International Journal of Applied Mathematics

Volume 38 No. 6s, 2025

ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

AJEGWO: A HYBRID METAHEURISTIC FOR JOINT TASK OFFLOADING AND RESOURCE MANAGEMENT IN MEC

[1] Abdelhamid Zabi, [2]Abdelkamel Ben Ali

[1] LIAP Laboratory, University of El Oued, PO Box 789, El Oued 39000, Algeria

Email: zabi-abdelhamid@univ-eloued.dz

[2] LIAP Laboratory, University of El Oued, PO Box 789, El Oued 39000, Algeria

Email: benali-abdelkamel@univ-eloued.dz

Received : 05/05/2025 Accepted : 23/08/2025 Published : 16/10/2025

Abstract

In mobile edge computing (MEC), efficient task offloading and resource allocation is more challenging and of great concern, particularly for resource-constrained user devices (UDs) with diverse workloads. The underlying joint task offloading and resource allocation problem is known to be NP-hard due to its combinatorial nature and complex coupling of variables. Thus, this paper introduces a novel hybrid metaheuristic algorithm, Adaptive Jaya Embedded GWO (AJEGWO), for jointly optimizing computation decision and resource allocation in MEC environments. The AJEGWO is based on integrating the leadership, and exploration capabilities of Grey Wolf Optimization into the adaptive JAYA algorithm, along with momentum and Lévy flight strategies to enhance global search, enables AJEGWO to achieve improved convergence and diversity. The approach aims to minimize a weighted system cost considering both energy consumption and latency, subject to user and system constraints. Simulation results show that AJEGWO outperforms baseline algorithms, achieving robust convergence regardless of initial conditions, and significantly reducing average latency (by over 95% and 85% compared to Jaya and GWO, respectively) as well as energy consumption (by over 60% versus both benchmarks).

Keywords: Mobile edge computing; Computation offloading; Resource management; Optimization; JAYA; GWO

1. Introduction

Mobile Edge Computing (MEC) has emerged as a key technology that extends cloud computing capabilities to the edge of the network, pushing computing resources closer to end users to enable low latency, high bandwidth, and improved quality of service (QoS), as well as more efficient use of network resources [1]. Task offloading is a crucial aspect of MEC referring to delegating computational tasks from user devices to nearby edge servers, deployed in the RAN for remote processing, thus alleviating the burden of resource constraints on user devices.

Task offloading is a fundamental mechanism in mobile edge computing (MEC) for alleviating

Received: August 07, 2025

Volume 38 No. 6s, 2025

ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

the resource burden on user devices by migrating computationally intensive tasks to remote, resource-rich edge servers[2]. This process helps to prolong device battery life, meet quality of service (QoS) and quality of experience (QoE) requirements, and enhance application responsiveness by reducing local computation and associated energy consumption. However, the rapidly increasing data and task volumes generated by modern applications alongside the limited resources available at edge nodes render efficient task offloading and resource management a highly challenging optimization problem.

In particular, managing resources efficiently in a multi-user, multi-server MEC environment centers on solving the joint task offloading and resource allocation problem. This involves jointly deciding local or edge processing for each task, selecting edge servers for offloaded tasks, and allocating CPU frequencies and transmission power for all processing. The strong interdependence and combinatorial nature of these decisions make the problem NP-hard. As a result, metaheuristic algorithms have become the preferred approach for achieving near-optimal solutions efficiently in such complex MEC scenarios.

In recent years, various metaheuristic algorithms have been proposed for Task offloading and resource allocation problem. In [3], Abbas et al. proposed a metaheuristic-based task offloading model for Mobile Edge Computing (MEC) systems. They posed a multi objective optimization problem to minimize both total network energy consumption and overall execution latency by finding the optimal offload decision. Accordingly, three metaheuristics: Ant Colony Optimization (ACO), Grey Wolf Optimization (GWO) and Whale Optimization Algorithm (WOA) were proposed. simulation results showed that GWO outperforms both ACO and WOA in optimizing latency and energy consumption. In [4], a Discrete Teaching Learning-Based Optimization (DTLBO) strategy was suggested for computation offloading and scheduling of IoT workflows in a MEC. environment. The goal was to jointly minimize energy consumption and delay time, while considering task dependencies, communication costs, and deadline constraints. The proposed (DTLBO), outperformed both: Ant Colony Optimization (ACO) and Particle Swarm Optimization (PSO) in reducing delays and energy. These single algorithm methods, while effective for specific objectives, often struggle to find a robust balance between exploration and exploitation. In [5], a task offloading schema based on the Particle Swarm Optimization (PSO) algorithm for Industrial Internet of Things (IIoT) environments was proposed. The task offloading decision is formulated as a weighted sum multi-objective optimization problem to jointly minimize: time delay, energy consumption, and task execution cost. The simulation results demonstrate that the proposed PSO-based, achieved better efficiency and convergence compared to the baseline methods. In [6], a novel GPU-based Particle Swarm Optimization (PSO) algorithm was proposed to optimize task offloading decisions and improve the Quality of Service (QoS) in vehicular edge computing (VEC). The proposed approach surpasses baseline algorithms in terms of convergence speed and solution quality. In [7], the researched suggested an Improved Bat Algorithm (IBA) multi objective optimization problem of minimizing energy consumption and delay. The IBA, enhanced by a linear inertia weight in its velocity update equation to improve its global search capability, exhibited superior accuracy, stability, and efficiency in reducing both metrics. In [8], a Binary

Received: August 07, 2025

International Journal of Applied Mathematics

Volume 38 No. 6s, 2025

ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

Cuckoo Search Algorithm (BCSA) was proposed for computational task offloading decision in MEC. The offloading decision was formulated as multi objective problem to jointly minimize completion time, energy consumption, and service costs, while considering network congestion and server queues. The simulation results demonstrated that the ODM-BCSA is more efficient than a brute force search, significantly minimizing execution time. In [9], the researchers proposed a whale optimization algorithm termed (D-WOA) for the dependent tasks offloading and communication resource allocation problem in UAV-assisted MEC. The aim was to minimize the user's Quality of Experience (QoE) by considering task dependencies. In [10], Feng et al. introduced a hybrid GWO-WOA-based algorithm for computation offloading optimization in MEC environments. Hence, they considered a multi-objective optimization problem with three objectives: time delay, energy consumption, and service cost. While experimental results showed a superior trade-off between solution quality and convergence rate compared to GWO, WOA, PSO, and GA, the proposed algorithm, WOA, PSO, and GA, the hybrid algorithm demonstrates inconsistency in its stability, a significant drawback for a robust.

