Computers, Materials & Continua

Tech Science Press

Doi:10.32604/cmc.2025.062451

ARTICLE

Efficient Task Allocation for Energy and Execution Time Trade-Off in Edge Computing Using Multi-Objective IPSO

Jafar Aminu[1,2,*], Rohaya Latip[1,*], Zurina Mohd Hanafi[1], Shafinah Kamarudin[1] and Danlami Gabi[2]

[1]Faculty of Computer Science and Information Technology, University Putra Malaysia, Selango, 43400, Malaysia
[2]Department of Computer Science, Kebbi State University of Science and Technology, Aliero, 1144, Nigeria

*Corresponding Authors: Jafar Aminu. Email: gs65795@student.upm.edu.my; Rohaya Latip. Email: rohayalt@upm.edu.my

Received: 18 December 2024; Accepted: 01 March 2025; Published: 03 July 2025

ABSTRACT: As mobile edge computing continues to develop, the demand for resource-intensive applications is steadily increasing, placing a significant strain on edge nodes. These nodes are normally subject to various constraints, for instance, limited processing capability, a few energy sources, and erratic availability being some of the common ones. Correspondingly, these problems require an effective task allocation algorithm to optimize the resources through continued high system performance and dependability in dynamic environments. This paper proposes an improved Particle Swarm Optimization technique, known as IPSO, for multi-objective optimization in edge computing to overcome these issues. To this end, the IPSO algorithm tries to make a trade-off between two important objectives, which are energy consumption minimization and task execution time reduction. Because of global optimal position mutation and dynamic adjustment to inertia weight, the proposed optimization algorithm can effectively distribute tasks among edge nodes. As a result, it reduces the execution time of tasks and energy consumption. In comparative assessments carried out by IPSO with benchmark methods such as Energy-aware Double-fitness Particle Swarm Optimization (EADPSO) and ICBA, IPSO provides better results than these algorithms. For the maximum task size, when compared with the benchmark methods, IPSO reduces the execution time by 17.1% and energy consumption by 31.58%. These results allow the conclusion that IPSO is an efficient and scalable technique for task allocation at the edge environment. It provides peak efficiency while handling scarce resources and variable workloads.

KEYWORDS: Keyword edge computing; energy consumption execution time particle swarm optimization; task allocation

1 Introduction

Modern computing operates on a decentralized paradigm, aiming to decrease reliance on distant cloud data centers to accelerate decision-making processes. Applications that require high bandwidth and low latency, such as augmented reality, autonomous vehicles, and the Internet of Things (IoT) [1,2]. In such scenarios, edge devices are often tasked with executing resource-intensive and delay-sensitive applications. These applications are divided into smaller, manageable tasks that are subsequently offloaded to edge nodes for execution via virtual machines or containers [3,4]. However, poor availability at edge nodes during task distribution poses a significant issue. Poor selection of edge nodes can lead to system performance degradation or even server crashes, which can significantly impact the overall reliability of the system [5].

The literature has addressed various task allocation issues in edge computing, but it hasn't extensively explored the continuous availability of edge nodes for task allocation [6]. This gap poses significant challenges

Copyright © 2025 The Authors. Published by Tech Science Press.
This work is licensed under a Creative Commons Attribution 4.0 International License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

to the reliable deployment of applications, preventing failures due to inconsistent node availability [7]. Edge computing refers to task allocations that involve distributing computational tasks across different edge nodes with variables in memory, bandwidth, and processing power requirements [8]. For instance, tasks requiring real-time execution should assign nodes with high processing power, while storage-intensive tasks should assign nodes with abundant storage resources. Similarly, we need to allocate delay-sensitive tasks to nodes closer to the data source to minimize delay. Therefore, we must perform task allocation thoughtfully, considering multiple, often conflicting objectives such as processing capacity, energy consumption, and execution time operating at different scales.

While edge computing provides significant benefits, it also encounters several critical challenges that impact its efficiency and scalability, especially regarding task allocation and resource management. These include energy consumption and execution time balance, adaptation to resource variability and dynamic workloads, and scalability in large-scale systems [9].

One of the most critical challenges in edge computing is to strike an optimal balance between energy consumption and execution time. Most tasks are computation-intensive, and their distribution to edge nodes with limited energy resources may result in excessive energy consumption, which reduces the system's lifetime and increases operational costs [10]. On the other hand, applications that have strict latency requirements necessitate fast execution of tasks, which may conflict with energy-saving policies [11]. The current approaches do not balance these conflicting objectives well, often leading to suboptimal system performance [12]. The environments of the edge are highly dynamic due to the fluctuating availability of resources, varying task demands, and heterogeneous edge nodes with different computational capacities. Static and semi-dynamic task allocation algorithms such as ICBA and AEDPSO cannot adapt to such changes in real time. As a result, inflexibility has often led to overloaded nodes, underutilized resources, and degraded performance, especially in resource-intensive applications [13].

As the number of tasks and edge nodes increases, existing algorithms face scalability issues, leading to longer execution times and higher energy consumption [14]. The complexity of managing large task sets and diverse resource types requires robust and scalable optimization frameworks, which many current methods fail to provide.

Normalization plays a major role in balancing these objectives, ensuring each is scaled appropriately within the optimization process [15,16]. Besides, edge computing environments are highly dynamic since devices are constantly joining or leaving the network. This fluctuating resource availability, therefore, requires flexible and adaptive task allocation algorithms that can make real-time adjustments. Furthermore, these algorithms need to face multi-objective challenges regarding reduction of execution time, enhancement of energy efficiency, ensuring data privacy, and a high level of QoS [17].

The metaheuristic approaches have recently attracted considerable interest in solving complex optimization problems in edge computing due to their flexibility and adaptiveness [18,19]. In these techniques, PSO has emerged as a powerful and simple technique, with much faster convergence and lower computational complexity compared to other optimization algorithms such as GA and ACO. The latter usually suffer from heavy computational loads. However, PSO may get entrapped into local optima at any instant, especially when the diversity of the population reduces with successive generation, thereby preventing the algorithm from exploring other globally optimal solutions. This matter is of primary importance when one tries to minimize both energy consumption and task execution time [20,21] For enhancing the effectiveness of PSO, three main issues must be emphasized: mutation strategy for the global optimal position, inertia weight management and acceleration coefficients adjustment, which allow avoiding of the premature convergence and increasing of the optimization accuracy Introduction of a mutation strategy for the global optimal position facilitates balancing between local and global searches and allows finding the optimal solution with

higher accuracy. Even with these improvements, PSO might have deficiencies regarding the diversity of the swarm after a few iterations to get away from local optima. This is the only way to ensure diversity in the population and thereby avoid the entrapment in the local optima and allow multi-objective minimization of energy consumption and time of execution [22]. The normalization of the fitness function causes appropriate scaling of both energy consumption and execution time so that no objective dominates the other in an optimization process. In the recent past, there has been growing interest in Multi-Objective Problems (MOPs) and thus seeks to provide a balance of trade-offs between conflicting objectives, especially when objectives have different units and scales, such as energy consumption and execution time. Techniques such as GOPMS and dynamic adjustment of acceleration coefficients are pivotal in solving these problems [23].

This work focuses on the emerging pressure that edge nodes face when the exponential development of data from massive edge devices and fast-growing user requests is growing explosively. When multitasking is performed on edge devices, execution time can be further reduced, but mostly at the increased cost of energy consumption. Offloading computation to edge nodes, on the other hand, reduces energy consumption but may lead to longer execution times, especially during peak load periods. Since the edge nodes are much closer to the users, a well-balanced strategy will be required for the optimization of energy use and execution time regarding overall performance.

In this regard, this work proposed a new task allocation approach in edge computing that integrates normalization techniques within the Improved Particle Swarm Optimization algorithm. Normalization serves to ensure that the usually conflicting objectives of energy consumption and task execution time get a fair deal and are not dominated by either during the optimization process. By normalizing both objectives to the same scale, the IPSO algorithm will further provide more efficient and balanced task allocations across available edge nodes. The incorporation of flexible weighting coefficients is one salient feature of the approach undertaken herein, enabling dynamic adaptation of the system to priorities in real time. It gives more weight, for example, to reducing energy consumption when energy efficiency is at a premium. Where the algorithm needs faster task execution, it could always give higher priority to the minimization of execution time. This flexibility makes IPSO particularly effective for dynamic environments such as smart manufacturing, smart homes, and smart transportation, where the ability to optimize both energy efficiency and execution speed becomes highly critical to maintaining high performance.

To the best of our knowledge, this is the first study to apply IPSO with normalization techniques to optimize execution time and energy consumption in an edge computing environment. This addresses the dual challenges major step forward in the area and further offers a more holistic and adaptable solution to modern edge computing systems' complex task allocation problems.

The main contributions of this study are as follows:

I. The various existing research on task allocation and the challenges coupled to energy consumption and execution time are modeled as multi-objective optimization problems in edge computing environments.
II. We developed an Improved Particle Swarm Optimization technique for edge computing multi-objective optimization. The suggested IPSO algorithm achieves a suitable trade-off between two goals: reducing energy usage and job execution time.
III. Comprehensive experimental comparisons were performed with the existing algorithms like ICBA and AEDPSO. Energy consumption decreases by 31.58%, and task execution time is reduced by 17.1% in the largest task size, which confirms the fact that IPSO is indeed scalable and effective in optimizing task allocation in edge computing.

