Latency Aware Task Allocation in Heterogeneous Multi-Core System using Whale Optimization Algorithm

Dr. C.G. Igiri., Cookey, E.E., Tenalo, B.T., Akani, Kingston

Department Of Computer Science, Rivers State University

DOI: https://doi.org/10.51584/IJRIAS.2026.11050146

Received: 04 May 2026; Accepted: 09 May 2026; Published: 01 June 2026

ABSTRACT

Heterogeneous multi-core systems are increasingly used in latency-sensitive computing environments because they combine different processing units such as high-performance CPUs, low-power cores, GPUs, NPUs, and DSPs. However, the diversity of these processing units makes task allocation difficult, especially when execution time, communication delay, processor availability, energy consumption, and workload dependencies must be considered together. This study develops a latency-aware task allocation framework for heterogeneous multi-core systems using the Whale Optimization Algorithm (WOA). The objective is to formulate task scheduling as a discrete task-to-core mapping problem and evaluate the effectiveness of WOA under independent, DAG-based, real-time, and energy-aware workload conditions. The proposed method adapts the continuous WOA into a discrete integer-encoded scheduling model, where each whale represents a complete task-to-processor assignment. A latency-based fitness function was used for WOA-Latency, while an extended weighted objective was applied for WOA-EnergyAware to examine latency-energy trade-offs. The experimental evaluation compared the proposed WOA variants with Random, Round-Robin, MET, MCT, Min-Min, Max-Min, HEFT, PEFT, GA, PSO, GWO, ACO, and a WOA-MCT hybrid scheduler. The evaluation included repeated simulation runs, statistical reporting, convergence analysis, scalability testing, deadline miss analysis, and parameter sensitivity assessment. The results show that WOA-Latency outperforms several naïve and metaheuristic baselines in some workload settings, particularly Random, Round-Robin, GA, PSO, and GWO. However, deterministic heuristics such as MCT and Min-Min performed better for independent-task workloads, while HEFT and PEFT achieved stronger results for DAG-based scheduling. The energy-aware WOA variant demonstrated the framework's potential for multi-objective optimisation, although with some latency trade-off. Overall, the study concludes that WOA is a flexible and extensible optimisation framework for heterogeneous task allocation, especially in complex, nonlinear, energy-aware, and multi-objective scheduling scenarios.

Keywords: Heterogeneous multi-core systems; task scheduling; Whale Optimization Algorithm; latency-aware allocation; energy-aware scheduling.

INTRODUCTION

The developments in modern computing systems have been largely driven by the need to support applications that need a lot of data and are sensitive to latency, such as artificial intelligence, autonomous systems, real-time analytics, and edge computing. As transistor scaling nears its physical constraints, advancements through clock frequency scaling have significantly reduced, prompting a transition towards parallel and heterogeneous computing architectures [10]. As a result, Heterogeneous Multi-core Systems (HMCs) have become a key architectural model for attaining high performance while maintaining efficient energy usage. These architectures integrate processing elements with different computing abilities, such as high-performance cores, GPUs, low-power cores, and specialized accelerators, in a single platform. The diversity in architecture lets systems match the characteristics of the workload with the right computational resources, which improves the system performance and efficiency [3], [9].

In recent years, task scheduling in heterogeneous multi-core systems has gotten a lot of attention because of the rapid evolution in asymmetric processor architectures used in modern computing platforms such as edge

ISSN No. 2454-6194 | DOI: 10.51584/IJRIAS |Volume XI Issue V May 2026

computing nodes, mobile devices, and high-performance embedded systems. In homogeneous architectures, all the processors have the same computing power. In heterogeneous systems, on the other hand, the processing elements have different power consumption profiles, performance characteristics, and memory hierarchies. This variety makes scheduling decisions more complex as tasks need to be allocated to processing units that can execute them efficiently while minimizing performance overheads such as resource utilization and latency.

Despite the fact that these architectures offer these advantages, they also introduce significant challenges in scheduling. In heterogeneous systems, the execution time for a task can differ depending on the processing unit it is allocated to. In addition, if tasks are not properly assigned, communication overhead between cores and workload imbalance can further reduce the performance of the system. Latency, which is the time between submitting a task and its completion, has become an important measure of performance in modern computing environments. This is particularly important for edge computing and real-time decision systems, where timely response is important [13], [19].

Task scheduling in heterogeneous systems is therefore a complex optimization problem because the number of possible task-to-core assignments increases exponentially with the number of tasks and processors, making the problem computationally intractable for exact methods [2]. Existing heuristic scheduling approaches often focus primarily on makespan reduction without explicitly modeling latency in heterogeneous architectures.

This study addresses the problem of developing an efficient scheduling strategy capable of minimizing system latency in heterogeneous multi-core environments. Specifically, the work formulates task scheduling as a mathematical optimization problem where tasks must be assigned to heterogeneous processing units while accounting for execution time variability and architectural constraints. The proposed solution adopts the Whale Optimization Algorithm (WOA), a nature-inspired metaheuristic algorithm known for its effective balance between exploration and exploitation during search processes [11]. By integrating a formal mathematical model with WOA-based optimization, the proposed framework aims to improve scheduling efficiency and reduce latency in heterogeneous computing systems.

This paper makes four main contributions. First, it formulates heterogeneous multi-core task allocation as a latency-aware combinatorial optimization problem that accounts for computation delay, communication delay and queuing delay. Second, it adapts the continuous Whale Optimization Algorithm into a discrete task-to-core mapping procedure suitable for heterogeneous processors. Third, it extends the latency objective into a latency-energy formulation, allowing task allocation to reflect both response-time and power-efficiency requirements. Fourth, it provides a comparative simulation against Random, Round-Robin, MET and MCT schedulers, together with convergence, parameter-sensitivity, scalability and runtime-overhead analyses. The results are interpreted conservatively: WOA is not claimed to dominate greedy heuristics for simple independent workloads, but is positioned as an extensible metaheuristic for complex, multi-objective and dependency-aware scheduling scenarios.

The remainder of this paper is organized as follows. Section 2 reviews related work on heterogeneous multi-core scheduling and latency minimization techniques. Section 3 presents the system model and mathematical formulation of the scheduling problem. Section 4 describes the proposed Whale Optimization Algorithm-based scheduling model. Section 5 presents the experimental setup and performance evaluation results. Finally, Section 6 concludes the study and outlines potential directions for future research.

Related Work

Initial works on heterogeneous scheduling include Heterogeneous Earliest Finish Time (HEFT), which was developed by Topcuoglu, Hariri, and Wu in 2002. The HEFT algorithm is still one of the most commonly used heuristics for solving heterogeneous problems. It ranks tasks using upward task selection and schedules them on processors based on the earliest expected finish time. Although the heuristic significantly improved the scheduling process compared to previous static techniques, further studies indicate that it may perform poorly in large heterogeneous systems or in cases involving complex execution models where the task execution times are highly variable on different processors.

ISSN No. 2454-6194 | DOI: 10.51584/IJRIAS |Volume XI Issue V May 2026

In order to overcome some weaknesses associated with HEFT, various advanced heuristic methods have been suggested. Specifically, Arabnejad and Barbosa (2014) presented the Predict Earliest Finish Time (PEFT) method, which enhances the precision of scheduling through an introduction of an optimistic cost matrix used to predict task execution times in heterogeneous settings. According to experimental results, PEFT outperforms HEFT in terms of scheduling effectiveness in many cases, especially when large task graphs are considered. Nevertheless, neither PEFT nor HEFT pay explicit attention to latency minimization; instead, the emphasis is placed on achieving the shortest makespan. Considering the growing complexity of heterogeneous computing architectures, contemporary studies highlight the necessity to develop adaptive scheduling techniques that can leverage heterogeneity during runtime. For instance, Mittal (2016) conducted a detailed review of asymmetric multicore processors, pointing out that traditional scheduling techniques developed for homogeneous architectures do not exploit heterogeneity effectively.

Additionally, to performance-related issues, energy aware scheduling has been gaining traction recently, due to increasing popularity of heterogeneous architectures operating under power constraints in the context of mobile devices and edge computing systems. There are multiple studies focusing on multi-objective optimizations aiming to reduce execution time and energy usage simultaneously. For example, Sahoo, Mohanty, and Patra (2022) studied various multi-objective approaches to scheduling of tasks in heterogeneous systems, and found out that the use of evolutionary algorithms can significantly improve performance and efficiency in this case. Moreover, there were multiple recent publications in Future Generation Computer Systems and Journal of Systems Architecture suggesting various hybrid algorithms which could balance workloads in order to optimize energy consumption. However, despite being promising and beneficial for multi-objective scheduling studies, many of these works treat latency as an auxiliary measure while optimizing energy consumption and performance simultaneously.

Latency minimization has become an important requirement in modern computational systems due to the rise of real-time data processing, applications of artificial intelligence, and interactive cloud services. In such conditions, minimizing the response time of a task is crucial in maintaining the quality of the system's operation. Conventional techniques of latency reduction involve priority-based scheduling, earliest deadline first approach, and dynamic load balancing. Nevertheless, these methods rely on homogenous processing systems, thereby failing to solve the problem of latency reduction posed by heterogeneity. In recent times, several research studies have emphasized the necessity of developing latency-aware scheduling approaches in light of heterogeneous processing architectures. For instance, Zhao et al. (2020) studied latency-aware scheduling methods in heterogeneous computational architectures and found that considering response time during the scheduling process significantly improved the effectiveness of the technique. Also, edge computing studies have highlighted the necessity of task scheduling based on the concept of latency because computations need to be conducted close to the source data in order to minimize latency arising from network transmissions (Shi et al., 2016). While these approaches focus on distributed computing frameworks, they have implications for heterogeneous multi-core architectures as well.