Furthermore, several works have explored the use of hybrid metaheuristic algorithms to improve the optimization of MEC systems. For instance, Liu et al. suggested a novel hybrid Bat Scheduling Algorithm (HIBSA) for MEC systems [11]. While the proposed HIBSA was shown to be effective and efficient in finding a balanced optimal solution, the approach is limited as it fails to incorporate critical metrics like energy consumption and does not jointly optimize continuous resource allocation variables.

Other task offloading and resource allocation approaches based on problem decomposition and convex optimization have also been introduced in the literature. For instance, in [12], a joint task offloading and resource allocation scheme for NOMA-MEC networks was proposed, where the complex problem is decomposed and solved via matching theory and convex optimization. While this reduces computational complexity. In [13], Xiang Li et al. proposed a decomposed, tractable approach to joint task offloading and resource allocation for streaming applications in cooperative MEC, using convex optimization techniques.

A common limitation of such decomposition and sequential methods is that relaxing the original problem structure and decoupling variables can prevent the global optimality solutions, leading to suboptimal solutions.

Extensive research has investigated the use of metaheuristics for task offloading and resource allocation. Due to the specific limitations of existing techniques, there is a clear need for specialized algorithms that fuse complementary strengths. The Grey Wolf Optimizer (GWO) and the Jaya algorithm are two particularly relevant examples.

The Grey Wolf Optimizer (GWO) is a metaheuristic imitating the hierarchical hunting behavior of gray wolves [14]. It is recognized for its robust global exploration, but a known limitation is its tendency for slow convergence and potential stagnation in local optima during the exploitation phase. Another such algorithm is the Jaya algorithm, it is a parameter less swarm intelligence metaheuristic known for its simplicity, fast convergence, and strong exploitation capabilities [15]. However, its focus on moving towards the best solution often leads to poor

Received: August 07, 2025

Volume 38 No. 6s, 2025

ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

exploration. The extensive research on both algorithms has been synthesized in key survey papers [16], [17], which demonstrate that a wide array of hybrid approaches have been developed to further enhance their performance by leveraging complementary strengths.

Based on the limitations of existing works for task offloading and resource allocation, the No Free Lunch (NFL) theorem [10], and the fact that no prior work has proposed a GWO-Jaya-based approach, this paper introduces the AJEGWO algorithm.

The main contributions of this paper are outlined as follows:
- A novel hybrid metaheuristic algorithm, the Adaptive Jaya Embedded GWO (AJEGWO), is proposed by deeply embedding the social leadership mechanism of the Grey Wolf Optimizer (GWO) into an enhanced Jaya update equation. An adaptive momentum and Levy flight mechanism is further introduced to effectively balance exploration and exploitation, and accelerate convergence towards an optimal solution.

- The problem of task offloading and resource allocation is formulated by considering a unified solution representation that jointly optimizes both the discrete offloading decisions and the continuous resource allocation variables (CPU frequency, transmission power) simultaneously.
- The proposed AJEGWO is validated considering a realistic system model, based on the Shannon-Hartley theorem for communication rates and a DVFS-aware model for energy consumption.
- The performance of AJEGWO in terms of robustness and efficiency is validated and compared against baseline competitors.

The remainder of this paper is outlined as follows: Section 2 presents the system model and MINLP problem formulation. Section 3 describes the proposed AJEGWO solution. Section 4 evaluates the performance of the AJEGWO algorithm and discusses the simulation results. Section 5 provides the conclusion and future works.

2. System model and Problem formulation

In this section, we first introduce the system model studied in this paper, specifically a mobile edge computing (MEC) system, and describe both the computation model and energy model. And second, the problem formulation is presented.

Received: August 07, 2025

International Journal of Applied Mathematics

Volume 38 No. 6s, 2025

ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

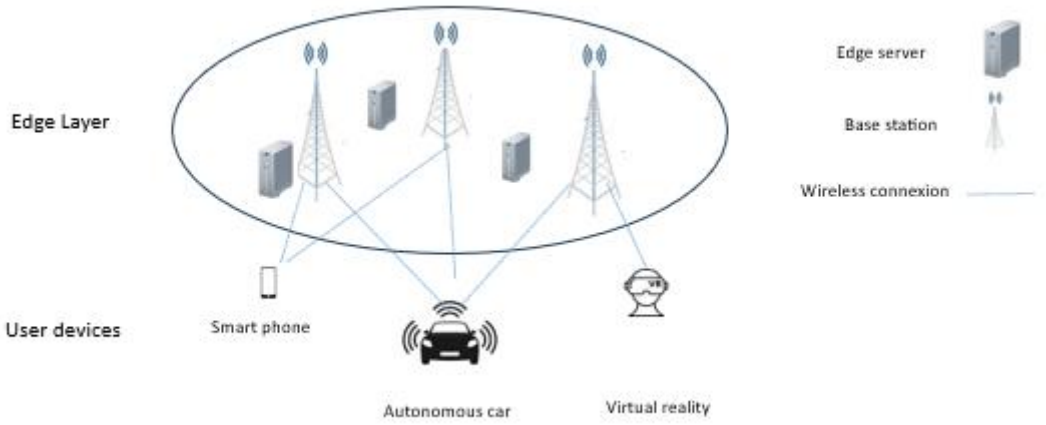


Fig 1. The MEC system architecture

2.1 MEC architecture

As shown in Fig. 1, we consider a MEC system with a set of $\boldsymbol{N}$ = {1, 2, . . ., N} user devices, each has one task to process, and a set of edge servers $\boldsymbol{M}$ = { 1, 2, 3, . . .,M}. Each task i is defined by a tuple:

$\{C_i, D_i, T\_max_i\}$,

where:: C_i is computational demand, this defines the total number of CPU cycles required to complete the task i.

D_i: Data size, this represents the amount of data (in Kilobits) that needs to be transmitted if the task i is offloaded to an edge server.