This paper is structured as follows: Section 1 introduces the topic and provides an overview of the study. Section 2 reviews the relevant literature. The system model is presented in Section 3, as the problem description. Section 4 offers a detailed explanation of the proposed methods, while Section 5 outlines the experimental design. The results and discussion are provided in Section 6, and finally, Section 7 concludes the work, summarizing key findings and future directions.

2 Related Works

This research [24] presents a cluster-based Wireless Sensor Network (WSN) concept and a task model for edge computing. The variety of the model makes it more useful in practice because different tasks call for different kinds of resources, and different sensors offer different kinds of resources. The study uses a hybrid strategy that combines Genetic Algorithm (GA) and Ant Colony Optimization (ACO) to handle work allocation. The algorithm's main goals are to accomplish load balancing and energy conservation, which will ultimately increase the network's longevity. The experiments' outcomes show how well the algorithm works to optimize energy consumption while distributing the burden throughout the network. This study [25] focuses on optimizing transmission power allocation and task offloading scheduling in Mobile Edge Computing (MEC) systems that incorporate numerous independent jobs. Reducing the gadgets' energy consumption and execution delay is the goal. An alternating minimization-based low-complexity sub-optimal algorithm is suggested to reduce the weighted sum of execution delay and energy consumption in order to accomplish this. Given the transmit power allotment, flow shop scheduling theory is used to establish the task offloading scheduling or the order in which tasks are offloaded. Furthermore, for a given task offloading sequence, the optimal transmit power allocation is determined using convex optimization techniques. This research [26] examines how decisions about task offloading and resource allocation in a two-user Mobile Edge Computing (MEC) network are affected by inter-user task interdependence. The objective is to reduce the wireless devices' (WDs') weighted total energy consumption and task execution time. We created effective algorithms to optimize job offloading and resource allocation to do this. Moreover, we proved that a "one-climb" strategy governs the best offloading choices. We used this knowledge to develop a lower-complexity Gibbs sampling technique that helps determine the optimal offloading options. The suggested method considerably exceeds established benchmarks, according to simulation results, highlighting the need to take inter-user task interdependence into account in MEC systems. In this paper [23], the authors look at the simultaneous optimization of computing frequency allocation, communication rates, and task assignment for a multi-helper Mobile Edge Computing (MEC) system that is D2D enabled and assumes binary task offloading. While taking into account the energy and computing constraints of each user and helper, the goal is to minimize the overall computation latency under a TDMA communication protocol. Reference [27] examines the best-combined energy and task allocation issue for a wirelessly powered MEC system for a single user that experiences dynamic task arrivals over time. Within a finite time, horizon with many slots, the objective is to minimize the transmission energy consumption at the energy transmitter (ET) while respecting energy and task causality and task completion restrictions at the user. Using convex optimization methods, we obtained well-organized optimal offline solutions for both static and time-varying channel circumstances, presuming prior knowledge of task state information (TSI) and non-causal channel state information (CSI). The objective of this study is to reduce the amount of brown energy used by tackling the issues of workload distribution, virtual machine migration, and energy scheduling. The work suggests an optimal model for discrete time-slot scheduling that takes user demand fluctuation and green energy supply into account. It is demonstrated that the problem is NP-hard, and a relaxation-based heuristic solution is created to handle the uncertainty in the supply of green energy and the dynamic nature of consumer demand [28]. An approach to task allocation for cooperative edge computing is presented in this author [29] Initially,

it presents a task allocation model that relies on two edge nodes working together. The objective of this model is to minimize the average job completion time while meeting business requirements. Moreover, a technique called TCA-IPSO two-edge-node cooperative task allocation employing an improved particle swarm optimization is put forth. This algorithm efficiently addresses the problem of job allocation schemes in cooperative situations, which frequently tend to become trapped in local optima. It improves on the basic particle swarm optimization by using crossover and mutation strategies from genetic algorithms. In [30], the study focuses on scenarios where a mobile device handles multiple tasks or an application comprises various subtasks, the paper aims to tackle the challenge of task allocation in multi-task environments within MEC systems, striving to minimize total energy consumption while considering cases where a mobile device may manage one or more tasks simultaneously. However, this assumption may not align with real-world situations where a mobile device often has numerous tasks queued for execution. In [31], the paper examines the problem of minimizing total energy consumption in energy harvesting (EH)-enabled Mobile Edge Computing (MEC) networks by jointly optimizing the task offloading ratio and resource allocation. The analysis considers the task uplink transmission time, MEC computation time, and the time required for downloading computation results, addressing these factors simultaneously. However, the proposed method is its reliance on idealized assumptions about energy harvesting and network stability, which may not fully account for the dynamic and unpredictable nature of real-world MEC environments. In [32], the study proposes a heuristic method to address the resource allocation subproblem and an approximation algorithm for the offloading decision subproblem. The proposed approximation algorithm is demonstrated to be a polynomial-time approximation scheme, effectively balancing the trade-off between optimality and computational complexity. However, the study does not address energy consumption, a critical factor in the context of edge computing. In [33], the paper introduces several algorithms tailored for various scenarios. In cases where historical data is unavailable, a multi-round allocation algorithm utilizing Exponential Moving Average (EMA) prediction is proposed. Experimental results validate the effectiveness and importance of employing multiple rounds of transmission in such situations. However, the proposed method overlooks energy consumption, a critical factor in edge computing environments. In [34], the paper presents an online resource allocation approach inspired by offline optimization. It first formulates the energy minimization problem as a convex optimization under ideal conditions with known Time-State Information (TSI) and Channel-State Information (CSI), solving it using the Lagrange duality method. Leveraging these insights, a sliding-window-based online method is proposed for real-world scenarios. The results highlight significant improvements in energy efficiency for wireless-powered MEC systems. However, the proposed mechanism incurs significant overhead, which may impact its efficiency and scalability in practical applications.

Previous studies have shown that the classical techniques for task allocation in edge computing are resolved by artificial or optimal approaches, both of which perform well. These methods are extremely complex, though, and as a result, resource computation and energy usage go up. In an edge computing context, where the least amount of energy and execution time is needed, this could not be beneficial. However, some studies have used the popular meta-heuristic method of ACO and GA. By producing a realistic, high-quality answer that is often less complex than the ideal solution, the meta-heuristic technique ensures energy utilization. However, because they need more parameters to provide the same results, the GA and ACO utilize more energy than other meta-heuristic approaches. There is an increase in computation time and energy consumption because this component does not meet the service standards required to process Internet of Things applications in an edge computing environment. From this perspective, choosing a suitable approach to meet the platform's needs is essential. Therefore, when compared to other metaheuristics, the IPSO method requires the fewest parameters and produces reasonable results, making it the optimal option for workload allocation in edge computing. The MOP (IPSO) algorithm is used in this study because it has

fewer parameters, which helps it achieve two of its main objectives: a shorter execution time and less energy consumption. Therefore, it's critical to increase response times and manage a greater volume of tasks from IoT devices. According to the previously mentioned information, this study meets the need for reducing the energy and execution time.

Limitations of Existing Algorithms in Multi-Objective Optimization for Edge Computing

Although some existing algorithms such as the Improved Chaotic Bat Algorithm (ICBA) and Adaptive Evolutionary Dual Particle Swarm Optimization (AEDPSO) have been quite promising in tending to a few difficulties in task allocation at edge computing, they suffer critical limitations when dealing with multi-objective optimization problems. Among these, being prone to a local optimum with an early convergence rate is a major deficiency. This problem arises due to insufficient mechanisms that ensure diversity in the population, which is very necessary for the effective exploration of the solution space [35]. This lack usually results in their failure to provide globally optimal solutions for problems that involve complex domains.

Another limitation involves the use of pre-defined trade-offs among competing objectives, such as energy consumption vs. execution time, considered as strict dependence that leads to suboptimal task allocation when priorities change operationally and thus dynamically according to conditions arising at runtime in edge computing [36]. For instance, in those conditions where energy efficiency is crucial in peak workloads or low latency in delay-sensitive tasks, ICBA and AEDPSO cannot provide the flexibility to change according to those. Thus, these algorithms very often generate imbalanced and inefficient solutions in different scenarios.

Besides, the computational complexity of these algorithms serves as another barrier to their practical implementation. Iterative updates in ICBA, together with chaotic map calculations, bring many overheads and are not suitable for resource-constrained environments. Similarly, AEDPSO employs heavy load processing due to its two-update mechanism and thus serves as a barrier toward real-time applications. This high complexity reduces efficiency and limits scalability in larger, more diverse edge computing environments. The second aspect that ICBA and AEDPSO lag on is dynamic adaptability. Inherently, changes involve alterations to the availability of resources and modifications in task requests within dynamic edge computing environments. However, these algorithms do not have a means of adapting in real-time to such changes. Therefore, they tend to fail when on-the-fly adaptation is a prerequisite for successful performance, rendering them less reliable for real-world use cases [37]. In addition, there is poor scalability of the algorithms in practice since they tend to have inferior performance with increased task sets or different resource configurations.