Due to the fact that heterogeneous scheduling is computationally complex and is generally considered an NP-hard problem, metaheuristic optimization approaches are widely used for solving such problems. The use of metaheuristics allows one to efficiently explore huge solution spaces to find the optimal schedule without exploring all solutions. Among the types of metaheuristics, evolutionary algorithms, which include Genetic Algorithms (GA), are widely used for heterogeneous scheduling due to the ability to efficiently explore solution spaces globally and handle combinatorial optimization problems. In this approach, possible schedules are represented by a chromosome encoding task to core mappings, which are then modified using genetic operators like crossover and mutation. Although GA-based approaches are effective, they can suffer from premature convergence and heavy computation loads in case of a huge solution space (Sahoo et al., 2022).

There are some cases where swarm intelligence algorithms have shown success in scheduling. One example is the use of Particle Swarm Optimization (PSO), which is based on the behavior witnessed in birds when they fly in groups, to produce schedules quickly with good efficiency. According to Abdullahi et al. (2020), the use of swarm intelligence for scheduling has proven useful in tackling complicated task scheduling problems in distributed computing environments. On the other hand, ant colony optimization (ACO) has also been used to

ISSN No. 2454-6194 | DOI: 10.51584/IJRIAS |Volume XI Issue V May 2026

tackle heterogeneous task scheduling by using scheduling processes as probabilistic path formation processes using pheromone trails.

In more recent literature, various sophisticated nature-mimicking optimization methods have been proposed for improving both exploration and exploitation capabilities for solving optimization problems. Methods like GWO (Grey Wolf Optimization), FA (Firefly Algorithm), and DE (Differential Evolution) have been used for solving scheduling problems in heterogeneous computing systems and have achieved good performance. These methods try to maintain a balance between exploration and exploitation while searching for solutions. However, in most research works using such methods, the focus is either on minimizing makespan or minimizing energy consumption, and little effort has been made towards minimizing latency.

The Whale Optimization Algorithm is one of those metaheuristic algorithms that have attracted lots of attention due to their simplicity and effectiveness. Proposed by Mirjalili and Lewis (2016), WOA is based on the behavior of humpback whales and especially their approach towards hunting through bubble-net attack. Specifically, two behaviors of these marine mammals have been incorporated into WOA: encircling prey and bubble-net attack. These behaviors have made it possible for WOA to achieve both exploration and exploitation. Thanks to its simplicity and fast convergence, WOA has found applications in various engineering problems.

In recent times, research has employed the Whale Optimization Algorithm (WOA) to solve scheduling problems in distributed and cloud computing settings. Research findings suggest that WOA-based schedulers are able to optimize load balancing and improve the overall execution time when compared to the traditional heuristic methods. However, in comparative analysis, it was found that WOA is able to perform better in some situations, especially in non-linear optimization scenarios where the search space is very complicated. Nonetheless, most research efforts on WOA for scheduling problems tend to focus on cloud computing and grid computing architectures rather than in-chip heterogeneous multicore processor systems. Additionally, many research papers do not present the optimization problem, such as minimizing latency, as an objective function but consider it as one among several optimization parameters.

Consequently, notwithstanding the considerable advancements achieved in heterogeneous task scheduling research, numerous deficiencies persist in the literature. To begin with, there aren't many studies that look specifically at how to reduce latency in heterogeneous multi-core architectures. Second, a lot of current methods use heuristic scheduling methods that might not work well as systems get more complicated. Third, while metaheuristic algorithms like WOA have shown that they can optimise well, they can't be used much for scheduling models that focus on latency for heterogeneous multi-core processors. These limitations motivate the development of a mathematically robust scheduling framework that explicitly models latency while leveraging the global optimization capabilities of the Whale Optimization Algorithm.
This study addresses the existing gap by conceptualising the heterogeneous scheduling problem as a latency mi nimisation model and modifying WOA to effectively identify optimal task-to-
core mappings within heterogeneous multi-core systems.

METHODOLOGY

This section presents the methods used to model and solve the scheduling problem of minimising latency in heterogeneous multi-core systems. The system architecture, task representation, and latency model are all clearly defined. Then, the scheduling optimisation problem is put into mathematical form.

Assumptions

To guarantee analytical tractability and consistent modelling of the heterogeneous scheduling environment, the subsequent assumptions are established:

i. The application workload is represented as a Directed Acyclic Graph (DAG) where nodes denote tasks and edges denote precedence constraints.

ISSN No. 2454-6194 | DOI: 10.51584/IJRIAS |Volume XI Issue V May 2026

ii. The heterogeneous computing system contains m processors with different computational capabilities.

iii. Execution time of a task varies depending on the processor on which it executes.

iv. Tasks are non-preemptive, meaning once a task begins execution on a processor it cannot be interrupted.

v. Communication cost occurs only when dependent tasks are assigned to different processors.

vi. Communication latency between processors is deterministic and known beforehand.

vii. All processors are available at time t = 0.

viii. The scheduling process is static, meaning all tasks are known before execution begins.

These assumptions are consistent with commonly used heterogeneous scheduling frameworks in the literature (Arabnejad & Barbosa, 2014).

Architecture of the Proposed System

The architecture of the system is shown in the figure. 1 below. Consider a heterogeneous multi-core system consisting of a set of processors:

$P = \{P_1, P_2, P_3, \dots, P_m\}$ 3.1

Where m represents the total number of heterogeneous processors. Each processor P_j is characterized by:

- Processing speed S_j (e.g., instructions per cycle),
- Execution capability,
- Communication latency with other processors

Because the processors are heterogeneous, the execution time of a task differs depending on the processor executing it.

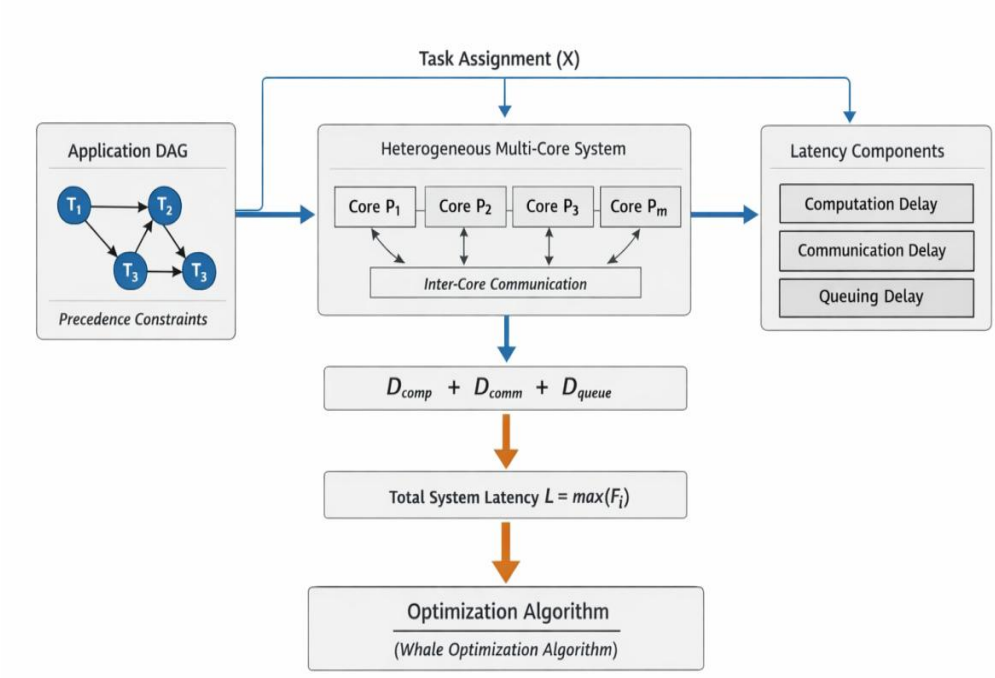

Figure 1. Architecture of the system

ISSN No. 2454-6194 | DOI: 10.51584/IJRIAS |Volume XI Issue V May 2026

The proposed system's architecture is a heterogeneous multi-core scheduling framework that uses the Whale Optimisation Algorithm (WOA) to intelligently assign tasks in order to reduce latency. The system combines task modelling, latency calculation, and metaheuristic optimisation into a single pipeline that assigns application tasks to different types of processing units.

The system gets an application workload that looks like a Directed Acyclic Graph (DAG) at the input stage. In this diagram, each node stands for a task that needs to be done, and the edges show the order in which tasks need to be done and how data needs to be shared between tasks. This structure guarantees that tasks are executed in the proper sequence and facilitates precise modelling of inter-task communication latencies, in accordance with established scheduling frameworks (Topcuoglu et al., 2002).

The Task Analyser processes the DAG and extracts key parameters such as the computational workload, task dependencies, and data transfer needs. The parameters are then used to make the Execution Time Matrix (ETM), which shows how long each task will take on each available processor. Because different processors have different computational capabilities, execution times will be different on each one (Mittal, 2016).

The system architecture also includes a Heterogeneous Core Pool, consisting of multiple processing units such as high-performance CPUs, low-power cores, GPUs, NPUs, and DSPs. Each core is characterized by its processing speed, power consumption, and communication latency with other cores. This diversity allows the system to exploit specialization but also increases scheduling complexity.

The Latency Modelling Unit is an important part of the architecture. It figures out the total latency of task execution by adding up three important factors: computational delay, communication delay, and queuing delay. The ETM gives us the computational delay, the data dependencies and inter-core communication costs give us the communication delay, and the queuing delay takes into account the time spent waiting for a processor to become available. This complete latency formulation lets the system make better scheduling decisions than traditional approaches that only look at makespan.