T_max_i: Maximum latency, this is the maximum tolerable delay or completion deadline for the task i.

2.2 Communication Model

As mentioned earlier, UD*s* transmit data to ES deployed on the BS through the wireless channel. The data transmission rate R_{ij} is determined by the general communication model, the Shannon-Hartley theorem[18], and given by.

$$R_{ij} = B \cdot \log_2\left(1 + \frac{p_i \cdot h_{ij}}{N_0}\right) \quad (1)$$

Where:

B is the channel bandwidth (in Hz).

p_i is the transmission power of the *UD i* (in Watts), and represents the power allocated for transmitting task data over the wireless channel.

h_{ij} is the channel power gain between *UD i* and server j, which represents a factor for combined path loss and multipath fading, and is calculated using the Rayleigh fading model [19].

Received: August 07, 2025

International Journal of Applied Mathematics

Volume 38 No. 6s, 2025

ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

N_0 is the communication channel's noise power.

2.3 Computation Model

a. Local computation:

The local processing time of the task *i* on the *UD* is given by:

$$T_i^L = \frac{C_i}{f_i^L} \tag{2}$$

Where f_i^L is the computational capacity (i.e., CPU cycles per second) of the *UD i*.

b. Offloading computation:

The offloading time $T_{i,j}^O$ has two parts: transmission time delay $T_{i,j}^T$ and edge server time delay $T_{i,j}^E$ which are given by Eqs: 1 and 2, respectively:

$$T_{i,j}^T = \frac{D_i}{R_{i,j}} \tag{3}$$

$$T_{i,j}^E = \frac{C_i}{f_j^E} \tag{4}$$

In this paper, we assume the response time required for sending results from *ES j* can be neglected due to the size of the output data of processing.

where f_j^E is the computational capacity (i.e., CPU cycles per second) of the *ES j*.

Based on eqs1,2 and 3, the offloading time delay $T_{i,j}^O$ is calculated by:

$$T_{i,j}^O = T_{i,j}^T + T_{i,j}^E = \frac{D_i}{R_{i,j}} + \frac{C_i}{f_j^E} \tag{5}$$

2.4 Energy model

a. Local energy

$$E_i^L = \kappa \cdot C_i \cdot (f_i^L)^2 \text{ [20]} \tag{6}$$

where κ is the effective switched capacitance that depends on the chip architecture, and is set to $10^{-27.}$ similar to [21].

b. Transmission energy

The energy model for an offloaded task is defined from a user-centric perspective, which is a standard approach, as the primary goal is to prolong the battery life of user devices, and is calculated by:

$$E_i^T = p_i \cdot T_{i,j}^T \tag{7}$$

The problem is formulated as the constraint minimization of the weighted sum of the total system delay and the total energy consumption of user devices. Based on the system model, the problem can be expressed as follows:

Received: August 07, 2025

International Journal of Applied Mathematics

Volume 38 No. 6s, 2025

ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

$$min \sum_{i=1}^{N} (\omega \cdot E_i + (1-\omega) \cdot T_i) \quad (8)$$

$$s.t.\ T_i \leq T_{max,i},\ \ \forall i \in N, \quad (9)$$

$$f_i^L \geq 0,\ \forall i \in N, \quad (10)$$

$$p_i \geq 0,\ \ \forall i \in N, \quad (11)$$

$$f_j^E \geq 0\ \ \forall j \in M, \quad (12)$$

$$0 < \omega < 1, \quad (13)$$

where ω,1- ω denote the weights of latency and energy, respectively, reflecting their relative importance in the objective function, with w∈[0,1] as in Eq. 13. Eqs [9-13] define the constraints of the system. Specifically, Eqs (10), (11), and (12) ensure that the CPU and power of user devices (*UD*s), and the CPU of edge servers (ES), are all positive, respectively.

The key system notations are listed in **Table 1**.

3. Proposed algorithm AJEGWO

In this section, the proposed approach is introduced and described in detail.

a. Description

We designed a novel hybrid metaheuristic, the Adaptive Jaya Embedded GWO (AJEGWO), for joint computation offloading and resource allocation. Unlike conventional methods, AJEGWO encodes both offloading and resource decisions in a single solution vector, ensuring their simultaneous, unified optimization.

The core of AJEGWO is an enhanced update mechanism that integrates three main innovations: The original Jaya algorithm updates each candidate solution by guiding it toward the best solution and away from the worst solution found in the population, using the following update equation:

$$X_{i,\text{new}} = X_i + r_1 \cdot (X_{\text{best}} - |X_i|) - r_2 \cdot (X_{\text{worst}} - |X_i|) \quad (14)$$

Here X_i is the current solution, X_{best} and X_{worst} are the best and worst solutions in the current population, respectively, and $r1$,$r2$ are random vectors [15].

To enhance the search ability and specifically tailor the optimizer for the complex joint task offloading and resource allocation problem, we propose a novel update equation that deeply embeds the Grey Wolf Optimizer (GWO) leadership mechanism and additional search dynamics.

The main innovations in this equation are:

First, the attraction toward the single best solution in Jaya is replaced by a composite target position multi leader direction derived from the top three GWO solutions (alpha, beta, delta) [14], which is defined by:

Received: August 07, 2025

International Journal of Applied Mathematics

Volume 38 No. 6s, 2025

ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

$$X_{\text{target}} = \frac{1}{3}(X_1 + X_2 + X_3) \tag{15}$$

where, X1, X2, X3 are the positions guided by the alpha, beta, and delta wolves, respectively, calculated as:

$$X_1 = X_\alpha - A_1 \cdot |C_1 \cdot X_\alpha - X_i| \tag{16}$$

$$X_2 = X_\beta - A_2 \cdot |C_2 \cdot X_\beta - X_i| \tag{17}$$

$$X_3 = X_\delta - A_3 \cdot |C_3 \cdot X_\delta - X_i| \tag{18}$$

Here, X_i is the current solution, while X_α, X_β, X_δ are the top three solutions in the population, and A_i and C_i are calculated as in standard GWO

Second, To enhance the convergence speed and stability of the proposed metaheuristic, we introduce a momentum term in the update step [22], applied to each candidate solution, and calculated as:

momentum =$m \cdot vi$

where,

vi represents the previous update vector (or velocity) of the candidate solution *i*, calculated as the difference between its current and previous positions.

m is the momentum coefficient between 0 and 1 that scales this previous update, controlling the influence of past movements on the current search step, in this paper is set to 0.4. This momentum, $m \cdot vi$ is a form of memory that help the algorithm to retain beneficial search directions from past iterations [22]; this reduces oscillations and cyclic patterns, leading to more stable and efficient convergence.