Finally, both ICBA and AEDPSO have deficiencies in handling multi-objective problems whose objectives are of different units and scales. In situations where energy consumption and execution time are to be optimized together, the lack of strong normalization techniques often leads to an improper preference for one objective over the other. It then leads to imbalance and suboptimal solutions for task allocation and resource utilization.

3 System Model

Fig. 1 shows the proposed system model designed for efficient task allocation in the edge computing environment. The proposed framework is based on a PSO-based task allocation approach to optimize the resource utilization at the edge of the network. Diverse devices, including smartphones, laptops, and IoT devices, interact with users, generating computation-intensive tasks that nearby edge nodes process. These offload tasks to edge nodes, leveraging proximity to edge resources to minimize both execution time and energy consumption. Once generated, these tasks enter the task queues on user devices, where they undergo

sorting based on resource requirements, priority, or urgency. This structure hence makes the allocation process capable of handling diversified loads. The PSO task allocation module forms the heart of the system and optimizes task distribution across edge nodes by continuously assessing available resources. It selects the most appropriate edge node for each task, ensuring reduced energy consumption and execution time. The PSO algorithm dynamically adapts to changes in the surroundings, for instance, resource and task variations for optimal utilization of the available resources at any instant. Every edge node of this system has its own set of computational resources and processing powers. This process executes the assigned task at a node in a manner that reduces energy consumption; hence, keeping the execution time as low as possible. The system will perform each task at the highest speed it can; therefore, this system will be appropriate for use cases that demand both energy efficiency and rapid execution of tasks. Further improvements involve a system of real-time feedback within the system to monitor the performance of edge nodes. This will ensure that the PSO task allocation module provides real-time adjustments to the allocation strategy in response to changes in system conditions, such as surges in task volumes or shifts in resource availability. It can keep the system highly efficient by dynamically fine-tuning task distribution according to the inherent variability of edge computing environments. This combination of real-time feedback and adaptive PSO task allocation ensures that, even under dynamic conditions, the system consistently performs well, optimizes resource usage, minimizes energy consumption, and maintains low execution times. The adaptiveness of the system brings much effectiveness in handling the complexities of real-world edge computing, wherein both the demand for tasks and resource availability may fluctuate without any approximation.

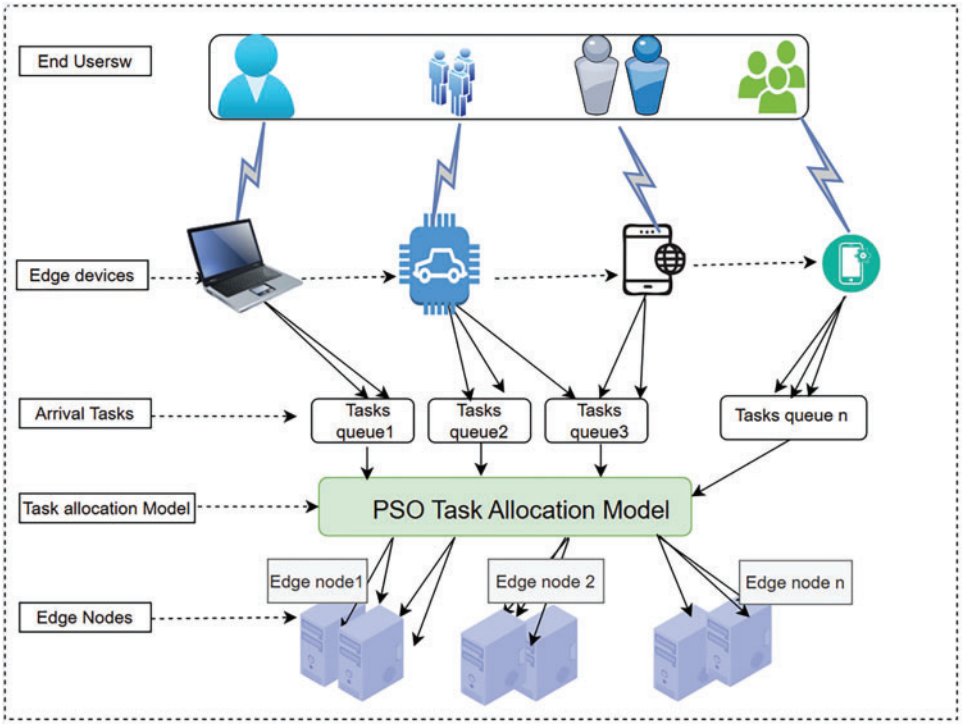

Figure 1: Task allocation process

The figure illustrates the overall architecture of the task allocation process in an edge computing environment. It consists of multiple layers: (i) User Devices Layer, where mobile devices such as smartphones, IoT devices, and sensors generate computational tasks; (ii) Edge Layer, which includes multiple edge nodes

responsible for executing offloaded tasks using virtual machines (VMs) or containers; and (iii) Optimization Module, which employs the Improved Particle Swarm Optimization (IPSO) algorithm to allocate tasks efficiently. The IPSO-based task allocation mechanism dynamically evaluates available resources, considering factors such as computational capacity, energy consumption, and network bandwidth, to optimize execution time and energy consumption. The system also integrates a real-time feedback loop to continuously monitor resource availability and workload distribution, ensuring optimal task allocation

3.1 Problem Description

In such scenarios, edge nodes handle massive amounts of data from IoT devices for efficient processing. An efficient task allocation strategy is thus highly important for a multi-user, multi-server edge computing environment to perform optimally. In such resource-limited and dynamic environments, it is very challenging to minimize energy consumption on the one hand and reduce overall task execution time.

The system consists of a set of tasks $T = (t1, t2, \ldots, t_n)$ and edge nodes $R = (r1, r2, \ldots, r_m)$. Each task t_i is characterized by its computational demand (C_i in CPU cycles) and input data size (D_i MB). Task Size refers to the number of tasks in each workload batch, where each task has a predefined computational demand measured in millions of CPU cycles (MC). Each edge nodes r_j is defined by its processing capacity (P_j in CPU cycles) per second and energy consumption rate (E_j, in joules per second).

3.2 Energy Consumption Model

The total energy consumption of the system is calculated as the sum of the energy consumed by all edge nodes while processing the assigned tasks. For each task t_i assigned to edge node r_j, the energy consumption E_{ij} is given by:

$$E_{ij} = P_j \times T_{ij} \tag{1}$$

where P_j represents the power consumption rate of the edge node r_j, measured in watts (w), representing the energy consumed by the node per unit during task execution. And T_{ij} is the execution time of tsk t_i on edge node r_j, calculated as the time required to process the task based on the node's computational capacity and the task's computational demand.

$$T_{ij} = \frac{C_i}{P_j} \tag{2}$$

$$E_{total} = \sum_{i=1}^{n} \sum_{i=0}^{m} x_{ij} \times P_j \times \frac{C_i}{P_j} \tag{3}$$

where x_{ij} is a binary variable indicating whether the task t_i is assigned to a node r_j($x_{ij} = 1$ if assigned, $x_{ij} = 0$ otherwise).

Delay Model. The delay for each task includes both transmission delay $T_{transit,ij}$ and execution delay T_{ij}, it can be calculated by the following formula.

$$T_{transit,ij} = \frac{D_i}{B_j} \tag{4}$$

where D_i represents the amount of data required by the task t_i to be processed. typically measured in megabytes (MB) and includes all input data that must be transmitted to the edge node before computation can begin. B_j denotes the communication bandwidth capacity available at the edge node r_j, measured in

megabits per second (Mbps). Bandwidth determines the speed at which input data is transmitted to the edge node.

Execution Delay. Reflects the total time required to complete a task after it is offloaded to an edge node. It consists of two key components: the transmission delay $T_{trnsmit,ij}$ which accounts for the time to transfer the task's input data to the edge node, and the processing delay T_{ij}, which represents the time taken by the node to execute the task. Minimizing execution delay is critical in edge computing, particularly for latency-sensitive applications, as it directly impacts the system's performance and the user experience. Efficient task allocation strategies must balance these delays while considering the computational and network resources available by the following formula.

$$T_{total,i} = T_{trnsmit,ij} + T_{ij} \tag{5}$$

$$T_{total} = \sum_{i=1}^{n} \sum_{j=1}^{m} x_{ij} \times \left(\begin{matrix} D_i \\ Bj \end{matrix} + \frac{C_i}{P_j} \right) \tag{6}$$

The system-wide delay T_{total} represents the aggregate delay experienced by all tasks in the system. It is calculated as the sum of the delays for each task t_i assigned to each edge node r_j. The delay for each task consists of two components: transmission delay $\left(\frac{D_i}{B_j}\right)$, which accounts for the time to transfer the task's input data to the edge node, and execution delay $\left(\frac{C_i}{P_i}\right)$, which represents the time required to process the task on the assigned edge node.