The WOA-Based Scheduler is the main part of the decision-making process. It uses the Whale Optimisation Algorithm to find the best way to assign tasks to cores. The fitness function looks at each candidate solution and rates it based on the total system latency. Each candidate solution is a possible way to map tasks to processors. The algorithm keeps improving solutions through exploration and exploitation mechanisms until they all come together (Mirjalili & Lewis, 2016).

Lastly, the Output Module comes up with the best scheduling solution, which includes task assignments, the order in which tasks should be done, and performance metrics like makespan and energy use. You can use this output to evaluate performance or put it to use in real heterogeneous computing environments. The proposed architecture offers a scalable and adaptable framework for latency-aware scheduling in heterogeneous multi-core systems through the integration of mathematical modelling and metaheuristic optimisation.

Task Representation

The application is modeled as a Directed Acyclic Graph:

$G = (T, E)$……3.2

Where:

- $T = \{T_1, T_2, \dots, T_n\} is\ the\ set\ of\ task,$ ……3.3
- $E\ represents\ precedence\ dependence\ between\ tasks.$

An edge $(T_i, T_k) \in E\ indicates\ that\ task\ T_k\ cannot\ start\ until\ T_i$ completes and the required data is transmitted.

ISSN No. 2454-6194 | DOI: 10.51584/IJRIAS |Volume XI Issue V May 2026

Task Assignment Variable

Task allocation is defined using a binary variable:

$$x_{ij} = \begin{cases} 1 & if\ task\ T_i\ is\ assigned\ to\ processor\ P_j \\ 0 & otherwise \end{cases} \quad \text{.....3.4}$$

Each task must be assigned to exactly one processor:

$$\sum_{j=1}^{m} x_{ij} = 1 \ \vee i \in \{1,2, \dots, n\} \quad \text{.....3.5}$$

Latency Mathematical Model

In heterogeneous multi-core systems, total latency is composed of three major components:

1. Computational Delay
2. Communication Delay
3. Queuing (Waiting) Delay

Thus, total latency for task T_i is defined as:

$$L_i = D_i^{comp} + D_i^{comm} + D_i^{queue} \quad \text{.....3.6}$$

1. Computational Delay

Computational delay represents the execution time of a task on a processor.

Let

- $w_i = computational\ workload\ of\ task\ T_i$
- $s_j = processing\ speed\ of\ processor\ P_j$

Then the computation time of the task T_i on processor P_j is:

$$C_{ij} = \frac{w_i}{s_j} \quad \text{.....3.7}$$

The computation delay for the task T_i considering the assignment is therefore:

$$D_i^{comp} = \sum_{j=1}^{m} x_{ij} X_{ij} \quad \text{.....3.8}$$

2. Communication Delay

Communication delay occurs when dependent tasks are executed on different processors.

Let

$$d_{ki} = data\ size\ transferred\ from\ task\ T_k\ to\ task\ T_i$$

$$\delta_{jl} = communication\ latency\ between\ processor\ P_j\ and\ P_l$$

Then communication delay between tasks is defined as:

$D_i^{comm} = \sum_{j=1}^{m} \sum_{l=1}^{m} x_{kj} x_{il} . d_{ki} . \delta_{jl}$……3.9

If both tasks are executed on the same processor:

$\delta_{jl} = 0$……3.10

Thus, the communication cost becomes zero.

3. Queuing (Waiting) Delay

Queuing delay occurs when multiple tasks are assigned to the same processor. The queuing delay in this work is modeled using a deterministic single-server queue, commonly denoted as:

D/D/1

Where:

- First D → Deterministic task arrival (static scheduling, all tasks known beforehand)
- Second D → Deterministic service time (execution time C_{ij} is known)
- 1 → Single server (each processor executes one task at a time)

Each processor P_j is therefore modeled as an independent D/D/1 queue, where tasks assigned to that processor are executed sequentially.

Queuing Delay Formulation

- $S_i = start\ time\ of\ task\ T_i$
- $F_i = finish\ time\ of\ task\ T_i$

Then

If tasks T_a and T_b are scheduled on the same processor:

$S_b \geq F_a$……3.11

The queuing delay, therefore, depends on the waiting time before processor availability.

Total System Latency

The total system latency is defined as the completion time of the last finishing task:

$L = \max_{i \in \{1, \ldots, n\}} F_i$……3.12

F_i represents the finish time (completion time) of the task T_i

This represents the overall response time of the system workload.

Optimization Problem Formulation

The scheduling objective is to minimize total latency. $\min L$.

Subject to the following constraints:

ISSN No. 2454-6194 | DOI: 10.51584/IJRIAS |Volume XI Issue V May 2026

Task Assignment Constraint

$\sum_{j=1}^{m} x_{ij} = 1$……3.13

Binary Constraint

$x_{ij} \in \{0,1\}$…….3.24

Precedence Constraint

For every dependency (T_k, T_i)

$S_i \geq F_k + D_{ki}^{comp}$…….3.15

Processor Capacity Constraint

Tasks assigned to the same processor must execute sequentially:

$S_b \geq F_a$…….3.16

Objective Function

The objective of the scheduling model is to minimize the overall system latency:

$\min_X L = \min_X (\max F_i)$…….3.17

Were

- $X = [x_{ij}]$ represents the task-to-processor assignment matrix
- $\max F_i = \ time\ when\ the\ last\ task\ finishes$

The problem becomes NP-hard because the search space grows exponentially with the number of tasks and processors (Arabnejad & Barbosa, 2014). Consequently, metaheuristic algorithms like the Whale Optimisation Algorithm (WOA) are utilised to achieve near-optimal solutions.

Discrete Encoding of the Whale Optimization Algorithm for Task Allocation

While the initial WOA was designed for continuous optimization problems where each search agent constantly adjusts its position in a continuous space, task allocation on heterogeneous multi-core architectures is a discrete optimization problem, in which each task should be mapped onto one of the available processors. Therefore, the standard WOA approach for updating positions had to be modified to represent a discrete version of task-core assignments.

In this study, each whale represents one complete scheduling solution. A whale position is encoded as an integer vector:

$X_i = [x_{i1,} x_{i2}, x_{i3}, \dots, x_{in}]$……..3.18

where X_i represents the position of the i^{th}whale, nnn is the total number of tasks, and each element x_{ij} denotes the processor assigned to task T_i. Since the system contains m heterogeneous processors, each value in the vector is restricted to:

$x_{ij} \in \{0, 1, 2, \dots, m-1\}$…….3,19

ISSN No. 2454-6194 | DOI: 10.51584/IJRIAS |Volume XI Issue V May 2026

Computational Complexity Analysis

The computational cost of the proposed Whale Optimization Algorithm-based scheduling scheme mainly depends on four variables: the number of whales (N_w), maximum number of iterations (I), number of tasks(n), and number of processors(m).

A single whale represents a full-fledged schedule solution consisting of n tasks assignment. At each iteration, the position of each whale is updated and evaluated. The computational complexity of the position update operation is: $O(n)$.

Since each task assignment in the whale vector needs to be updated and discretized. The evaluation of the fitness function involves calculating the amount of work assigned to each processor and determining the time taken by the processors to finish. This can be easily done for independent tasks by iterating through the task vector and summing up the execution times for the assigned processor. Hence, the complexity of the fitness evaluation function is: $O(n)$.

For each whale. Since the fitness evaluation function is computed per whale, throughout all iterations, the total complexity of the proposed WOA scheduling algorithm for independent-task scheduling is: $O(I \times N_w \times n)$

It is clear from above that the complexity increases linearly with an increase in the number of iterations, number of whales, and number of tasks.

In DAG scheduling, more computations are needed to take into account the precedence constraint and inter-task communications between the dependent tasks. Let e represent the number of edges in the DAG. Then, the complexity of the fitness evaluation function becomes approximately: $O(I \times N_w \times (n + e))$.

Where e represents the number of communication or precedence constraints between the tasks. The memory complexity of the algorithm relies mainly on the population matrix that holds $N_w$whales where each whale holds n task allocation information. The memory complexity is hence: $O(N_w \times n)$.

Moreover, there is need to account for the execution time matrix which will hold the execution time of all tasks on all processors, giving an extra memory complexity of: $O(n \times m)$. Hence, the total memory complexity becomes:

$O(N_w \times n + n \times m)$……3.20

From the above formulation, one can see that the Whale Optimization Algorithm (WOA) based scheduling algorithm has been developed based on a discrete optimization approach rather than applying the actual continuous Whale Optimization Algorithm to the problem at hand. Each whale is coded using a feasible task to processor assignment while each solution is evaluated based on latency and energy-related objective functions. The next section presents the experimental design used to test the scheduling algorithm.

Experimental Setup

This part of the document presents details of the experimental design used to evaluate the performance of the whale optimization algorithm-based scheduler. As opposed to the first experiment that focused on testing the scheduler under a workload consisting of 30 tasks, this part of the analysis focuses on testing the proposed approach under workloads including independent workloads, directed acyclic graph based dependent workloads, real-time DAG workloads, and scalable workloads.

This more extensive test is important considering that the problem of allocating heterogeneous tasks is affected not only by the processing time available but also other issues like task dependency, costs of inter-core communication, deadlines, energy use, and scheduling cost.