Third, to improve global exploration and prevent the algorithm from getting trapped in local optima, Levy flight perturbations are introduced by Eq. 19 with a certain probability.

$$X_{i,\text{new}} = X_{i,\text{new}} + w \cdot L(\text{if rand } () < p_{\text{levy}}) \tag{19}$$

where, *L* is a step length drawn from a Levy distribution [23], *w* is an adaptive weight that decreases over iterations, and p_{levy} is the probability of performing a Levy flight.

Accordingly, the overall update equation of the proposed AJEGWO is formulated as:

$$X_{i,\text{new}} = X_i + \underbrace{r_1 \cdot (X_{\text{target}} - |X_i|)}_{\text{Attraction}} - \underbrace{r_2 \cdot (X_{\text{worst}} - |X_i|)}_{\text{.epulsion}} + \underbrace{m \cdot v_i}_{\text{Momentum}} + \underbrace{w \cdot L}_{\text{Levy Flight}} \tag{20}$$

b. Solution encoding

In the proposed framework, each candidate solution is represented as a unified, real valued vector that jointly encodes both offloading decisions and resource allocation variables. This design allows all relevant parameters to be optimized simultaneously, preserving the

Received: August 07, 2025

International Journal of Applied Mathematics

Volume 38 No. 6s, 2025

ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

interdependence between computing placement and resource assignment, and avoiding the suboptimality of decomposed methods.

Formally, for a system with N user devices and M edge servers, each solution vector is represented as:

$$\text{individual} = [\underbrace{x_{11}, x_{12}, \dots, x_{1M}, \dots, x_{N1}, \dots, x_{NM}}_{\text{x}}, \underbrace{f_1^L, \dots, f_N^L}_{\text{f}_{\text{local}}}, \underbrace{p_1, \dots, p_N}_{\text{p}}, \underbrace{f_1^E, \dots, f_M^E}_{\text{f}_{\text{edge}}}] \quad (21)$$

where, $x_{ij} \in [0,1]$ are continuous offloading variables indicating the preference for assigning task *i* to *ESj*. These variables are dynamically binarized during the optimization process by:

$$x_{ij}^{(\text{bin})} = \begin{cases} 1 & \text{if} x_{ij} \geq \text{threshold}(t) \\ 0 & \text{otherwise} \end{cases} \quad (22)$$

If all $x_{ij}^{(\text{bin})} = 0$ for user *i*, the task is processed locally.

where,

f_i^L is the local CPU frequency for *UD i*.

p_1 is the transmission power used by device *i*.

f_j^E is the assigned CPU frequency resource for *ES j*.

threshold is a real number between 0.3 and 0.9, chosen by experimentation and calculated as:

$$\text{threshold}(t) = 0.3 + 0.6 \times \frac{t}{T_{max}} \quad (23)$$

where, t and T_{max} are the current iteration number and the maximum iteration number, respectively.

This dynamic thresholding strategy renders the algorithm more adaptive by promoting more exploration in the early stages and encouraging more decisive decisions in later iterations.

Therefore, the total solution length is: N×M+2N+M*N*×*M*+2*N*+*M*.

This unified encoding design enables the algorithm to jointly navigate the entire solution space, maintaining interdependencies between task assignment and resource allocation problems and supporting practical, feasible solutions for real-world edge systems.

Algorithm 1: The Adaptive Jaya Embedded GWO (AJEGWO)

Input: Population size N_p, max iterations Max_iter, objective function Fitness(X)
Output: The best solution found X*

1. Initialize:
2. Initialize population X and velocities v.
3. Evaluate fitness f(X_i) for all solutions i = 1, ..., N_p by Eq. .
4. Identify initial global best X* and its fitness f*.
5. Main Loop:

Received: August 07, 2025

International Journal of Applied Mathematics

Volume 38 No. 6s, 2025

ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

```
for t = 1 to Max_iter do
    Identify leaders X_α, X_β, X_δ and the worst solution X_worst from the population.
    Update global best X* if a better solution is found.
    Update adaptive GWO parameter a and Levy flight weight w.
    for each solution X_i in population X do
        // GWO-Embedded Guidance
        Calculate the GWO-guided target X_target = (X_1 + X_2 + X_3)  Equ.  / 3, where
X_1, X_2, X_3 are derived from the encircling behavior of the leaders X_α, X_β, X_δ.
        // Enhanced Jaya Update
        Generate random vectors r_1, r_2.
        ΔX ← r_1 · (X_target - |X_i|)  // Attraction towards GWO target
                - r_2 · (X_worst - |X_i|) // Repulsion from worst
                + m · v_i              // Momentum term
        // Levy Flight for Diversification
        if rand() < p_levy then
            Generate Levy step L.
            ΔX ← ΔX + w · L
        end if
        // Generate and Evaluate Candidate
        X_candidate ← X_i + ΔX
        Apply boundary handling to X_candidate.
        f_candidate ← Fitness(X_candidate) by Eq. .
        // Greedy Selection
        if f_candidate < f(X_i) then
            X_i ← X_candidate
            v_i ← ΔX  // Update velocity
        end if
    end for
end for
Return X*
```

Table 1 Key system notations

Symbol	Definition
M	Set of *ESs*
N	Set of *UDs*
E_i^T	Transmission energy of task *i*
p_i	Transmission power of *UD i*
$T_{i,j}^T$	Transmission delay of task *i*
E_i^L	Local computation energy of *UD i*

Received: August 07, 2025

International Journal of Applied Mathematics

Volume 38 No. 6s, 2025

ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

κ	Fixed factor depends on device's chip architecture
D_i	The size of task *i*
C_i	Computational demand (required CPU cycles) of task *i*
f_i^L	Computational capacity of *UD i*
$T_{i,j}^O$	Offloading time delay of task *i*
$T_{i,j}^T$	Transmission time delay of *UD i*
$T_{i,j}^E$	The edge server time delay of task *i on ES j*
T_i^L	Local processing time of task *i*
f_j^E	Computational capacity of *ES j*
R_{ij}	Channel transmission rate between *UD i and ES j*
B	Channel bandwidth
p_i	Transmission power of the *UD i*
h_{ij}	Channel power gain
N_0	Communication channel's noise power

4. Results and discussion

4.1 Experiment setup

The proposed AJEGWO and the related simulation scenarios are implemented with the Python environment on the system with an Intel Core i5 processor, 8th Gen Intel(R) Core with a clock rate of 3.00 GHz, and 8 GB RAM. The simulation settings are given in **Table 2.**

4.2 Baseline algorithms

In order to validate the effectiveness and efficiency of the proposed AJEGWO algorithm, the following algorithms: GWO [14] and Jaya [15] are considered for benchmark comparison.