3.3 Balancing Energy Consumption and Execution Time in Task Allocation as Multi-Objective Problem (MOP)

The proposed Improved Particle Swarm Optimization (IPSO) algorithm optimizes task allocation in edge computing environments by employing a multi-objective fitness function that minimizes both energy consumption and execution time. This is achieved through the normalization of these objectives, ensuring neither dominates the optimization process the goal of a Multi-Objective Problem (MOP) is to simultaneously optimize multiple conflicting objectives. Task allocation in edge computing, involves finding a balance between minimizing energy consumption and execution time. In general, an MOP can be represented by m decision variables and n objectives as follows:

Minimize $y = f(x) = f_1(x), \ldots, f_n(x)$ where $x = (x_1, \ldots, x_m) \in X$ is an m-dimensional decision vector, X represents the search space, and $y = (y_1, \ldots, y_m) \in Y$ is the objective space. Generally, no single solution is ideal for all objectives simultaneously. Instead, the solution space consists of multiple potential outcomes, each optimal for one or more objectives.

3.4 Normalization in Multi-Objective Optimization

Normalization in multi-objective optimization is essential when one has to deal with several objectives in different units and scales, such as energy consumption and execution time. This ensures that all objectives contribute equally to the optimization process and that no objective dominates another due to magnitude. We normally normalize each objective, $f_i(x)$ is typically done by scaling the objective values to a common range, such as (0, 1). For instance, we determine the normalized execution time $T_{(norm)}$ and normalized energy consumption E_{norm} are calculated as follows:

$$T_{(norm)} = \frac{T_{(total)} - T_{(\min)}}{T_{(max)} - T_{(\min)}} \tag{7}$$

$$E_{(norm)} = \frac{E_{total} - E_{(\min)}}{E_{max} - E_{(\min)}} \tag{8}$$

where $E_{(norm)}$ and $T_{(norm)}$ represent normalized energy consumption and execution time, respectively, $T_{(total)}$ and $E_{(total)}$ are the total execution time and energy consumption for a given solution, $T_{(\min)}$ and $T_{(max)}$ are the minimum and maximum execution times observed in the current population. $E_{(\min)}$ and $E_{(max)}$ are the minimum and maximum energy consumption values observed. This normalization scales both objectives, allowing us to combine them in the fitness function using the weighted sums method or any other multi-objective optimization technique. We combined the normalized objectives using the weighted sum technique into one fitness function.

3.5 Fitness Function

The fitness function ensures that the solution reaches a compromise between these conflicting objectives of minimizing energy consumption and execution time through the assessment of the position of a particle during optimization. To balance these two factors, the fitness function implements the weighted sum approach, which merges these two objectives into a single computable result. Normalization needs to be done so that all the objectives will have an equal contribution to the fitness value since energy consumption and execution time are measured in different units and can have widely differing scales. The calculation of the normalized fitness function is as follows:

$$F = a_1 \cdot E_{(norm)} + a_2 \cdot T_{(norm)} \tag{9}$$

The balance between energy consumption and execution time strongly depends on the weighting factors a_1 and a_2 The latter can be modified over time based on the pursued goals from the system in that instant. For example, if the system is required to be more energy efficient, more weight should be attributed to a_1, whereas overweight is to be put on a_2 if speed of execution is more critical. The fitness function allows flexibility in the optimization process by dynamically changing priorities according to the real-time condition. Moreover, in an edge computing environment, there is often a necessity where both these factors are critical. Hence, equal importance can be given by setting $a_1 = a_2 = 0.5$, for energy consumption and execution time, respectively. However, it can share those weights to adapt to various operational requirements and render the system very effective for responses against dynamic edge computing conditions. It means that the approach towards task allocation will be optimized with respect to prevailing conditions and variation in task volume, resource availability, and system performance requirements for efficient adaptive task management. Tasks are allocated to edge nodes dynamically, considering resource availability, processing capacity, and network bandwidth. The algorithm also integrates the Global Optimal Position Mutation Strategy (GOPMS) to enhance solution diversity and avoid premature convergence, ensuring a more effective exploration of the solution space. These features enable IPSO to adapt to the dynamic and heterogeneous nature of edge environments, achieving a balanced optimization of energy consumption and execution time.

4 Improved Particle Swarm Optimization (IPSO)

Particle Swarm Optimization (PSO), introduced by Kennedy and Eberhart, is inspired by the behaviour of social systems observed in animals [38]. It is a population-based evolutionary algorithm, commonly referred to as a swarm, where each swarm comprises, multiple particles representing potential solutions. Each particle retains local knowledge of its best solution and shares information about the global best solution identified by the entire swarm. In PSO, the search space is divided into D dimensions, and the algorithm

employs a velocity vector for each particle to navigate the search space. Unlike other evolutionary algorithms, PSO does not use selection operators, which can result in increased computational time. The movement of particles within the search space depends on their direction and velocity, guiding them toward potential solutions. However, a significant drawback of PSO is its tendency to become trapped in local optima [22]. To address this limitation, the Improved Particle Swarm Optimization (IPSO) algorithm has been proposed.

4.1 Enhancements to Standard PSO

The standard PSO is one of the well-known optimization algorithms due to its simplicity and speed of convergence. However, it faces serious challenges when applied to multi-objective optimization in edge computing, such as premature convergence to local optima, poor adaptability to dynamic environments, and difficulty in balancing conflicting objectives such as energy consumption and execution time [39]. Here in this proposed improved PSO algorithm, several improvements are introduced for enhancement in robustness and efficiency when applied for optimization concerning multi-parameters. Among such improvements, the incorporation of GOPMS stands supreme. The rationale behind GOPMS usage would be the issue of the inability to avoid standard PSOs reaching prematurely to converge divergence as facilitated by this treatment. GOPMS mutates the global best position iteratively by dimensionality in the solution space [40]. This does not allow particles to rest in a local optimum, and the particles continuously explore the search space, hence enhancing the capability of convergence to global solutions in complex multi-objective problems [41]. Dynamic adjustment of the inertia weight is another critical improvement in IPSO. While standard PSO normally uses a fixed or linearly decreasing inertia weight, in IPSO it is changed dynamically depending on the development of the iteration. A larger inertia weight during the early stages of optimization allows for exploration, enabling the swarm to search out a wider solution space. As the iterations of optimization proceed, the inertia weight is reduced to allow the algorithm to focus on exploitation and refinement toward the best solutions. This adaptive approach has thus far demonstrated a good balance between exploration and exploitation in improving the convergence performance of the algorithm. In addressing the conflicting objectives of energy consumption and execution time, IPSO applies normalization techniques that scale these objectives to lie within a common range. This prevents one objective from dominating the others because of its scale or unit differences. It does this by ensuring that both objectives contribute equitably to the fitness function in achieving a well-balanced trade-off, which is very important in dynamic and resource-constrained edge computing environments. Another enhancement involves an adaptive tuning of acceleration coefficients, which govern the influence of personal and global best positions on the movement of particles. Unlike standard PSO, which implements fixed values for these coefficients, IPSO dynamically adjusts them throughout the optimization phase. In that case, in the early iteration, a larger value of a cognitive component induces particles to explore around their optimum, while a larger social component in the later iterations focuses the particles onto the global best position to enable an accurate convergence, keeping diversity at the solution space.

Finally, IPSO embeds real-time feedback mechanisms to adapt dynamically to the ever-changing conditions of edge computing environments. These mechanisms monitor task arrival rates, resource availability, and workload fluctuations, enabling the algorithm to update its task allocation strategies in real time. This adaptability ensures efficient resource utilization and responsiveness to dynamic workloads, making IPSO highly suitable for edge computing scenarios.

We consider everyone as a particle flying in the D-dimensional search space, lacking both volume and weight and moving at a specific velocity. We dynamically update each particle's velocity based on its personal experience and the cumulative experience of the swarm. Let M be the population size of the swarm. In a D-dimensional search space, we can define the position and velocity of the ith particle as follows:

$X^i = (x_1^i, x_2^i, \ldots, x_{nd}^i$ and $V^i = (v_1^i, v_2^i, \ldots, v_{nd}^i)$, respectively.

Let $Pbest = (pbest_1^i, pbest_2^i, \ldots, pbest_{nd}^i)$ denotes its personal best position, and $Gbest = (gbest_1, gbest_2, \ldots, gbest_{nd})$ the global best position achieved by the entire swarm is denoted as:

The updated velocities and positions for each particle in the next iteration are calculated using the following two equations:

$$v_{ind}^{(k)} = v_{ind}^{(k+1)} + c_1 \cdot rand_1 \cdot (pbest_{nd}^i - x_{ind}^{(k+1)}) + c_2 \cdot rand_2 \cdot (gbest_{nd} - x_{ind}^{(k+1)}) \tag{10}$$

$$x_{ind}^{(k)} = x_{ind}^{(k+1)} + v_{ind}^{(k)} \tag{11}$$

where 1 and 2 are constants referred to as acceleration coefficients, and 1 and 2 are two randomly generated, equally distributed values from the interval [0, 1].

The PSO approach performs substantially better when the inertia weight (w) varies linearly over generations, as demonstrated by [42]. This is a result of the algorithm's standard version's poor handling of optimization issues. Mathematical ideas are represented by Eqs. (10) and (11).

$$v_{ind}^{(k)} = w \cdot v_{ind}^{(k+1)} + c_1 \cdot rand_1 \cdot (pbest_{ind} - x_{ind}^{(k+1)} + c_2 \cdot rand_2 \cdot (gbest_{nd} - x_{ind}^{(k+1)}) \tag{12}$$

where w is the inertia weight that controls the influence of the particle's previous velocity on its current velocity.