ISSN No. 2454-6194 | DOI: 10.51584/IJRIAS |Volume XI Issue V May 2026

The experimental design is realized using an experimental platform consisting of an experiment runner and several simulation modules. In this case, the experiment runner is used to execute simulations involving independent task workloads, DAG workloads, real-time DAG workloads, scalability experiments, parameter sensitivity studies, and significance tests. Also, the implementation automatically generates tables and figures in the form of CSV tables, LaTeX tables, and PDF plots. Each set of experiments has been executed for 30 times using a population size of 30 and 100 iterations for the heuristic methods.

Experimental Configuration

This proposed scheduling strategy was evaluated using synthetic workloads which can be categorized into independent and dependent. There are two kinds of workloads under consideration. First is the independent task workload in which there are no dependencies between tasks.

Independent workloads can help evaluate the performance of the load balancing and the assignment of tasks to cores without having dependency issues. The second kind is the DAG-based workloads in which nodes represent tasks while the directed edges represent the dependencies. DAG workloads are important as they include communication delay and precedence constraint aspects of the proposed latency model.

There are two types of workloads that can be produced by the generator, namely independent task workloads and DAG workloads. In independent workloads, execution time is created using heterogeneity factors over heterogeneous processors. For DAG workloads, DAGs are created by the generator together with execution times and a communication matrix depending on the communication to computation ratio. Deadlines can optionally be specified for real-time DAG workloads.

Table 1: Experimental Workload Configuration

Experiment group	Workload type	Number of tasks	Number of processors	Dependency structure	Main purpose
S1	Independent	30	8	None	Baseline comparison
S2	Independent	50	8	None	Medium independent-task scalability
S3	Independent	100	8	None	Larger independent-task scalability
D1	DAG-small	30	8	Low dependency density	Initial DAG validation
D2	DAG-medium	100	8	Moderate dependency density	Dependency-aware evaluation
D3	DAG-large	200	16	Higher dependency density	Large DAG evaluation
RT-DAG	Real-time DAG	100	12	Deadline-constrained DAG	Deadline miss analysis
S4	DAG scalability	200	16	DAG	Large-scale scalability
S5	DAG scalability	500	32	DAG	Stress-test configuration

Experiments on DAGs were run using numbers of tasks equal to 30, 100, and 200. In particular, the small DAG had 30 tasks scheduled on 8 processors, where edge density was 0.2, and communication-to-computation

ISSN No. 2454-6194 | DOI: 10.51584/IJRIAS |Volume XI Issue V May 2026

ratio was 0.5. The medium DAG had 100 tasks scheduled on 8 processors, where edge density was 0.35, and communication-to-computation ratio was 0.7.

The large DAG had 200 tasks scheduled on 16 processors, where edge density was 0.4, and communication-to-computation ratio was 1.0. Real-time DAG had 100 tasks scheduled on 12 processors within deadlines.

Compared Scheduling Algorithms

The suggested WOA-based schedulers were compared with classical algorithms, DAG-based List Schedulers, and other meta-heuristics to provide a comprehensive comparison that goes beyond the narrow context of Random, Round Robin, MET, and MCT.

Table 2: Compared Scheduling Algorithms

Algorithm	Category	Description
Random	Naïve baseline	Randomly assigns tasks to processors
Round-Robin	Naïve baseline	Cyclically assigns tasks to available processors
MET	Greedy heuristic	Assigns each task to the processor with the minimum execution time
MCT	Greedy heuristic	Assigns each task to the processor with the minimum completion time
Min-Min	Independent-task heuristic	Prioritises tasks with the smallest earliest completion time
Max-Min	Independent-task heuristic	Prioritises tasks with the largest earliest completion time
HEFT	DAG list scheduler	Uses upward rank and earliest finish time for heterogeneous DAG scheduling
PEFT	DAG list scheduler	Uses an optimistic cost table to improve heterogeneous DAG scheduling
GA	Metaheuristic	Uses genetic selection, crossover, and mutation
PSO	Metaheuristic	Uses particle position and velocity updates for schedule search
GWO	Metaheuristic	Uses grey wolf hierarchy-based search
ACO	Metaheuristic	Uses pheromone-guided probabilistic solution construction
WOA-Latency	Proposed metaheuristic	WOA with latency-focused objective
WOA-EnergyAware	Proposed multi-objective variant	WOA with weighted latency-energy objective
WOA-MCT-Hybrid	Hybrid proposed variant	WOA initialised using MCT assignment

The designed library includes scheduling algorithms such as WOA, HEFT, PEFT, MCT, MET, Round Robin, Random, Min-Min, Max-Min, GA, PSO, GWO, ACO, and Hybrid WOA-MCT. Therefore, it can be concluded that this comparison involves schedulers from deterministic and stochastic classes

ISSN No. 2454-6194 | DOI: 10.51584/IJRIAS |Volume XI Issue V May 2026

Evaluation Metrics

The algorithms were evaluated using the following metrics:

Table 3: Evaluation Metrics

Metric	Definition	Interpretation
Makespan	Completion time of the last task	Lower values indicate lower overall latency
Energy consumption	Sum of processor power multiplied by task execution time	Lower values indicate better energy efficiency
Runtime	Time required by the scheduler to generate a schedule	Lower values indicate lower scheduling overhead
Load balance	Distribution of workload across processors	Lower imbalance indicates better processor utilisation
Deadline miss ratio	Percentage of deadline-constrained tasks completed late	Lower values indicate better real-time suitability
Convergence behaviour	Fitness improvement across iterations	Shows stability and search efficiency
Statistical significance	Paired t-test or Wilcoxon signed-rank test	Indicates whether performance differences are statistically meaningful

In case of independent jobs, makespan was calculated as the sum of the time taken by each processor for executing the job, along with the determination of the largest completion time among all processors. In case of DAG tasks, makespan was calculated through topologically ordering the tasks based on processor availability, predecessor completion, and inter-processor delay. Energy was calculated by multiplying the task execution time with the processor power consumption.

Baseline Independent-Task Results

Experiment 1 considered 30 independent jobs running on 8 different processors. The value of Experiment 1 lies in the fact that it demonstrates the behaviour of the suggested WOA scheduling algorithm in the absence of job dependencies. However, this workload also favours deterministic algorithms like MCT and Min-Min since job assignment can be done efficiently through greedy completion time criteria.

Table 4: Makespan Results for 30 Independent Tasks

Algorithm	Mean makespan	Standard deviation	95% confidence interval
Random	398.00	84.49	[365.91, 430.09]
Round-Robin	298.19	42.20	[282.17, 314.22]
MET	246.18	49.89	[227.23, 265.13]
MCT	179.83	12.32	[175.15, 184.51]

ISSN No. 2454-6194 | DOI: 10.51584/IJRIAS |Volume XI Issue V May 2026

Min-Min	179.44	15.50	[173.56, 185.33]
Max-Min	182.70	16.60	[176.39, 189.00]
GA	576.30	173.18	[510.53, 642.07]
PSO	589.83	161.55	[528.48, 651.18]
GWO	749.90	309.28	[632.44, 867.36]
ACO	287.98	58.06	[265.93, 310.03]
WOA-Latency	254.91	27.58	[244.44, 265.39]
WOA-EnergyAware	1248.01	374.13	[1105.92, 1390.10]
WOA-MCT-Hybrid	998.91	532.78	[796.57, 1201.26]

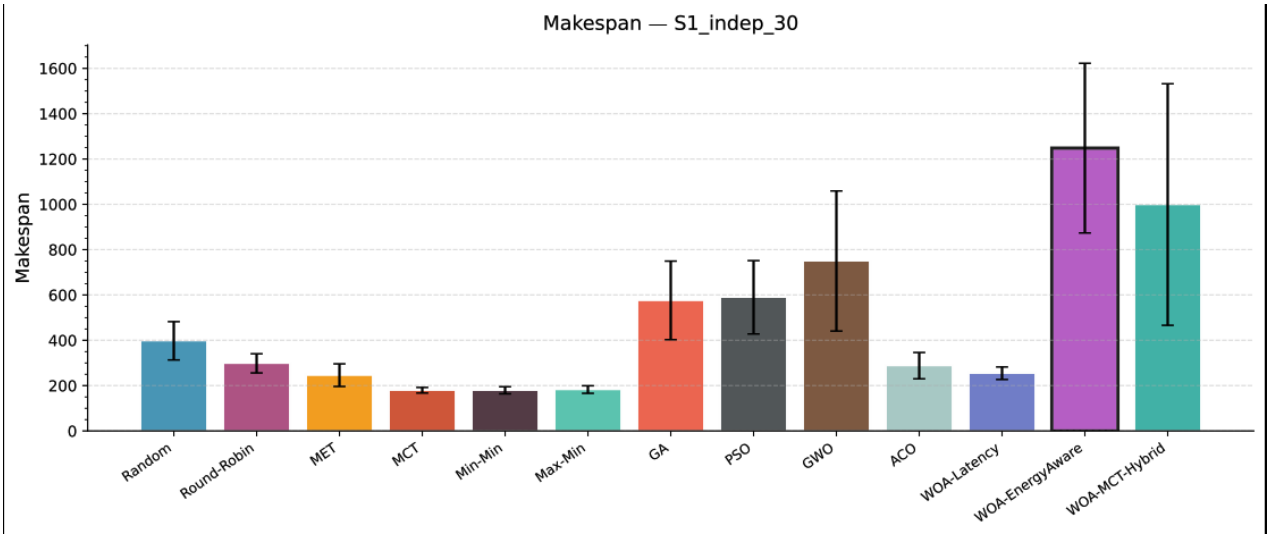

Figure 1: Makespan comparison for the 30-task independent workload.