Table 2 Main simulation setting

Parameter	Value
Population size	50
Maximum iterations	100
Edge servers	10
User devices	70
Bandwidth	1 MHZ
p_i	1.5 W

Received: August 07, 2025

International Journal of Applied Mathematics

Volume 38 No. 6s, 2025

ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

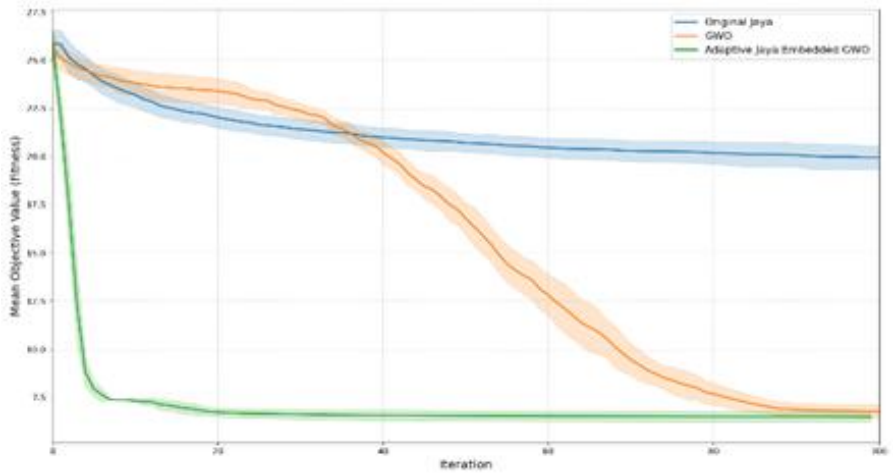


Fig 2. Convergence analysis

a. Convergence analysis

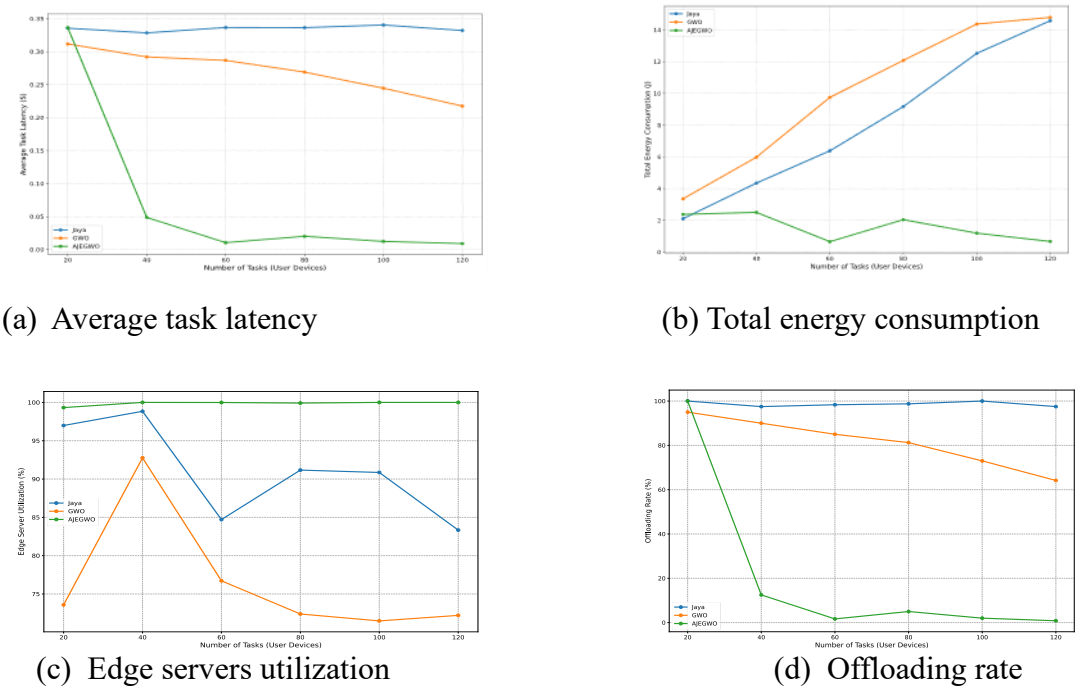
In this scenario, convergence and stability evaluation are carried out. Due to the stochastic nature of metaheuristic algorithms, and to ensure a fair and reliable analysis, each algorithm was executed for 10 independent runs.

Figure .2 shows the results of this analysis, plotting the mean objective value and the standard deviation, represented by the shaded bands across these runs.

As can be seen from the figure, the Original Jaya algorithm consistently suffers from premature convergence, with its mean performance stagnating after approximately 40 iterations due to entrapment in a local optimum. Entering a phase of rapid exploitation after iteration 30; however, its extremely wide shaded band indicates significant performance variance and a lack of consistency across the runs. In contrast, the proposed AJEGWO demonstrates superior performance, achieving the lowest mean fitness value with exceptionally swift convergence within the first 15 iterations. Critically, its narrow shaded band signifies exceptional stability and reliability. These findings confirm that the proposed hybrid algorithm is the most effective strategy, outperforming its parent algorithms not only in mean convergence speed and final solution quality but also in its consistency.