4.2 Inertia Weight (w)

The inertia weight controls the balance between exploration and exploitation in the PSO algorithm. Dynamically varying inertia weights enable IPSO to explore a more extensive solution space during early iterations while refinement is done with the same focus in later iterations. Thus, dynamic inertia ensures that IPSO avoids convergence to local optima and will identify the most optimal solutions within the defined boundaries and objective function values.

The inertia weight w is dynamically adjusted according to the following equation:

$$w = w_{max} - \left(\frac{iter}{iter_{max}}\right) \cdot (w_{max} - w_{min}) \tag{13}$$

where w_{max} and w_{min} are the maximum and minimum values of the inertia weight, respectively, $iter$ is the current iteration number and $iter_{max}$ is the maximum number of iterations. Typically, w_{max} is set to 0.9, and w_{min} is set to 0.4. These values help balance exploration and exploitation during the optimization process. Compared to the traditional approach, this enhanced algorithm demonstrates significantly improved performance due to its ability to dynamically adjust the inertia weight. This technique helps the particles maintain diversity and avoid premature convergence to local optima.

4.3 Mutation Strategy of Global Optimal Position (GOPMS)

The PSO approach draws all particles to the global optimal position by using it as an attractor during the evolutionary process. This is where all the particles will eventually converge. Therefore, if particles are unable to efficiently update the ideal global location and instead become caught in the local optimal zone, the particle swarm will materialize early. To address this problem, this work employs the GOPMS mutation approach, which stands for global optimal position. The GOPMS operator iteratively alters the value of each dimension by generating random numbers with a uniform distribution, while the other dimensions remain

fixed. If the suitability of the new position is higher than that of the previous one, it drops the old one and keeps the new one [43]. The GOPMS algorithm is presented in Algorithm 1.

Algorithm 1: GOPMS

Input: Current global best solution (*Gbest*), search space boundaries, and number of mutation iterations
Output: Updated global best solution (*Gbest*),
1 **Begin**
2 For each dimension d in $Gbest$
For each mutation $iterk$
Mutate the dth dimension of $Gbest$
MutateGbest = Random value within the search space of dimension d
If the fitness of mutateGbest is better than the fitness $Gbest$:
$Gbest$ = MutateGbest
End for
3 **End For**
4 **Return updated** $Gbest$

4.4 Acceleration Coefficients

The algorithm IPSO introduces a new parameter automation method to enhance its capability for optimization. A new mutation strategy for the global optimal position combines with the adaptive application of acceleration coefficients, denoted as (c_1 and c_2). Fig. 2 emphasizes how important the roles these coefficients have in particle movement dynamics are and how they affect the process of optimization. The trade-off between exploration-searching new solutions and the exploitation-fine-tuning of the best-found solutions underlies any successful task allocation scheme in edge computing. Acceleration coefficients define the step to take by the particle based on the personal best experience (best individual experience) and social best solution. The role of these coefficients is as follows:

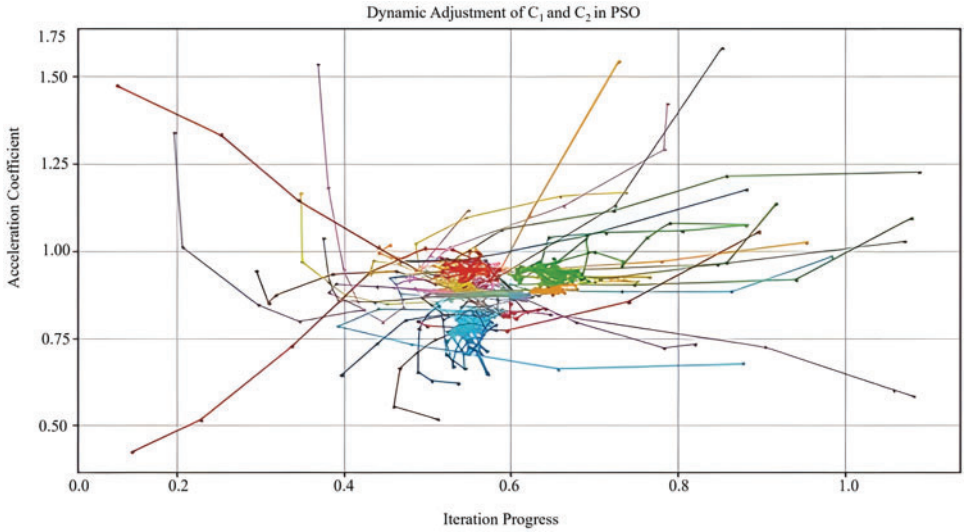

Figure 2: Coefficient of acceleration

Cognitive Coefficient c_1 Exploration Component. The cognitive coefficient c_1 determines the potential of a particle to explore independently by using its best personal solution. A higher value of c_1 in the early iterations promotes exploration, enabling the particles to investigate a wider range of possible solutions and escape local optima. However, excessive exploration slows down convergence, leading to unstable search behavior and suboptimal performance.

Social Coefficient c_2 Exploitation Component. The social coefficient c_2 determines the strength of a particle's attraction to the best solution found so far by the swarm. The higher c_2 in later iterations enhances convergence, directing particles toward the most optimal solution. If c_2 is, the stronger the convergence, and the more the particles will be guided toward the optimal solution, which may be achieved in later iterations. If set too high too early, the algorithm might converge prematurely, hence losing diversity for an effective global search.

Dynamic Adjustment of $(c_1$ and $c_2)$ in IPSO. Unlike standard PSO, where c_1 and c_2 remain static, IPSO dynamically adjusts these coefficients to ensure an optimal balance between exploration and exploitation:

- Early search phase: a higher c_1 encourages diverse exploration, preventing premature convergence.
- Later search phase: a higher c_2 shift the focus toward convergence, refining the best solutions.

This adaptive mechanism prevents premature convergence and enhances the ability of the algorithm to optimize execution time and energy consumption in edge computing task allocation. By dynamically tuning the acceleration coefficients, IPSO achieves improved efficiency in task distribution, making it highly suitable for dynamic and resource-constrained environments.

Fig. 2: influence of the cognitive coefficient c_1 and social coefficient c_2 on particle movement strategy in IPSO. The best dependency of the cognitive coefficient c_1 introduces independent search space, whereas the social coefficient c_2 allows for convergence to the optimum global solution. It dynamically adjusts these coefficients: in early iterations, higher c_1 increases exploration to avoid early convergence, while higher c_2 in later iterations favor convergence toward the optimal solutions. Dynamic adaptation in this manner enables IPSO to balance exploration and exploitation effectively for better performance in task allocation within edge computing environments. The IPSO algorithm is presented in Algorithm 2.

Algorithm 2: IPSO algorithms

Input:	Task set $T = \{t_i, t_2, \ldots, t_n\}$, a set of edge nodes $R = \{r_1, r_2, \ldots, r_m\}$
Output:	Optimized task-to-node mapping, (x_{ij}). (E_{total}) and (T_{total})
1.	Initialize particle positions (X_i) and velocities (V_i)
2.	Set inertia weight (w), acceleration coefficients (c_1, c_2), and normalization parameters.
3.	Define max iterations $(itermax)$, set iteration counter to 0.
4.	For each particle i, Normalize energy (E_{norm}) and execution time (T_{norm})
5.	Calculate fitness: $F = a_1 \cdot E_{(norm)} + a_2 \cdot T_{(norm)}$
6.	Update Personal Best $(Pbest)$:
7.	If current fitness Fi, is better than $Pbesti$, update $Pbesti$.
8.	Set $Gbest$ as the position of the particle with the best fitness.
9.	Apply GOPMS, mutate dimensions $Gbest$ of and update if the mutation improves fitness
10.	Update $v_{ind}^{(k)} = w \cdot v_{ind}^{(k+1)} + c_1 \cdot rand_1 \cdot (pbest_{ind} - x_{ind}^{(k+1)} + c_2 \cdot rand_2 \cdot (gbest_{nd} - x_{ind}^{(k+1)})$
11.	Update $x_{ind}^{(k)} = x_{ind}^{(k+1)} + v_{ind}^{(k)}$
12.	Ensure each task t_i is assigned to exactly one edge node by Eq. (2)

(Continued)

Algorithm 2 (continued)

13.	Ensure the total computational demand of tasks assigned to each edge node does not exceed its capacity P_j capacity constraint by Eq. (3)
14.	Adjust inertia weight (w) dynamically
15.	Terminate if max iterations or other criteria are met; otherwise, repeat Steps 4–14.

4.5 Complexity Analysis of the IPSO Algorithm

The computational complexity of the proposed Improved Particle Swarm Optimization (IPSO) algorithm is determined by analyzing the individual steps and components of the algorithm. The overall complexity depends on factors such as the number of particles (n_p), the number of tasks (n_t), the number of edge nodes (n_r), and the maximum number of iterations $(iter_{max})$. The algorithm ensures that each task is assigned to exactly one edge node, and the total computational demand of tasks assigned to a node does not exceed its capacity. The overall Complexity is: $O(iter_{max} \times n_p \times n_t \times n_r)$.