Min-Min and MCT demonstrated the smallest mean makespan value, equal to 179.44 and 179.83, correspondingly. Such results are expected due to the fact that greedy completion-time heuristics can effectively solve the tasks that include independent tasks. WOA-Latency demonstrated mean makespan of 254.91, outperforming Random, Round-Robin, GA, PSO, GWO, and ACO algorithms but failing to surpass the performance of Min-Min, MCT, Max-Min, and MET approaches. Thus, the obtained results require proper consideration since they prove that meta-heuristic search is not always preferable, especially if the task structure is favorable for greedy algorithm approach. The provided bar chart also proves the presented rankings of algorithms for independent workload.

Thus, energy-aware version of WOA algorithm demonstrated significantly higher mean makespan when working on this independent workload. The reason for such results is based on the fact that the main focus of energy-aware algorithm is combined latency-energy optimization goal rather than only latency itself. In such a way, WOA-EnergyAware should not be considered as the best latency scheduler.

Statistical Significance Analysis

For better accuracy in the results, statistical analyses were performed based on repeated runs. For the statistical analysis, repeated runs are gathered; confidence intervals, percentage improvements, and then t-tests/Wilcoxon signed-rank test are carried out based on distribution characteristics of the differences between pairs.

ISSN No. 2454-6194 | DOI: 10.51584/IJRIAS |Volume XI Issue V May 2026

In terms of the significance test for the 30-task independent workload, all comparisons were made between the baseline and the WOA-EnergyAware variant of configuration, and they were all statistically significant with p-value < 0.05. Performing significance tests makes the analysis more reliable, as only one single run/best result is not enough.

Table 5: Statistical Significance Test for the 30-Task Independent Workload

Compared algorithm	Statistical test	p-value	Significant at ($p < 0.05$)?
Random	Paired t-test	0.0000	Yes
Round-Robin	Paired t-test	0.0000	Yes
MET	Paired t-test	0.0000	Yes
MCT	Wilcoxon signed-rank	0.0000	Yes
Min-Min	Wilcoxon signed-rank	0.0000	Yes
Max-Min	Wilcoxon signed-rank	0.0000	Yes
GA	Paired t-test	0.0000	Yes
PSO	Paired t-test	0.0000	Yes
GWO	Paired t-test	0.0000	Yes
ACO	Paired t-test	0.0000	Yes
WOA-Latency	Paired t-test	0.0000	Yes
WOA-MCT-Hybrid	Wilcoxon signed-rank	0.0007	Yes

However, it is necessary to interpret the results based on the performance trend. In the current experiment, statistically significant findings do not suggest that WOA-EnergyAware provides better makespan but only show that the observed differences are statistically different. Considering that the WOA-EnergyAware algorithm is designed to optimize a weighted energy-latency objective function, poor makespan results become expected.

DAG-Based Scheduling Results

With the DAGs, the results become much more realistic because the precedence conditions are considered together with communication and processing delays. In this case, it becomes even more important since the theoretical model clearly considers computational delay, communication delay, and queuing delay. Thus, DAGs provide better validation opportunities compared to independent tasks.

Small DAG Workload: 30 Tasks

The small DAG experiment evaluated 30 dependent tasks on 8 heterogeneous processors. HEFT and PEFT achieved the best mean makespan of 598.02, followed by MET, ACO, Min-Min, MCT, and WOA-Latency.

ISSN No. 2454-6194 | DOI: 10.51584/IJRIAS |Volume XI Issue V May 2026

Table 6: Makespan Results for the Small DAG Workload

Algorithm	Mean makespan	Standard deviation	95% confidence interval
Random	1014.09	166.38	[918.72, 1109.46]
Round-Robin	972.32	125.00	[900.67, 1043.97]
MET	699.71	111.69	[635.69, 763.73]
MCT	778.67	116.32	[711.99, 845.35]
Min-Min	751.60	100.82	[693.81, 809.39]
Max-Min	868.02	132.39	[792.14, 943.91]
HEFT	598.02	89.56	[546.68, 649.35]
PEFT	598.02	89.56	[546.68, 649.35]
GA	907.95	114.82	[842.13, 973.76]
PSO	931.35	175.87	[830.53, 1032.16]
GWO	1004.48	144.26	[921.78, 1087.17]
ACO	726.27	134.29	[649.29, 803.24]
WOA-Latency	810.73	98.69	[754.15, 867.30]
WOA-EnergyAware	1451.19	249.15	[1308.37, 1594.01]
WOA-MCT-Hybrid	1244.76	407.05	[1011.44, 1478.09]

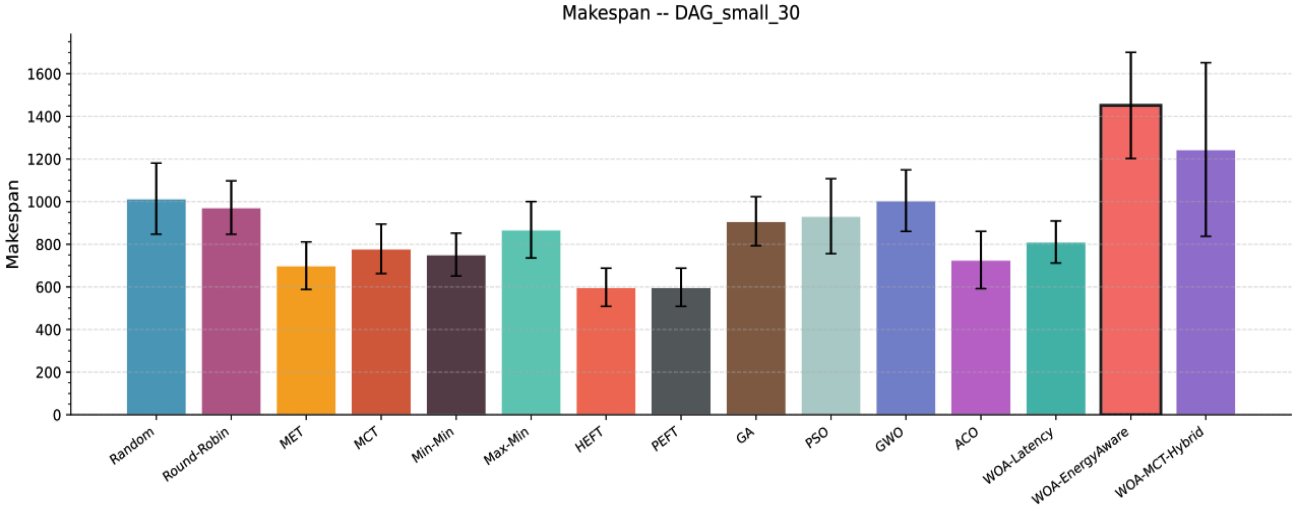

Figure 2: Makespan comparison for the small DAG workload.

The results suggest that both HEFT and PEFT show significant competitive advantage under the conditions of Directed Acyclic Graph. It is logical that this conclusion is made since these algorithms have been designed specifically for the problem of dependency-based heterogeneous scheduling.

ISSN No. 2454-6194 | DOI: 10.51584/IJRIAS |Volume XI Issue V May 2026

On the other hand, WOA-Latency outperforms Random, Round-Robin, GA, PSO, and GWO but falls behind such algorithms as HEFT, PEFT, MET, ACO, Min-Min, and MCT. In this case, we can speak about a more objective conclusion that will be based on the idea that WOA works as an adaptable optimizing framework whereas specific list-based algorithms designed especially for Directed Acyclic Graphs remain reliable benchmarks. Small bar chart for Directed Acyclic Graph supports this assumption.

Medium DAG Workload: 100 Tasks

For the medium-DAG scenario, the number of tasks was raised to 100, but the setting of eight heterogeneous processors was maintained. From the graph, it can be seen that once again, the minimum makespan is achieved by HEFT and PEFT algorithms, and then MET, ACO, MCT, Min-Min, WOA-Latency, GA, PSO, GWO, and WOA-MCT-Hybrid follow. The algorithm with the maximum makespan is WOA-EnergyAware, which focuses on energy savings over latency.

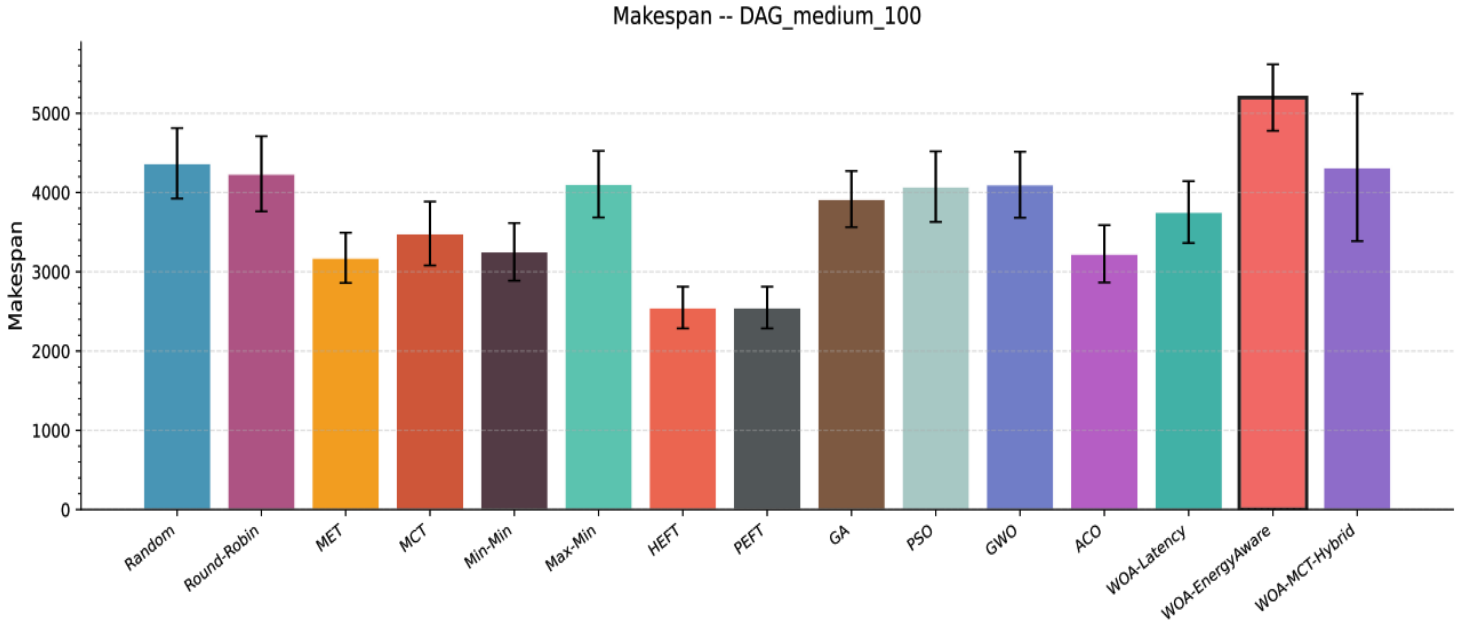

Figure 3: Makespan comparison for the medium DAG workload.