Received: August 07, 2025

International Journal of Applied Mathematics
Volume 38 No. 6s, 2025
ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)



(a) Average task latency (b) Total energy consumption

(c) Edge servers utilization (d) Offloading rate

Fig 3 Performance metrics versus number of tasks

b. Impact of a varying number of tasks (User devices) on performance metrics

In this scenario, we evaluate the impact of varying the number of tasks on the performance metrics: Average task latency, Total energy consumption, Edge server utilization, and Offloading rate.

Figure. 3.a plots the impacts of varying the number of tasks on edge average task latency. As can be found in the figure that AJEGWO demonstrates the lowest and most significantly decreasing average task latency as the number of tasks increases. It starts with a relatively high latency similar to GWO, around 0.31 seconds, at 20 tasks, but then experiences a dramatic decrease, plummeting to approximately 0.05 seconds at 40 tasks. After that, AJEGWO's latency stabilizes at very low values, hovering around 0.01 seconds for 60, 80, and 100 tasks, and slightly decreasing further to just above 0.01 seconds at 120 tasks. This demonstrates its capacity for exceptionally efficient and rapid task processing, especially as the workload scales. In contrast, maintains a relatively high and stable average task latency across all tested task numbers. Starting at around 0.33 seconds for 20 tasks, slightly decreases to approximately 0.32 seconds at 40 tasks, then slightly increases to around 0.33-0.34 seconds for 60, 80, and 100 tasks, before a slight decrease to approximately 0.33 seconds at 120 tasks. This suggests that despite its consistency, Jaya's latency is consistent fails to improve with more tasks. Lastly, GWO (illustrated by the orange line) exhibits a continuously decreasing average task latency as the number of tasks increases, but remains consistently higher than AJEGWO's after 20 tasks. It begins with a latency of just over 0.31 seconds at 20 tasks, decreases to around 0.295 seconds at 40 tasks, and continues a gradual decline to approximately 0.29 seconds at 60 tasks, 0.27 seconds at 80 tasks, 0.245 seconds at 100 tasks, and finally to around 0.22 seconds at 120 tasks. Figure 3.b plots the total energy consumption versus UDs. It can be seen that AJEGWO

Received: August 07, 2025

Volume 38 No. 6s, 2025

ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

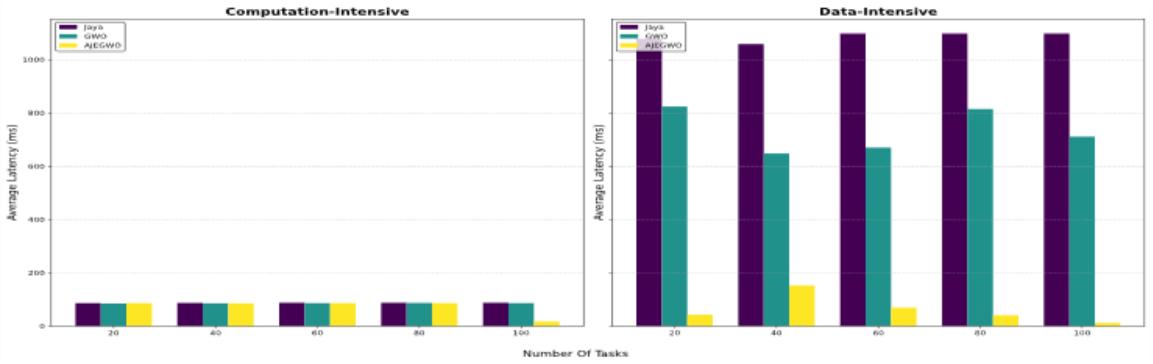
consistently has the lowest total energy consumption as the number of tasks increases. It starts with a low consumption of just over 2 J, then slightly increases to approximately 2.5 J at 40 tasks, and then experiences a sharp decrease, plummeting to around 0.8 J at 60 tasks. After that, AJEGWO's energy consumption remains remarkably low, showing a slight increase to around 2.1 J at 80 tasks, followed by a steady decrease to approximately 1.2 J at 100 tasks, and finally to below 1 J at 120 tasks. In contrast, the total energy consumption of both Jaya and GWO escalates as the number of tasks increases. Jaya's total energy consumption starts at a relatively low level. It then steadily increases with the number of tasks, showing its energy inefficiency for resource-constrained devices.

Figure 3.c depicts the impacts of a varying number of tasks on edge server utilization. From the figure, it can be seen that AJEGWO consistently maintains the highest edge server utilization, staying remarkably close to 100% across the entire range, regardless of the workload. In contrast, Jaya shows a more fluctuating pattern; it starts with high utilization around 97% at 20 tasks, increases slightly to nearly 99% at 40 tasks, then experiences a remarkable decrease to approximately 85% at 60 tasks. Jaya's utilization rises to a peak with increasing tasks, then decreases as the task count becomes large. GWO shows the lowest and most rapidly declining utilization as tasks increase. It starts high, rises to a peak as task count increases, then declines as more tasks are added. As tasks increase, it drops further and stabilizes near 72–73%., revealing its poor scalability under heavy loads.

Figure 3.d plots the impacts of a varying number of tasks on offloading rate. From the figure, it can be found that AJEGWO demonstrates consistently high edge server utilization, maintaining a near 100% utilization rate across all tested numbers of tasks (from 20 to 120 User Devices). This indicates its ability to effectively utilize the available resources on the edge server for the tasks it processes. On the other hand, Jaya shows fluctuating but consistently high edge server utilization. It starts around 97% at 20 tasks, drops to about 84.5% at 40 tasks, peaks at around 96.5% at 60 tasks, then decreases again before rising to approximately 91% at 120 tasks. In contrast, GWO generally exhibits the lowest edge server utilization among the three algorithms. It begins around 85% at 20 tasks, drops to its lowest point around 74% at 40 tasks, and then gradually fluctuates, staying mostly below 85% utilization across the increasing number of tasks. This confirms that AJEGWO effectively leverages the edge server's capacity for the tasks it manages.