5 Experimental Setup

In this part, we test our proposed IPSO algorithm against the ICBA, an earlier task allocation technique, and the AEDPSO methods from [44,45]. In the edge computing environment, our evaluation focuses on two key metrics: task execution time and energy consumption. Mobile devices generate diverse applications, each consisting of multiple functions that demand efficient management of edge resources.

The setup involves four edge nodes, each connected to multiple mobile devices. The simulation considers four task sets with total workloads of 40, 80, 120, and 160, respectively. To introduce unpredictability, task lengths are generated randomly. In our experiments, Task Size represents the total number of tasks in a workload batch. Each task has a computational requirement measured in millions of CPU cycles (MC), which determines execution time and energy consumption. The varying task sizes (40, 80, 120, 160) allow us to analyze the performance of the proposed algorithm under different workload conditions. The hardware configuration includes a high-performance system equipped with an Intel Core i7 processor (3.8 GHz, 8 cores), 16 GB of RAM, and an NVIDIA GTX 1080 GPU, enabling robust parallel processing. MEC servers and mobile devices are emulated using virtual machines designed to simulate typical mobile processor capabilities. The software environment is based on Ubuntu 20.04 LTS, utilizing Python 3.8 along with essential libraries for optimization and performance analysis. The algorithm is tested using both synthetic and real-world datasets. Synthetic datasets generate task offloading requests with varying parameters such as task size and computational requirements, while real-world datasets provide realistic task flow patterns. Evaluation scenarios are designed to reflect diverse operational conditions, including variations in network bandwidth, latency, and MEC server loads. Table 1 outlines the key parameters used in our simulations, providing an overview of the experimental setup.

Table 1: Parameters and their description

Parameters	Descriptions
X_i	Position of particle i in the search space
V_i	Velocity of particle i
w	Inertia weight, controls exploration and exploitation
c_1, c_2	Acceleration coefficients for personal and global best influences

(Continued)

Table 1 (continued)

Parameters	Descriptions
$iter$	Current iteration number
$iter_{mx}$	Maximum number of iterations
E_{norm}	Normalized energy consumption
E_{norm}	Normalized execution time
a_1, a_2	Weighting coefficients for energy and time objectives
F	Fitness value of a particle
$Pbest_i$	Best position found by particle i so far
$Gbest_i$	Best position found by the entire swarm
$rand_1, rand_2$	Random numbers uniformly distributed in [0, 1]
t_i	Task i
r_j	Edge node j
P_j	Processing capacity of edge node j
$v_{ind}^{(k)}$	velocity of the k-th particle for dimension ind
$x_{ind}^{(k)}$	Position of the k-th particle for dimension ind

6 Results Discussion

6.1 Performance of IPSO with an Increased Number of Tasks

The proposed Improved Particle Swarm Optimization algorithm is analyzed in terms of performance with varying task sizes to check scalability and efficiency in optimizing energy consumption and execution time. The trends that are evident show that the proposed IPSO is robust and scalable to handle the increase in workload for edge computing.

6.2 Trends in Energy Consumption

While increasing the size of tasks increases the overall energy consumption in the system due to higher computational demand, IPSO shows a clear ability to dampen this increase by effectively distributing tasks among edge nodes with the lowest energy consumption rates. For example, for smaller task sizes-say, 40 tasks energy savings of IPSO over ICBA and AEDPSO are moderate but significant. Thus, with larger task sizes increasing, such as 160 tasks, energy savings become extensive: in particular, the energy saving achieved by IPSO amounts to 31.58% compared to the baseline methods. It shows the scalability of IPSO when optimizing energy consumption even for workloads that become increasingly resource-intensive.

6.3 Trends in Execution Time

The execution time of tasks is another critical metric that scales with the number of tasks in the system. IPSO effectively minimizes execution time through its dynamic optimization mechanisms, ensuring that tasks are allocated to the most suitable edge nodes based on their computational capacity and proximity. At smaller task sizes, IPSO already outperforms ICBA and AEDPSO, reducing execution time by approximately 17.1% for 160 tasks. This trend demonstrates IPSO's ability to handle larger workloads efficiently while maintaining low latency, which is critical for delay-sensitive applications in edge computing.

6.4 Interpretation of Results

The observed trends in energy consumption and execution time indicate that IPSO's enhancements, including dynamic inertia weight adjustment and the Global Optimal Position Mutation Strategy GOPMS, play a pivotal role in maintaining performance as task sizes increase. These features enable IPSO to achieve a balanced trade-off between energy efficiency and execution speed, addressing the core challenges of multi-objective optimization in edge computing.

6.5 Impact of Task Size Variability on Energy Consumption and Execution Time

Large variability in task size significantly influences energy consumption and execution time since a bigger task requires more computational resources and time for processing. In such cases, the proposed IPSO algorithm copes with this variability through dynamic mapping of tasks to edge nodes that possess adequate computation and bandwidth. For example, larger tasks can be mapped onto nodes that possess high processing capability, reducing execution delay and thus preventing inefficient energy utilization arising from resource overloads. Normalizing energy consumption and execution time inside the fitness function balances both objectives so that IPSO can keep system performance according to the changes in the task size. As discussed in experimental results, IPSO adapts well in contrast to the baseline algorithms, ICBA and AEDPSO, which performed well in optimizing both energy consumption and execution time on diverse workloads.

Fig. 3 shows the energy Consumption Results for the energy consumption of the different number of tasks by the three algorithms shown in Table 2: ICBA, AEDPSO, and IPSO. The proposed IPSO algorithm exhibits the best performance in terms of energy consumption. Energy consumption decreases significantly with the increase of task size. For instance, when the task is 160, the proposed IPSO consumes only 5.2 units of energy consumption, whereas ICBA and AEDPSO consume 7.6 and 5.5 units, respectively. It achieves this reduction in energy consumption through the optimized task allocation strategy of IPSO, which tries to distribute the tasks among edge nodes with maximum efficiency and minimum superfluous energy consumption. Normalization and dynamic adaptation in the approach also make IPSO capable of handling variable task loads without compromising energy efficiency. Therefore, it is a highly suitable solution for an energy-constrained environment like mobile edge computing. By contrast, IPSO is scalable and practical for actual edge computing applications since it can be adapted to achieve energy efficiency for tasks of various sizes.

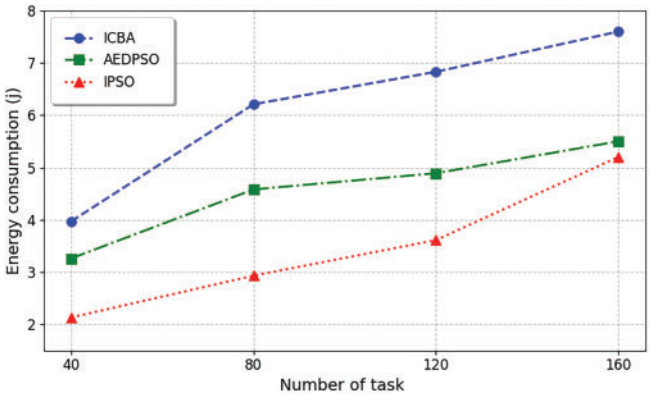

Figure 3: Compare the suggested IPOS algorithm's energy consumption

Table 2: Energy consumption results for different numbers of tasks

Algorithms	Number of tasks			
	40	80	120	160
ICBA	3.97	6.21	6.83	7.6
AEDPSO	3.26	4.58	4.89	5.5
IPSO	2.13	2.93	3.61	5.2

Fig. 4 presents the execution time it is obvious from the execution time results shown in Table 3 that in terms of reducing the execution time, IPSO always outperforms ICBA and AEDPSO. For example, at a task of 160, the execution time by IPSO was 34.98 s, while those by AEDPSO and ICBA were 40.29 and 42.2 s, respectively. This can be explained by the improved balance between exploration and exploitation in IPSO. Adopting GOPMS and adaptive inertia weight enables IPSO to make an appropriate task allocation to the most suitable edge nodes, optimizing resource utilization and hence minimizing higher execution time. This is particularly critical in edge computing, whereby applications in autonomous systems and smart cities demand low latency times and fast task execution. The fact that IPSO has been able to perform a reduction of execution time for all task sizes only strengthens this idea of how effective it is in managing the varied workload demand from modern edge computing systems.