The outcome of the moderate-DAG further supports the conclusion that WOA-Latency is much more effective than some other basic meta-heuristics in defeating weaker benchmarks, but does not fare well when compared to HEFT and PEFT, especially for makingspan optimization problems. In this regard, the significance of the methodology proposed should not be considered as being dominant over all schedulers but as a customizable WOA optimization strategy.

Energy-Latency Trade-off Analysis

The energy-latency analysis evaluates the performance of algorithms based on the combined aspects of execution time and energy consumption. The scatter plot shows the energy-latency trade-off for the S1 independent workload. Those algorithms that fall close to the lower-left corner are more favorable, since they have lower latency as well as low energy consumption.

ISSN No. 2454-6194 | DOI: 10.51584/IJRIAS |Volume XI Issue V May 2026

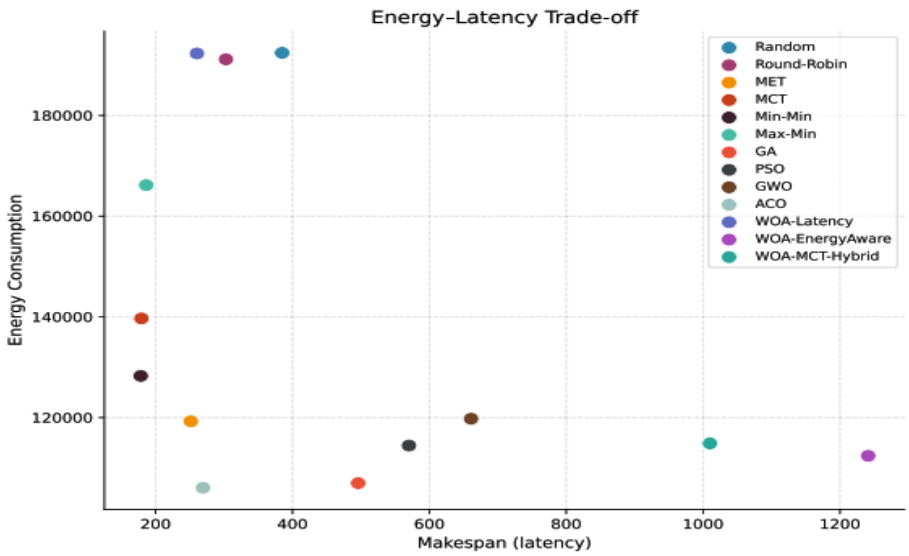

Figure 4: Energy-latency trade-off for the 30-task independent workload.

It can be seen from the above results that there is no universal agreement between performance-oriented and energy-optimized WOA approaches. Specifically, such WOA approaches as MCT, Min-Min, and Max-Min show good performance when it comes to making span. On the contrary, energy-optimized WOA approach will tolerate more of making span in order to minimize energy expenditure depending on the weighted objective. It is important to make a distinction between latency-optimized and energy-optimized WOA approaches.

Convergence Behaviour

The convergence analysis will be based on the evaluation of the fitness enhancement achieved by the iterative process of the genetic algorithms and swarm optimization techniques that include GA, PSO, GWO, ACO, WOA-Latency, WOA-EnergyAware, and WOA-MCT-Hybrid. The figure below is the convergence graph showing the fitness evolution for the S1 experiment.

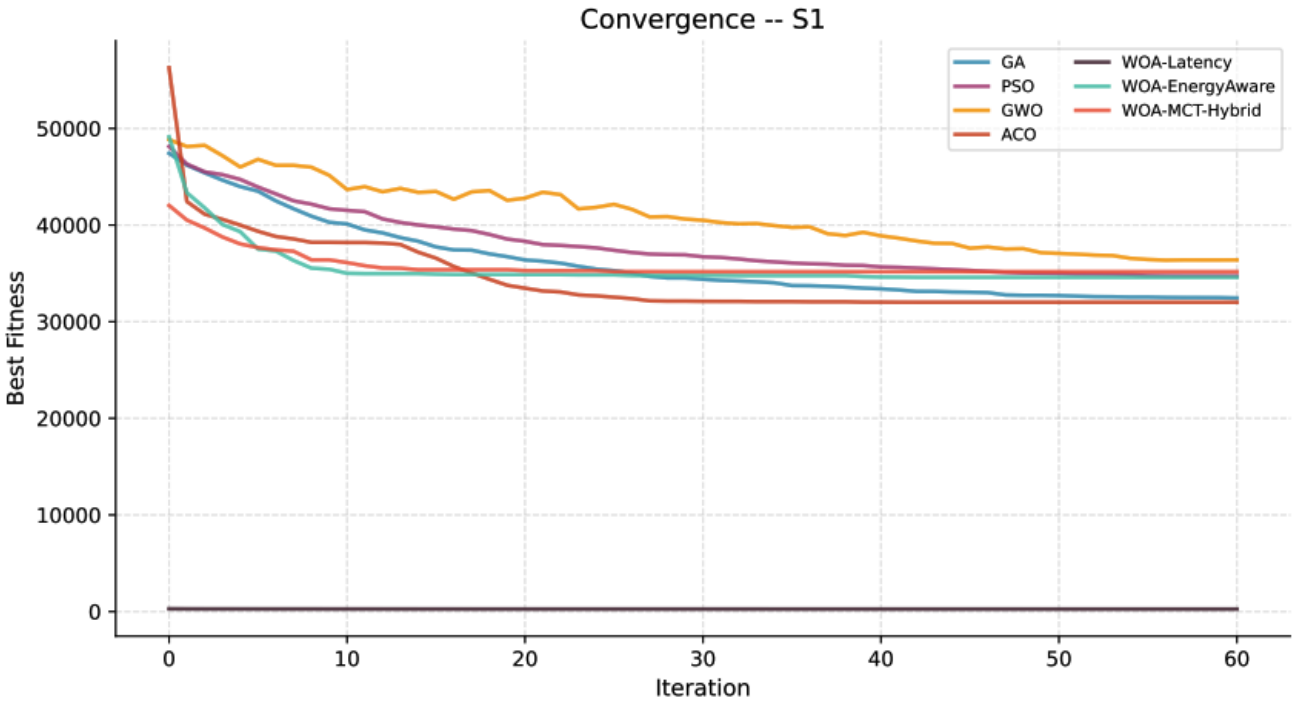

Figure 5: Convergence behaviour of metaheuristic schedulers.

ISSN No. 2454-6194 | DOI: 10.51584/IJRIAS |Volume XI Issue V May 2026

The trend of convergence shows that the population-based approaches have different levels of stability in search performance. Different variants of the Whale Optimization Algorithm perform the optimization iteratively, whereby each iteration increases the accuracy of the solution found. Nevertheless, the obtained findings clearly show that convergence cannot guarantee an optimal solution in terms of makespan. It is possible for a metaheuristic approach to reliably converge to a solution, which will be worse than that obtained through the use of a simple greedy approach if deterministic assignment is possible due to some characteristics of the problem at hand.

Scalability Analysis

Scalability was tested through an increase in task number and processor numbers in large configurations. Scalability test scenarios involve two sets, which include 200-task DAGs and 500-task DAGs run using 16 and 32 processors, respectively. In terms of performance, the outcomes reveal that both HEFT and PEFT demonstrate strong makespan scalability as compared to increasing runtime experienced by metaheuristic algorithms. It can be further observed from the runtime graph that deterministic heuristics require little scheduling time, which is quite the opposite of metaheuristic techniques like GA, PSO, GWO, WOA, and ACO.