Received: August 07, 2025

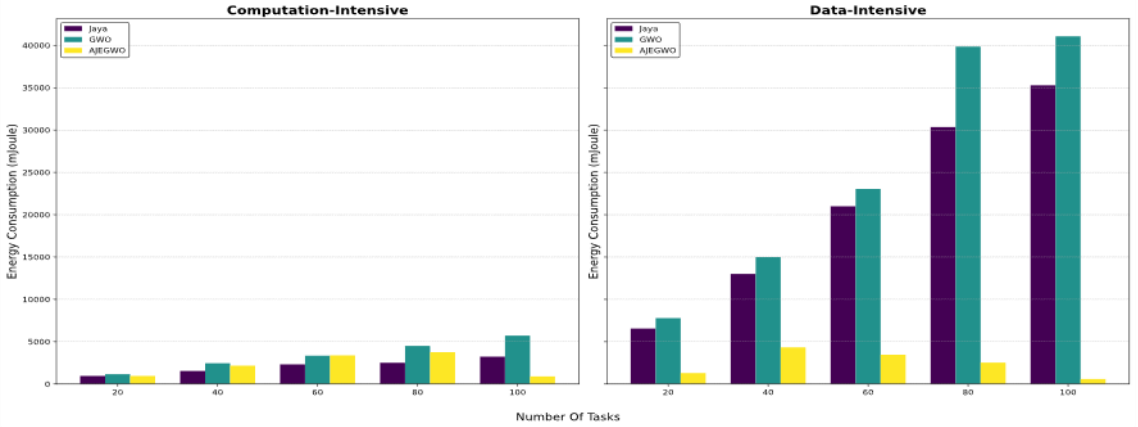
International Journal of Applied Mathematics
Volume 38 No. 6s, 2025
ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)



(a) Average latency for computation-intensive tasks

(b) Average latency for data-intensive tasks

Fig. 4 Impact of Computation workload on Average latency



(a) Energy consumption for computation-intensive tasks

(b) Energy consumption for data-intensive tasks

Fig. 5 Impact of Computation workload on Total energy consumption

c. Impact of workload on performance metrics

As representative edge computing workloads vary significantly, we consider two contrasting scenarios: computation intensive and data intensive [1], [24]. In the computation-intensive case, tasks have smaller data sizes (200–500 kbits) and higher CPU requirements (8–15 million cycles), whereas in the data intensive case, tasks feature larger data sizes (3000–6000 kbits) and lower CPU demands (1–4 million cycles). Moreover, we vary the number of tasks from 20 to 100.

Figure. 4 illustrates the impact of workload on average latency for both (a) data intensive and (b) computation intensive scenarios. In the data intensive scenario (Fig 4.a), for JAYA and GWO, the average latency increases sharply as the number of tasks rises, with Jaya exceeding 1,000 ms and GWO remaining between 600 and 800 ms, while AJEGWO maintains the lowest

Received: August 07, 2025

Volume 38 No. 6s, 2025

ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

latency close to zero. For computation intensive tasks (Fig 4.b), all algorithms exhibit comparably low latency as workload increase. These results demonstrate the outstanding scalability and robustness of AJEGWO, especially in demanding data intensive environments, while confirming its reliable performance for computation intensive workloads.

Figure. 5 plots the impact of workload on total energy consumption for both (a) data intensive and (b) computation intensive scenarios. For the data intensive scenario Fig 5.a, as the number of tasks increases, the energy consumption of Jaya and GWO rises steeply, whereas AJEGWO maintains a substantially lower energy level. By contrast, for computation intensive tasks Fig. 5.b, all algorithms demonstrate similar, moderate energy usage as the workload increases. These results highlight the superior efficiency and robustness of AJEGWO, especially under challenging data intensive conditions, while also confirming its solid performance for computation intensive workloads.

Discussion

The proposed AJEGWO achieves exceptional convergence and stability with a notable quick convergence rate and exceptional stability across multiple runs. Moreover, the AJEGWO makes effective use of edge server resources by intelligently and strategically maximizing their capacity. This is achieved even with a minimal offloading rate, as the algorithm selects only those tasks that will most effectively improve Quality of Service (QoS) and prevent the system from being underloaded, even under heavy loads. Additionally, it consistently shows low latency and low energy consumption across all types of workloads (computation intensive and data intensive), which is particularly beneficial for constrained-resource devices, confirming its efficiency.

6. Conclusion

In this paper, a hybrid metaheuristic-based strategy named AJEGWO for joint task offloading and resource allocation problem in MEC is proposed. This approach, which integrates the leader of the Grey Wolf Optimization (GWO) into the JAYA algorithm, enhances global exploration and diversity. The hybrid technique is further enhanced with a momentum term and Lévy flight to better dampen oscillations and accelerate convergence towards an optimal solution that minimize the weighted sum system cost of both energy consumption and latency, while meeting users and system constraints. The performance of the developed AJEGWO is evaluated trough simulation experiments analysis. The numerical results demonstrated the superiority of the proposed strategy compared to baseline. It yields a robust and reliable convergence regardless the randomness of the initial population, it also achieves a latency reduction of over 95% and 85% compared to Jaya and GWO, respectively, and energy consumption reduction of over 60% compared to both Jaya and GWO. Smart offloading enables high edge utilization with low latency and energy, ideal for resource constrained environments. For future, we plan to validate algorithm's performance under more complex, real-world scenarios. Moreover, the work can be extended to incorporate security and privacy constraints into the problem, and additional real-

Received: August 07, 2025

Volume 38 No. 6s, 2025

ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

world operational factors are recommended for future evaluation.

Declaration of competing interest

The authors declare that they have no competing interest.

References:

[1] P. Mach and Z. Becvar, "Mobile Edge Computing: A Survey on Architecture and Computation Offloading," *IEEE Commun. Surv. Tutor.*, vol. 19, no. 3, pp. 1628–1656, 2017, doi: 10.1109/COMST.2017.2682318.

[2] M. K. Hussein and M. H. Mousa, "Efficient Task Offloading for IoT-Based Applications in Fog Computing Using Ant Colony Optimization," *IEEE Access*, vol. 8, pp. 37191–37201, 2020, doi: 10.1109/ACCESS.2020.2975741.

[3] A. Abbas, A. Raza, F. Aadil, and M. Maqsood, "Meta-heuristic-based offloading task optimization in mobile edge computing," *Int. J. Distrib. Sens. Netw.*, vol. 17, no. 6, p. 155014772110230, June 2021, doi: 10.1177/15501477211023021.

