ibrahim, and H. Hamam, "AdPSO: Adaptive PSO-Based Task Scheduling Approach for Cloud Computing," Sensors, vol. 22, no. 3, pp. 1–22, 2022, doi: 10.3390/s22030920.

[25] J. Liu, C. Li, and Y. Luo, "Efficient resource allocation for IoT applications in mobile edge computing via dynamic request scheduling optimization," Expert Syst. Appl., vol. 255, no. PC, p. 124716, 2024, doi: 10.1016/j.eswa.2024.124716.

[26] M. Saad, R. N. Enam, and R. Qureshi, "Optimizing multi-objective task scheduling in fog computing with GA-PSO algorithm for big data application," Front. Big Data, vol. 7, 2024, doi: 10.3389/fdata.2024.1358486.

[27] W. Zhang, H. Xie, B. Cao, and A. M. K. Cheng, "Energy-Aware Real-Time Task Scheduling for Heterogeneous Multiprocessors with Particle Swarm Optimization Algorithm," Math. Probl. Eng., vol. 2014, 2014, doi: 10.1155/2014/287475.

[28] A. A. Amer, I. E. Talkhan, R. Ahmed, and T. Ismail, "An Optimized Collaborative Scheduling Algorithm for Prioritized Tasks with Shared Resources in Mobile Edge and Cloud Computing Systems," Mob. Networks Appl., pp. 1444–1460, 2022, doi: 10.1007/s11036-022-01974-y.

[29] C. Huang, H. Wang, L. Zeng, and T. Li, "Resource Scheduling and Energy Consumption Optimization Based on Lyapunov Optimization in Fog Computing," Sensors, vol. 22, no. 9, 2022, doi: 10.3390/s22093527.

[30] Y. Zhang, Y. Liu, J. Zhou, J. Sun, and K. Li, "Slow-movement particle swarm optimization algorithms for scheduling security-critical tasks in resource-limited mobile edge computing," Futur. Gener. Comput. Syst., vol. 112, pp. 148–161, 2020, doi: 10.1016/j.future.2020.05.025.

[31] S. Chakraborty and K. Mazumdar, "Sustainable task offloading decision using genetic algorithm in sensor mobile edge computing," J. King Saud Univ. - Comput. Inf. Sci., vol. 34, no. 4, pp. 1552–1568, 2022, doi: 10.1016/j.jksuci.2022.02.014.

[32] Z. Jalali Khalil Abadi and N. Mansouri, A comprehensive survey on scheduling algorithms using fuzzy systems in distributed environments, vol. 57, no. 1. Springer Netherlands, 2024. doi: 10.1007/s10462-023-10632-y.

[33] S. Azizi, M. Shojafar, J. Abawajy, and R. Buyya, "Deadline-aware and energy-efficient IoT task scheduling in fog computing systems: A semi-greedy approach," J. Netw. Comput. Appl., vol. 201, no. January, 2022, doi: 10.1016/j.jnca.2022.103333.

[34] Y. Chen, X. Bai, X. Jin, Z. Wang, F. Wang, and L. Ling, "Edge Computing Task Scheduling with Joint Blockchain and Task Caching in Industrial Internet," Comput. Mater. Contin., vol. 75, no. 1, pp. 2101–2117, 2023, doi: 10.32604/cmc.2023.035530.

[35] B. Hu, Y. Shi, and Z. Cao, "Adaptive Energy-Minimized Scheduling of Real-Time Applications in Vehicular Edge Computing," IEEE Trans. Ind. Informatics, vol. 19, no. 5, pp. 6895–6906, 2023, doi: 10.1109/TII.2022.3207754.

[36] K. Loheswaran, "An upgraded fruit fly optimisation algorithm for solving task scheduling and resource management problem in cloud infrastructure," IET Networks, vol. 10, no. 1, pp. 24–33, 2021, doi: 10.1049/ntw2.12001.

[37] A. Aggarwal, P. Dimri, A. Agarwal, M. Verma, H. A. Alhumyani, and M. Masud, "IFFO: An Improved Fruit Fly Optimization Algorithm for Multiple Workflow Scheduling Minimizing Cost and Makespan in Cloud Computing Environments," Math. Probl. Eng., vol. 2021, no. Vm, 2021, doi: 10.1155/2021/5205530.

[38] D. Gabi et al., "Dynamic scheduling of heterogeneous resources across mobile edge-cloud continuum using fruit fly-based simulated annealing optimization scheme," Neural Comput. Appl., vol. 34, no. 16, pp. 14085–14105, 2022, doi: 10.1007/s00521-022-07260-y.

[39] Y. Nan, "An improved ant colony optimization algorithm based on immunization strategy," Adv. Mater. Res., vol. 490–495, pp. 66–70, 2012, doi: 10.4028/www.scientific.net/AMR.490-495.66.

[40] M. Farid, R. Latip, M. Hussin, N. Asilah, and W. Abdul, "SS symmetry A Survey on QoS Requirements Based on Particle Swarm Optimization Scheduling Techniques for Workflow Scheduling in Cloud Computing," 2020.

[41] W. N. Chen, Y. Shi, and J. Zhang, "An ant colony optimization algorithm for the time-varying workflow scheduling problem in grids," 2009 IEEE Congr. Evol. Comput. CEC 2009, pp. 875–880, 2009, doi: 10.1109/CEC.2009.4983037.

[42] E. Pacini, C. Mateos, and C. García Garino, "Balancing throughput and response time in online scientific Clouds via Ant Colony Optimization (SP2013/2013/00006)," Adv. Eng. Softw., vol. 84, pp. 31–47, 2015, doi: 10.1016/j.advengsoft.2015.01.005.

[43] M. P. Mathiyalagan, S. Suriya, and S. N. Sivanandam, "Modified Ant Colony Algorithm for Grid Scheduling," IJCSE) Int. J. Comput. Sci. Eng., vol. 02, no. 02, pp. 132–139, 2010.

[44] A. Liu and Z. Wang, "Grid task scheduling based on adaptive ant colony algorithm," Proc. - Int. Conf. Manag. e-Commerce e-Government, ICMeCG 2008, pp. 415–418, 2008, doi: 10.1109/ICMECG.2008.50.

[45] T. Engineering, "On Green Edge Computing with Machine Learning Applications by," no. April, 2024.

[46] I. Attiya et al., "An Improved Hybrid Swarm Intelligence for Scheduling IoT Application Tasks in the Cloud," IEEE Trans. Ind. Informatics, vol. 18, no. 9, pp. 6264–6272, 2022, doi: 10.1109/TII.2022.3148288.

[47] W. Yu et al., "A Survey on the Edge Computing for the Internet of Things," IEEE Access, vol. 6, pp. 6900–6919, 2017, doi: 10.1109/ACCESS.2017.2778504.

[48] R. Roman, J. Lopez, and M. Mambo, "Mobile edge computing, Fog et al.: A survey and analysis of security threats and challenges," Futur. Gener. Comput. Syst., vol. 78, pp. 680–698, 2018, doi: 10.1016/j.future.2016.11.009.

[49] C. Emary, "Ambiguous measurements, signaling, and violations of Leggett-Garg inequalities," Phys. Rev. A, vol. 96, no. 4, pp. 1–9, 2017, doi: 10.1103/PhysRevA.96.042102.