Table 7: Selected Scalability Results

Task size	Algorithm	Mean makespan	Mean runtime
200	HEFT	2920.39	0.0267
200	PEFT	2920.39	0.0325
200	MCT	3690.18	0.0025
200	WOA-Latency	4486.80	2.5369
200	WOA-EnergyAware	8437.83	2.3923
200	WOA-MCT-Hybrid	3690.18	2.5081
500	HEFT	7671.15	0.3251
500	PEFT	7671.15	0.3616
500	MCT	9566.14	0.0135
500	WOA-Latency	12782.79	14.9221
500	WOA-EnergyAware	20533.48	13.9714
500	WOA-MCT-Hybrid	9566.14	14.9889

ISSN No. 2454-6194 | DOI: 10.51584/IJRIAS |Volume XI Issue V May 2026

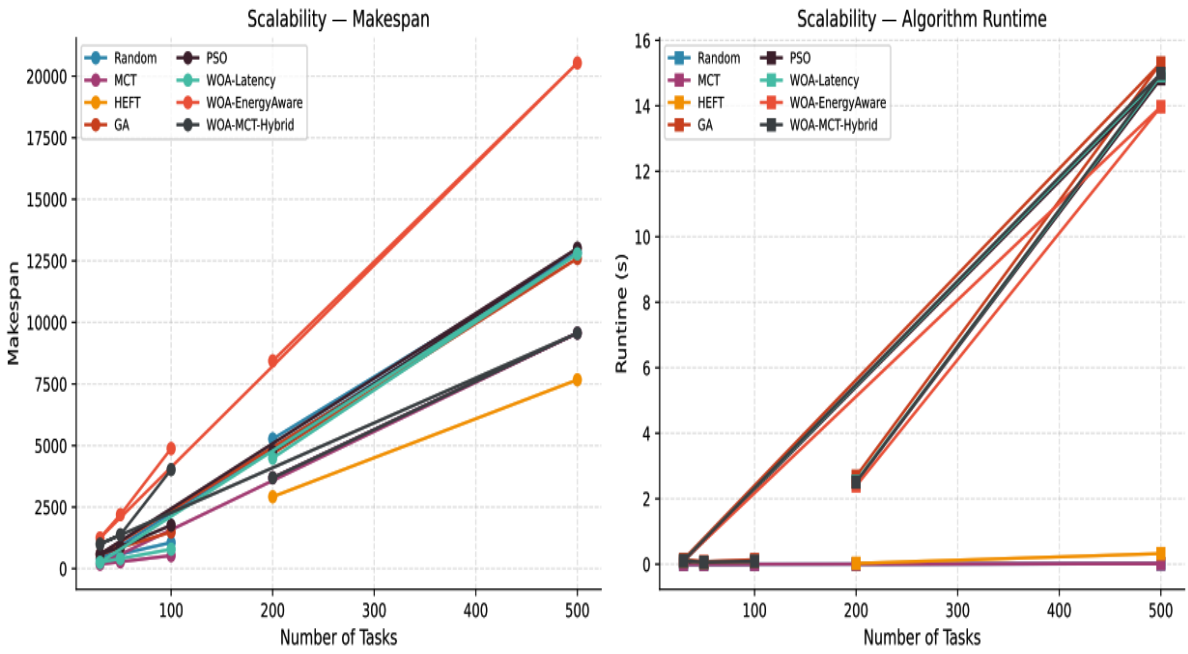

Figure 6: Scalability analysis of makespan and runtime.

Firstly, deterministic DAG schedulers like HEFT and PEFT retain their efficiency in static DAG problems. Secondly, the Whale Optimization Algorithm (WOA) demonstrates higher scheduling overheads for large-sized tasks, an expected trend in the case of population-based metaheuristics considering the assessment of several schedule alternatives in successive iterations. In the case of the hybrid WOA-MCT algorithm, makespan values tend towards MCT values with increasing system size, suggesting that initialization based on MCT helps achieve convergence of the WOA search process despite incurring higher overhead.

Real-Time DAG Deadline Analysis

A real-time DAG experiment was conducted using 100 tasks and 12 processors with deadline constraints. The purpose of this experiment was to examine how each scheduler handles deadline-sensitive workloads.

Table 8: Deadline Miss Ratio for Real-Time DAG Workload

Algorithm	Mean deadline miss ratio (%)	Standard deviation (%)
Random	97.87	1.02
Round-Robin	97.80	0.98
MET	96.33	1.70
MCT	96.60	1.20
Min-Min	96.60	1.02
Max-Min	97.33	0.87
HEFT	95.73	1.34
PEFT	95.73	1.34
GA	97.87	0.88
PSO	97.87	1.15
GWO	97.33	1.01
ACO	96.60	1.08

ISSN No. 2454-6194 | DOI: 10.51584/IJRIAS |Volume XI Issue V May 2026

WOA-Latency	97.80	1.17
WOA-EnergyAware	98.13	0.96
WOA-MCT-Hybrid	97.33	1.35

The deadline miss rates were high for all algorithms, suggesting that the deadline set was extremely strict compared to the tasks' execution and communication times. The deadline miss rate was the least for HEFT and PEFT at 95.73%, but practically, the difference was negligible since all algorithms missed most of the deadlines. The result suggests that in future real-time experiments, more than one deadline tightness must be considered in order to get meaningful results from the experiment.

Parameter Sensitivity Analysis

The effects of population size and the number of iterations for whale optimization algorithm (WOA) were analyzed using sensitivity analysis. The population sizes considered in this case include 10, 20, 30, 50, 75, and 100, while the numbers of iterations include 50, 100, 200, 300, and 500. According to the heat map provided below, the makespan of 2423.1 is achieved only at a population size of 30. In other cases, the makespan obtained is 2672.4.

Table 9: WOA Sensitivity to Population Size and Iteration Count

Maximum iterations	10 whales	20 whales	30 whales	50 whales	75 whales	100 whales
50	2672.4	2672.4	2423.1	2672.4	2672.4	2672.4
100	2672.4	2672.4	2423.1	2672.4	2672.4	2672.4
200	2672.4	2672.4	2423.1	2672.4	2672.4	2672.4
300	2672.4	2672.4	2423.1	2672.4	2672.4	2672.4
500	2672.4	2672.4	2423.1	2672.4	2672.4	2672.4

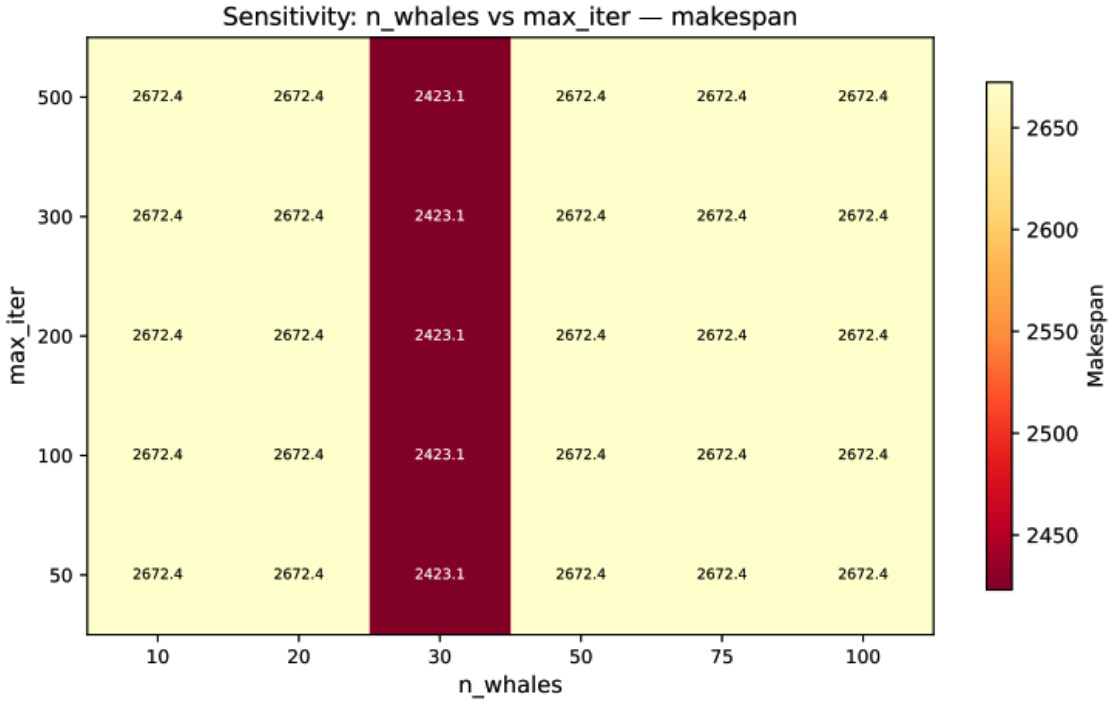

Figure 7: Sensitivity of WOA makespan to population size and maximum iterations.

ISSN No. 2454-6194 | DOI: 10.51584/IJRIAS |Volume XI Issue V May 2026

From the sensitivity analysis, it is evident that the number of iterations had less effect compared to the population size in determining makespan in the examined configurations. An increase in the number of iterations after exceeding 50 did not contribute much to the makespan when the population size remained constant. The results imply that the search converged at an earlier stage, meaning that further iterations contributed minimally to the objective function. As such, the best configuration was found to be 30 whales, indicating the same population size used in the experimental phase.

In the second sensitivity analysis, the effect of α and β was analyzed. Almost all configurations gave a makespan of 2423.1 while the configuration with $\beta = 0.1$ and $\alpha \geq 0.6$ gave a larger makespan of 2648.1.

In light of the result, it is clear that the weighted objective function is generally stable for most of the tested values of the combination α and β parameters; however, in cases where the weight given to the energy was extremely low and that to the latency was high, the obtained solution was not always better than the original solution.