[4] M. K. Hussein and M. H. Mousa, "Efficient Computation Offloading of IoT-Based Workflows Using Discrete Teaching Learning-Based Optimization," *Comput. Mater. Contin.*, vol. 73, no. 2, pp. 3685–3703, 2022, doi: 10.32604/cmc.2022.026370.

[5] Q. You and B. Tang, "Efficient task offloading using particle swarm optimization algorithm in edge computing for industrial internet of things," *J. Cloud Comput.*, vol. 10, no. 1, Dec. 2021, doi: 10.1186/s13677-021-00256-4.

[6] M. A. Alqarni, M. H. Mousa, and M. K. Hussein, "Task offloading using GPU-based particle swarm optimization for high-performance vehicular edge computing," *J. King Saud Univ. - Comput. Inf. Sci.*, vol. 34, no. 10, pp. 10356–10364, Nov. 2022, doi: 10.1016/j.jksuci.2022.10.026.

[7] F. Xu, S. Zi, J. Wang, and J. Ma, "A computing offloading strategy for UAV based on improved bat algorithm," *Cogn. Robot.*, vol. 3, pp. 265–283, 2023, doi: 10.1016/j.cogr.2023.07.005.

[8] M. Alqarni, A. Cherif, and E. Alkayyal, "ODM-BCSA: An Offloading Decision-Making Framework based on Binary Cuckoo Search Algorithm for Mobile Edge Computing," *Comput. Netw.*, vol. 226, p. 109647, May 2023, doi: 10.1016/j.comnet.2023.109647.

[9] L. X. Nguyen, Y. K. Tun, T. N. Dang, Y. M. Park, Z. Han, and C. S. Hong, "Dependency Tasks Offloading and Communication Resource Allocation in Collaborative UAVs Networks: A Meta-Heuristic Approach," *IEEE Internet Things J.*, vol. 10, no. 10, pp. 9062–9076, May 2023, doi: 10.1109/JIOT.2022.3233667.

[10] S. Feng, Y. Chen, Q. Zhai, M. Huang, and F. Shu, "Optimizing computation offloading strategy in mobile edge computing based on swarm intelligence algorithms," *EURASIP J. Adv. Signal Process.*, vol. 2021, no. 1, Dec. 2021, doi: 10.1186/s13634-021-00751-5.

Received: August 07, 2025

Volume 38 No. 6s, 2025

ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

[11] Y. Liu, J. Q. Zhu, and J. Wang, "Computation Offloading Optimization in Mobile Edge Computing Based on HIBSA," *Mob. Inf. Syst.*, vol. 2021, pp. 1–17, Nov. 2021, doi: 10.1155/2021/7716654.

[12] J. Xue and Y. An, "Joint Task Offloading and Resource Allocation for Multi-Task Multi-Server NOMA-MEC Networks," *IEEE Access*, vol. 9, pp. 16152–16163, 2021, doi: 10.1109/ACCESS.2021.3049883.

[13] X. Li, R. Fan, H. Hu, and X. Li, "Joint Task Offloading and Resource Allocation for Streaming Application in Cooperative Mobile Edge Computing," May 08, 2023, *arXiv*: arXiv:2305.04747. doi: 10.48550/arXiv.2305.04747.

[14] S. Mirjalili, S. M. Mirjalili, and A. Lewis, "Grey Wolf Optimizer," *Adv. Eng. Softw.*, vol. 69, pp. 46–61, Mar. 2014, doi: 10.1016/j.advengsoft.2013.12.007.

[15] R. Venkata Rao, "Jaya: A simple and new optimization algorithm for solving constrained and unconstrained optimization problems," *Int. J. Ind. Eng. Comput.*, pp. 19–34, 2016, doi: 10.5267/j.ijiec.2015.8.004.

[16] S. N. Makhadmeh *et al.*, "Recent Advances in Grey Wolf Optimizer, its Versions and Applications: Review," *IEEE Access*, vol. 12, pp. 22991–23028, 2024, doi: 10.1109/access.2023.3304889.

[17] R. A. Zitar, M. A. Al-Betar, M. A. Awadallah, I. A. Doush, and K. Assaleh, "An Intensive and Comprehensive Overview of JAYA Algorithm, its Versions and Applications," *Arch. Comput. Methods Eng.*, vol. 29, no. 2, Art. no. 2, Mar. 2022, doi: 10.1007/s11831-021-09585-8.

[18] Y. Mao, J. Zhang, and K. B. Letaief, "Dynamic Computation Offloading for Mobile-Edge Computing with Energy Harvesting Devices," *IEEE J. Sel. Areas Commun.*, vol. 34, no. 12, pp. 3590–3605, Dec. 2016, doi: 10.1109/JSAC.2016.2611964.

[19] X. Chen, Z. Liu, Y. Chen, and Z. Li, "Mobile Edge Computing Based Task Offloading and Resource Allocation in 5G Ultra-Dense Networks," *IEEE Access*, vol. 7, pp. 184172–184182, 2019, doi: 10.1109/access.2019.2960547.

[20] Q.-V. Pham, T. LeAnh, N. H. Tran, and C. S. Hong, "Decentralized Computation Offloading and Resource Allocation in Heterogeneous Networks with Mobile Edge Computing," Mar. 02, 2018, *arXiv*: arXiv:1803.00683. doi: 10.48550/arXiv.1803.00683.

[21] A. P. Miettinen and J. K. Nurminen, "Energy efficiency of mobile clients in cloud computing".

[22] M. Harper and J. Safyan, "Momentum Accelerates Evolutionary Dynamics," July 05, 2020, *arXiv*: arXiv:2007.02449. doi: 10.48550/arXiv.2007.02449.

[23] X.-S. Yang and S. Deb, "Cuckoo Search via L´evy Flights," *Proc World Congr. Nat. Biol. Inspired Comput. NaBIC 2009*, pp. 210–2014, Dec. 2009.

[24] S. Sardellitti, G. Scutari, and S. Barbarossa, "Joint Optimization of Radio and Computational Resources for Multicell Mobile-Edge Computing," *IEEE Trans. Signal Inf. Process. Netw.*, vol. 1, no. 2, pp. 89–103, June 2015, doi: 10.1109/TSIPN.2015.2448520.

Received: August 07, 2025