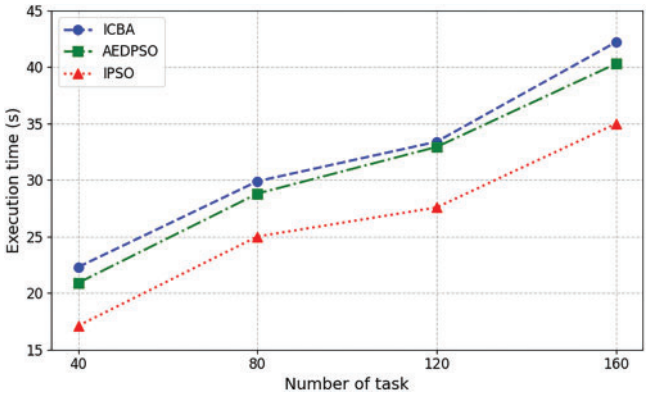

Figure 4: Compares the suggested IPOS algorithm's execution time

Table 3: Execution time results for different numbers of tasks

Algorithms	Number of task			
	40	80	120	160
ICBA	22.31	29.9	33.4	42.2
AEDPSO	20.91	28.81	32.94	40.29
IPSO	17.11	25.01	27.58	34.98

Fig. 5 evaluates the performance of the proposed IPSO allocation algorithm compared to traditional methods for a fixed set of 40 tasks executed on varying numbers of virtual machines (VMs). The number

of VMs was progressively increased from 10 to 40 in increments of 4, and the corresponding execution times were recorded. The results, depicted in Fig. 5, show a significant reduction in execution time as the number of VMs increased, regardless of the algorithm employed. This improvement is due to the additional VM resources in the edge environment enabling more efficient parallel processing, thereby accelerating task completion. The findings highlight that the proposed IPSO algorithm achieved the shortest execution times. For example, with 40 VMs, the IPSO algorithm completed all 40 tasks in 35.0 s, whereas the EA algorithm required 45.5 s, reflecting a time increase of more than 10 s. Other traditional algorithms took at least 10% longer to complete the tasks compared to IPSO. This superior performance is attributed to the iterative optimization of task allocation using the GOPMS method, which is seamlessly integrated into the IPSO approach. This integration not only minimized completion times but also enhanced energy efficiency.

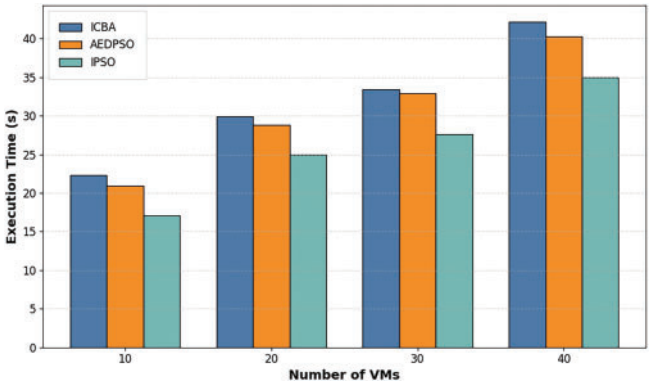

Figure 5: Number of VMs against execution time

Fig. 6 presents the analysis of task groups ranging from 40 to 160 tasks. The results highlight that the IPSO algorithm delivers substantial energy savings compared to the EA-DFPSO algorithm, achieving reductions of 22.5% for 40 tasks, 5.2% for 80 tasks, 15.2% for 120 tasks, and 9.1% for 160 tasks.

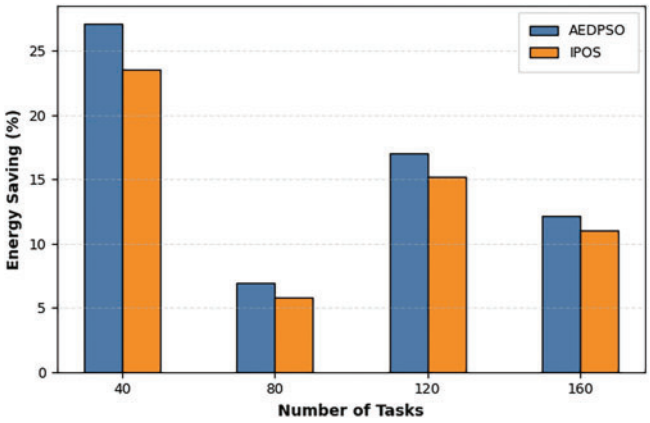

Figure 6: Energy saving for IPSO with EA-DFPSO

Fig. 7 illustrates the performance comparison between the proposed IPSO algorithm and the Improved Chaotic Bat Algorithm (ICBA) when allocating four sets of tasks, ranging from 40 to 160 tasks. The IPSO algorithm demonstrated significantly better energy efficiency compared to ICBA, achieving energy

conservation rates of 18.20% for 40 tasks, 17.20% for 80 tasks, 16.90% for 120 tasks, and 17.50% for 160 tasks. While the ICBA provides a relatively balanced task allocation by creating nearly equal execution slots for all tasks, this characteristic can lead to delays in completing longer-running tasks, highlighting a limitation in its allocation strategy.

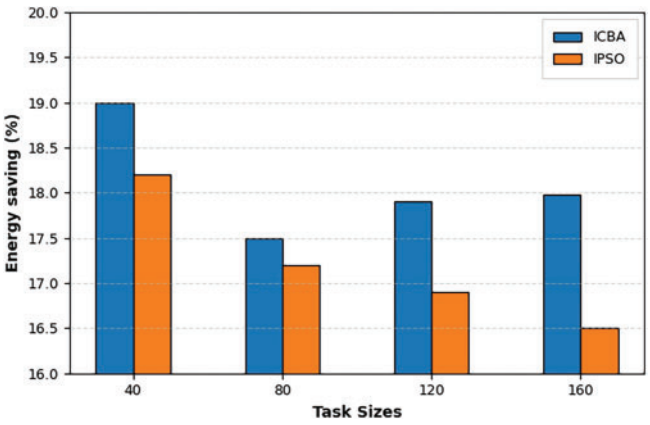

Figure 7: Energy saving for IPSO with ICBA

6.6 Overall

The results clearly show that IPSO outperforms others in balancing energy consumption and execution time, two important factors involved in edge computing. During this process, IPSO utilizes a multi-objective optimization framework where both objectives bear equal weights to ensure that energy efficiency does not suffer at the cost of task execution time and *vice versa*. This makes it a strong solution when both objectives bear equal importance, such as edge devices-based applications where energy consumption is highly relevant but at the same time, tasks are to be performed with minimal time consumption. Additionally, IPSO maintains a stable balance between energy consumption and execution time as task sizes increase. This is achieved through its adaptive parameter adjustment, which enhances resource utilization and prevents the overloading of specific edge nodes. Additionally, the Global Optimal Position Mutation Strategy GOPMS ensures diversity in the solution space, allowing IPSO to find near-optimal solutions under varying task loads.

7 Conclusion

The paper proposes an Improved Particle Swarm Optimization algorithm (IPSO) that can solve two very important challenges of task allocation in an edge computing environment: low resources, unpredictable workloads, and energy efficiency. The proposed IPSO algorithm effectively balances the two conflicting objectives: minimization of energy consumption and reduction of the execution time of the tasks critical in ensuring efficiency and reliability within edge computing systems. By incorporating the salient features of global optimum position mutation and dynamic inertia weight adaptation, the proposed method outperformed the other conventional algorithms, AEDPSO and ICBA. Experimental results disclose that for maximum task size, the proposed method IPSO reduced energy consumption by 31.58% and task execution time by 17.1%, confirming its capability to handle resource-intensive applications in edge computing. These results underline the scalability and performance of IPSO in handling the dynamic nature at the edge due to fluctuating resource availability and variable workload demand.

Other objectives than energy consumption and execution time will be considered in future work. Including task prioritization, latency constraints, and data privacy requirements, for example, into this algorithm would make it more versatile for an edge computing scenario such as real-time processing in healthcare or autonomous systems.

Acknowledgement: This study is supported by the University 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.

Funding Statement: This study was funded by University Putra Malaysia and the Ministry of Higher Education Malaysia.

Author Contributions: 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, Danlami Gabi: manuscript reviewing and editing. All authors reviewed the results and approved the final version of the manuscript.

Availability of Data and Materials: The code used to support the findings of this study can be obtained from the corresponding author upon request.

Ethics Approval: Not applicable.

Conflicts of Interest: The authors declare no conflicts of interest to report regarding the present study.

References

1. Elbamby MS, Perfecto C, Liu C-F, Park J, Samarakoon S, Chen X, et al. Wireless edge computing with latency and reliability guarantees. Proc IEEE. 2019;107(8):1717–37. doi:10.1109/jproc.2019.2917084.
2. Sakhdari J, Zolfaghari B, Izadpanah S. Edge computing: a systematic mapping study. Concurr Comput Pract Exp. 2023;35(22):e7741. doi:10.1002/cpe.7741.
3. Bukhsh M, Abdullah S, Bajwa IS. A decentralized edge computing latency-aware task management method with high availability for IoT applications. IEEE Access. 2021;9:138994–9008. doi:10.1109/access.2021.3116717.
4. Avan A, Azim A, Mahmoud QH. A state-of-the-art review of task scheduling for edge computing: a delay-sensitive application perspective. Electronics. 2023;12(12):2599. doi:10.3390/electronics12122599.
5. Nencioni G. 5G multi-access edge computing: a survey on security, dependability, and performance. IEEE Access. 2023;11:63496–533. doi:10.1109/access.2023.3288334.
6. Avcil MN, Soyturk M, Kantarci B. Fair and efficient resource allocation via vehicle-edge cooperation in 5G-V2X networks. Veh Commun. 2024;48(2):100773. doi:10.1016/j.vehcom.2024.100773.
7. Yu Z, Gong Y, Gong S, Guo Y. Joint task offloading and resource allocation in UAV-enabled mobile edge computing. IEEE Internet Things J. 2020;7(4):3147–59. doi:10.1109/jiot.2020.2965898.
8. Wang P, Member S, Yao C, Zheng Z, Member S. Joint task assignment, transmission, and computing resource allocation in multilayer mobile edge computing systems. IEEE Internet Things J. 2019;6(2):2872–84. doi:10.1109/jiot.2018.2876198.
9. Zhai L, Zhao P, Xue K, Li Y, Cheng C. Task offloading and multi-cache placement in multi-access mobile edge computing. Comput Netw. 2025;258(9):111030. doi:10.1016/j.comnet.2024.111030.
10. Li X, Bi S, Quan Z, Wang H. Online cognitive data sensing and processing optimization in energy-harvesting edge computing systems. IEEE Trans Wirel Commun. 2022;21(8):6611–26. doi:10.1109/twc.2022.3151509.
11. Hu B, Shi Y, Cao Z. Adaptive energy-minimized scheduling of real-time applications in vehicular edge computing. IEEE Trans Ind Inform. 2023;19(5):6895–906. doi:10.1109/tii.2022.3207754.
12. Kocot B, Czarnul P, Proficz J. Energy-aware scheduling for high-performance computing systems: a survey. Energies. 2023;16(2):1–28. doi:10.3390/en16020890.