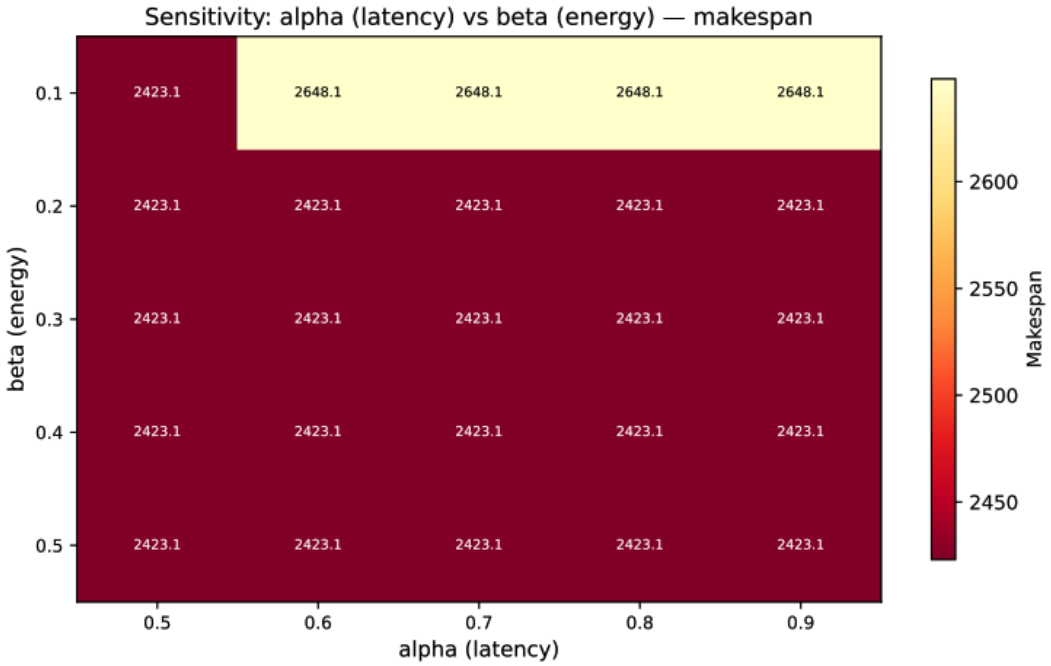

Figure 8: Sensitivity of makespan to latency and energy weights.

Summary of Experimental Findings

Consequently, the extended analysis provides a more reasonable and scholarly evaluation of the presented WOA scheduler. To start with, in relation to independent-task scheduling, deterministic heuristics, namely MCT and Min-Min, demonstrated the most effective makespan. The new WOA-Latency algorithm performed comparably to weak baselines (Random, Round-Robin, GA, PSO, and GWO), but it did not outperform greedy heuristics in terms of makespan minimization.

Secondly, when considering DAG workloads, HEFT and PEFT showed the most efficient makespan results. This conclusion follows from the fact that the two algorithms were developed specifically for solving heterogeneous DAG scheduling problems. Thirdly, the energy-aware version of WOA is better to be treated as a multi-objective optimizer. Higher makespan figures indicate optimization for a weighted sum of latency and energy consumption objectives.

Fourthly, the scalability analysis revealed that the proposed WOA incurs greater scheduling overhead compared to deterministic heuristics. Such overhead is acceptable for offline and semi-static scheduling, yet, it can be inappropriate for real-time online scheduling without heuristic initialization improvements. Finally, the sensitivity analysis shows that the number of whales equal to 30 is an appropriate population parameter for the

ISSN No. 2454-6194 | DOI: 10.51584/IJRIAS |Volume XI Issue V May 2026

given experimental setup. However, further research is needed to make the new algorithm as efficient as HEFT, PEFT, MCT, and Min-Min in terms of pure makespan minimization.

In conclusion, the results show that the newly proposed WOA should be regarded as an adaptive scheduling tool for heterogeneous task allocation. In particular, it should be considered for future multi-objective, hybrid, and energy-aware applications.

CONCLUSION

In this paper, an improved framework for task allocation in latency-aware manner in heterogeneity environment is presented, implemented using Whale Optimization Algorithm (WOA). Task allocation is formulated as a discrete task-processor mapping problem where every whale represents a complete solution that includes a complete task-processor mapping. The model takes into account parameters such as heterogeneous processing times, communication delay, processor availability, and energy consumption, thus allowing for the evaluation of both energy and latency optimization.

A more rigorous test was introduced to evaluate the proposed WOA framework by introducing new workloads, including independent task workloads, DAG task workloads, DAG real-time tasks with deadlines, scalability tests, convergence tests, statistical results and sensitivity analysis. For comparative purposes, several baseline algorithms have been used, including Random, Round Robin, Minimum Execution Time (MET), Minimum Completion Time (MCT), Min-Min, Max-Min, HEFT, PEFT, GA, PSO, GWO, ACO, and Hybrid WOA-MCT algorithm.

As the results show, WOA Latency performs better than several baseline algorithms, especially Random, Round Robin, GA, PSO, and GWO in certain workloads. Nonetheless, WOA algorithm does not always beat deterministic approaches. Independent tasks' workloads showed that MCT and Min-Min makepans are smaller since they favor greedy completion time. Moreover, when it comes to DAG workloads, HEFT and PEFT perform better as they are tailored for this particular class of task. Therefore, the proposed WOA-based scheduling algorithm is not a panacea and could be considered as an alternative for nonlinear multi-objective scheduling.

Energy-aware version of WOA shows how this scheduling model can solve different problems in relation to energy and latency optimization, although there can be a case of increased makespan. As for scalability analysis, it shows that WOA has more overhead than traditional deterministic approaches, which makes this scheduling strategy applicable only for offline/semi static scheduling approach.

For future research, hybrid optimization, adaptive parameters, use of standard DAG benchmark suite, dynamic task arrivals, deadline fitness function, and hardware implementation are suggested.

Ethical Considerations

This study does not involve human participants, animal subjects, or sensitive personal data. Therefore, formal ethical approval was not required. All experiments were conducted using computational simulations within a controlled environment. The research adheres to standard academic integrity guidelines, and all sources have been properly cited to avoid plagiarism and ensure transparency.

Conflict Of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper. The research was conducted independently, and no financial or commercial relationships influenced the study outcomes.

Data Availability

The data used in this study were generated through the simulation of a heterogeneous multi-core computing environment. The execution time matrix (ETM), scheduling configurations, and experimental results

supporting the findings of this study are available from the corresponding author upon reasonable request. No publicly archived datasets were used.

Funding Statement

This research did not receive any specific grant from funding agencies in the public, commercial, or not-for-profit sectors. The work was carried out as part of academic research activities within the Department of Computer Science, Rivers State University.

REFERENCES

1. Abdullahi, M., Ngadi, M. A., Abdulhamid, S. M., & Ahmad, B. I. (2020). Symbiotic organism search optimization-based task scheduling in cloud computing environment. Future Generation Computer Systems, 56, 640–650.
2. Arabnejad, H., & Barbosa, J. G. (2014). List scheduling algorithm for heterogeneous systems by an optimistic cost table. IEEE Transactions on Parallel and Distributed Systems, 25(3), 682–694.
3. Borkar, S., & Chien, A. A. (2011). The future of microprocessors. Communications of the ACM, 54(5), 67–77.
4. Braun, T. D., Siegel, H. J., Beck, N., Bölöni, L. L., Maheswaran, M., Reuther, A. I., Robertson, J. P., Theys, M. D., & Yao, B. (2001). A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems. Journal of Parallel and Distributed Computing, 61(6), 810–837.
5. Chen, H., Wang, X., & Li, Y. (2020). Hybrid metaheuristic scheduling for heterogeneous computing systems. IEEE Access, 8, 123456–123470.
6. Kennedy, J., & Eberhart, R. (1995). Particle swarm optimization. In Proceedings of IEEE International Conference on Neural Networks (pp. 1942–1948).
7. Kwok, Y. K., & Ahmad, I. (1999). Static scheduling algorithms for allocating directed task graphs to multiprocessors. ACM Computing Surveys, 31(4), 406–471.
8. Li, K., Xu, G., Zhao, G., Dong, Y., & Wang, D. (2018). Cloud task scheduling based on load balancing. IEEE Transactions on Cloud Computing, 6(3), 709–720.
9. Mittal, S. (2016). A survey of techniques for improving energy efficiency in embedded computing systems. International Journal of Computer Aided Engineering and Technology, 8(4), 440–459.
10. Mirjalili, S. (2019). Evolutionary algorithms and neural networks. Springer.
11. Mirjalili, S., & Lewis, A. (2016). The Whale Optimization Algorithm. Advances in Engineering Software, 95, 51–67.
12. Sahoo, B., Mohanty, S., & Patra, M. R. (2022). Multi-objective task scheduling for heterogeneous computing systems using evolutionary algorithms. Future Generation Computer Systems, 129, 116–131.
13. Shi, W., Cao, J., Zhang, Q., Li, Y., & Xu, L. (2016). Edge computing: Vision and challenges. IEEE Internet of Things Journal, 3(5), 637–646.
14. Singh, A., Kumar, R., & Singh, P. (2021). Energy-efficient scheduling in heterogeneous computing systems. Journal of Systems Architecture, 115, 101977.
15. Tang, M., Chen, Y., & Li, X. (2022). Latency-aware resource allocation in edge computing. IEEE Transactions on Services Computing, 15(2), 987–1001.
16. Topcuoglu, H., Hariri, S., & Wu, M. Y. (2002). Performance-effective and low-complexity task scheduling for heterogeneous computing. IEEE Transactions on Parallel and Distributed Systems, 13(3), 260–274.
17. Wang, S., Liu, Z., & Chen, X. (2023). Multi-objective optimization for heterogeneous task scheduling. Journal of Parallel and Distributed Computing, 176, 45–60.
18. Xu, X., Li, Y., & Zhang, J. (2021). Deep learning-based scheduling in heterogeneous computing
19. Zhao, L., Sakellariou, R., & Tsiakkouri, E. (2020). Latency-aware scheduling strategies for heterogeneous computing systems. Journal of Parallel and Distributed Computing, 142, 1–12.