13. Sefati SS, Haq AU, Nidhi, Craciunescu R, Halunga S, Mihovska A, et al. A comprehensive survey on resource management in 6G network based on internet of things. IEEE Access. 2024;12(1):113741–84. doi:10.1109/access.2024.3444313.
14. Xia Q, Ye W, Tao Z, Wu J, Li Q. A survey of federated learning for edge computing: research problems and solutions. High-Confid Comput. 2021;1(1):100008. doi:10.1016/j.hcc.2021.100008.
15. Huang L, Qin J, Zhou Y, Zhu F, Liu L, Shao L. Normalization techniques in training DNNs: methodology, analysis and application. IEEE Trans Pattern Anal Mach Intell. 2023;45(8):10173–96. doi:10.1109/tpami.2023.3250241.
16. Kuru K. Planning the future of smart cities with swarms of fully autonomous unmanned aerial vehicles using a novel framework. IEEE Access. 2021;9:6571–95. doi:10.1109/access.2020.3049094.
17. Han L, Zhu S, Zhao H, He Y. An enhanced whale optimization algorithm for task scheduling in edge computing environments. Front Big Data. 2024;7(56):39–44. doi:10.3389/fdata.2024.1422546.
18. Latip R, Aminu J, Mohd Z, Shafinah H, Danlami K. Metaheuristic task offloading approaches for minimization of energy consumption on edge computing: a systematic review. Discov Internet Things. 2024;4(1):1048. doi:10.1007/s43926-024-00089-y.
19. Alali S, Assalem A. Metaheuristics method for computation offloading in mobile edge computing: survey. J Adv Res Appl Sci Eng Technol. 2024;36(1):43–73. doi:10.37934/araset.36.1.4373.
20. Maia A, Ghamri-doudane Y, Vieira D, De MF, Maia A, Ghamri-doudane Y, et al. An improved multi-objective genetic algorithm with heuristic initialization for service placement and load distribution in edge computing. Comput Netw. 2021;194(4):108146. doi:10.1016/j.comnet.2021.108146.
21. Santoyo-gonzález A, Cervelló-pastor C. Network-aware placement optimization for edge computing infrastructure under 5G. IEEE Access. 2020;8:56015–28. doi:10.1109/access.2020.2982241.
22. Gad AG. Particle swarm optimization algorithm and its applications: a systematic review. Arch Comput Methods Eng. 2022;29(5):2531–61. doi:10.1007/s11831-021-09694-4.
23. Xing H, Liu L, Xu J, Nallanathan A. Joint task assignment and resource allocation for D2D-enabled mobile-edge computing. IEEE Trans Commun. 2019;67(6):4193–207. doi:10.1109/tcomm.2019.2903088.
24. Wen J, Yang J, Wang T, Li Y, Lv Z. Energy-efficient task allocation for reliable parallel computation of cluster-based wireless sensor network in edge computing. Digit Commun Netw. 2023;9(2):473–82. doi:10.1016/j.dcan.2022.06.014.
25. Mao Y, Zhang J, Letaief KB. Joint task offloading scheduling and transmit power allocation for mobile-edge computing systems. In: Proceedings of the 2017 IEEE Wireless Communications and Networking Conference (WCNC); 2017 Mar 19–22.
26. Yan J, Bi S, Zhang YJ, Tao M. Optimal task offloading and resource allocation in mobile-edge computing with inter-user task dependency. IEEE Trans Wirel Commun. 2019;19(1):235–50. doi:10.1109/glocom.2018.8647523.
27. Wang F, Xu J, Cui S. Optimal energy allocation and task offloading policy for wireless powered mobile edge computing systems. IEEE Trans Wirel Commun. 2020;19(4):2443–59. doi:10.1109/tcomm.2020.3011990.
28. Gu L, Cai J, Zeng D, Zhang Y, Jin H, Dai W. Energy efficient task allocation and energy scheduling in green energy powered edge computing. Futur Gener Comput Syst. 2019;95(4):89–99. doi:10.1016/j.future.2018.12.062.
29. Wang Q, Shao S, Guo S, Qiu X, Wang Z. Task allocation mechanism of power internet of things based on cooperative edge computing. IEEE Access. 2020;8:158488–501. doi:10.1109/access.2020.3020233.
30. Liu J, Liu X. Energy-efficient allocation for multiple tasks in mobile edge computing. J Cloud Comput. 2022;11(1):1–14. doi:10.1186/s13677-022-00342-1.
31. Li S, Zhang N, Jiang R, Zhou Z, Zheng F, Yang G. Joint task offloading and resource allocation in mobile edge computing with energy harvesting. J Cloud Comput. 2022;11(1):17. doi:10.1186/s13677-022-00290-w.
32. Liu X, Liu J, Wu H. Energy-efficient task allocation of heterogeneous resources in mobile edge computing. IEEE Access. 2021;9:119700–11. doi:10.1109/access.2021.3108342.
33. Deng X, Li J, Liu E, Zhang H. Task allocation algorithm and optimization model on edge collaboration. J Syst Arch. 2020;110:1–16.
34. Wang F, Xing H, Xu J. Real-time resource allocation for wireless powered multiuser mobile edge computing with energy and task causality. IEEE Trans Commun. 2020;68(11):7140–55. doi:10.1109/icc.2019.8761143.

35. Sarhani M, Voß S, Jovanovic R. Initialization of metaheuristics: comprehensive review, critical analysis, and research directions. Int Trans Oper Res. 2023;30(6):3361–97. doi:10.1111/itor.13237.
36. Bozkaya E, Erel-Özçevik M, Bilen T, Özçevik Y. Proof of evaluation-based energy and delay aware computation offloading for digital twin edge network. Ad Hoc Netw. 2023;149(4):103254. doi:10.1016/j.adhoc.2023.103254.
37. Golpayegani F, Chen N, Afraz N, Gyamfi E, Malekjafarian A, Schäfer D, et al. Adaptation in edge computing: a review on design principles and research challenges. ACM Trans Auton Adapt Syst. 2024;19(3):1–43. doi:10.1145/3664200.
38. Varna FT, Husbands P. Two new bio-inspired particle swarm optimisation algorithms for single-objective continuous variable problems based on eavesdropping and altruistic animal behaviours. Biomimetics. 2024;9(9):538. doi:10.3390/biomimetics9090538.
39. Rashed NA, Ali YH, Rashid TA. Advancements in optimization: critical analysis of evolutionary, swarm, and behavior-based algorithms. Algorithms. 2024;17(9):416. doi:10.3390/a17090416.
40. Lahmar I, Zaier A, Yahia M, Boaullegue R. A novel improved binary harris hawks optimization for high dimensionality feature selection. Pattern Recognit Lett. 2023;171(6):170–6. doi:10.1016/j.patrec.2023.05.007.
41. Niu Y, Yan X, Zeng W, Wang Y, Niu Y. Multi-objective sand cat swarm optimization based on adaptive clustering for solving multimodal multi-objective optimization problems. Math Comput Simul. 2025;227(4):391–404. doi:10.1016/j.matcom.2024.08.022.
42. Kiani AT, Nadeem MF, Ahmed A, Khan IA, Alkhammash HI, Sajjad IA, et al. An improved particle swarm optimization with chaotic inertia weight and acceleration coefficients for optimal extraction of PV models parameters. Energies. 2021;14(11):2980. doi:10.3390/en14112980.
43. Norouzzadeh MS, Ahmadzadeh MR, Palhang M. Plowing PSO: a novel approach to effectively initializing particle swarm optimization. In: Proceedings of the 2010 3rd International Conference on Computer Science and Information Technology; 2010 Jul 9–11; Chengdu, China. p. 705–9.
44. Jian C, Chen J, Ping J, Zhang M. An improved chaotic bat swarm scheduling learning model on edge computing. IEEE Access. 2019;7:58602–10. doi:10.1109/access.2019.2914261.
45. Lu Y, Liu L, Gu J, Panneerselvam J, Yuan B. EA-DFPSO: an intelligent energy-efficient scheduling algorithm for mobile edge networks. Digit Commun Netw. 2022;8(3):237–46. doi:10.1016/j.dcan.2021.09.011.