Journal of Cloud Computing:
Advances, Systems and Applications

RESEARCH

Open Access

Task offloading exploiting grey wolf optimization in collaborative edge computing

Nawmi Nujhat[1], Fahmida Haque Shanta[1], Sujan Sarker[2], Palash Roy[1,3], Md. Abdur Razzaque[1*], Md. Mamun-Or-Rashid[1], Mohammad Mehedi Hassan[4] and Giancarlo Fortino[5]

Abstract

The emergence of mobile edge computing (MEC) has brought cloud services to nearby edge servers facilitating penetration of real-time and resource-consuming applications from smart mobile devices at a high rate. The problem of task offloading from mobile devices to the edge servers has been addressed in the state-of-the-art works by introducing collaboration among the MEC servers. However, their contributions are either limited by minimization of service latency or cost reduction. In this paper, we address the problem by developing a multi-objective optimization framework that jointly optimizes the latency, energy consumption, and resource usage cost. The formulated problem is proven to be an NP-hard one. Thus, we develop an evolutionary meta-heuristic solution for the offloading problem, namely WOLVERINE, based on a Binary Multi-objective Grey Wolf Optimization algorithm that achieves a feasible solution within polynomial time having computational complexity of $O(M^3)$, where M is an integer that determines the number of segments in each dimension of the objective space. Our experimental results depict that the developed WOLVERINE system achieves as high as 33.33%, 35%, and 40% performance improvements in terms of execution latency, energy, and resource cost, respectively compared to the state-of-the-art.

Keywords Collaborative mobile edge computing, Multi-objective grey wolf optimization, Latency, Service caching, Task offloading

Introduction

The proliferation of seamless internet connectivity technologies, such as WiFi, 4G, 5G, or LTE, as well as the availability of high processing capabilities at the mobile edge, has pushed the horizon of a new computing paradigm called mobile edge computing (MEC) [1–3]. In recent years, the penetration of computation-intensive real-time applications has increased with the rapid rise of massively connected heterogeneous mobile devices (MDs) [4]. According to [5], Cisco predicts that by 2030, almost 500 billion gadgets will be associated with the Internet of Things (IoT). Frequent access to cloud services results in an increase in mobile data traffic as well as backhaul latency, which in turn diminishes the Quality of Experience (QoE) of the application users [1]. The MEC alleviates these problems by bringing the resources closer to the end users [6]. The benefits of MEC can further be extended by introducing collaboration among edge servers located in different geographical regions, called collaborative mobile edge computing (CoMEC) [7]. Not only do the edge servers participate in resource sharing, but vertical collaboration [8] also takes place among the three layers of CoMEC. Vertical collaboration in the MEC environment signifies collaboration among

*Correspondence:
Md. Abdur Razzaque
razzaque@du.ac.bd
[1] Green Networking Research Group, Department of Computer Science and Engineering, University of Dhaka, Dhaka 1000, Bangladesh
[2] Department of Robotics and Mechatronics Engineering, University of Dhaka, Dhaka 1000, Bangladesh
[3] Department of Computer Science and Engineering, Green University of Bangladesh, Narayanganj-1461, Dhaka, Bangladesh
[4] Information Systems Department, College of Computer and Information Sciences, King Saud University, Riyadh, Saudi Arabia
[5] Department of Informatics, Modeling, Electronics, and Systems, University of Calabria, 87036 Rende, Italy

multiple layers of IoT computing infrastructure, including the IoT devices at the bottom, the edge cloud servers at the middle, and the master cloud at the top, as shown in Fig. 1.

While CoMEC increases the sustainability of edge computing, service caching at the MEC layer favors the QoE of the real-time application users [9]. Service caching refers to caching the information that must be known by the edge server to complete the task execution. This information includes system settings, the heavy program code of the application, and their related databases/libraries [10]. Figure 1 illustrates some real-life use cases where caching is exploited in MEC for better QoE. One such case is where the MEC can be exploited for intelligent transportation systems (ITS), such as extending the connected vehicle cloud into the mobile network [11]. As a result, roadside applications operating directly at the MEC may receive local messages from vehicles and roadside sensors, process them, and broadcast alerts (e.g., an accident) to nearby vehicles within the shortest possible time [12]. The second case is of virtual reality and face-recognition data processing in various applications that require frequent database access. Both of these applications are data-intensive and need to deliver output in real time to ensure higher QoE to users. In all of the aforementioned cases, service caching can go a long way to ensure fast services to users. Caching prevents the same data from being offloaded multiple times, thus, both transmission latency and energy consumption can be reduced.

Computation offloading to a CoMEC network considering service caching may improve the overall QoE by reducing the associated system costs in terms of the queuing delay of tasks, energy consumption of devices, monetary costs, and so on [13, 14]. Additionally, it is not realistic to offload all tasks of MD to MEC all the time as the limited storage and computing resources of MEC significantly affect the time delay of the offloaded tasks. Therefore, an optimal task offloading decision needs to be formulated to achieve an efficient network model while keeping the aforementioned system costs minimal. A large number of researches have been done on caching strategies [15, 16] and CoMEC. Content caching, computation offloading, and resource allocation problems have been jointly considered in [4] to reduce users' overall task execution time but it lacks collaboration among the edge servers. An AI-based task allocation algorithm namely iRAF has been proposed in [17] for the CoMEC

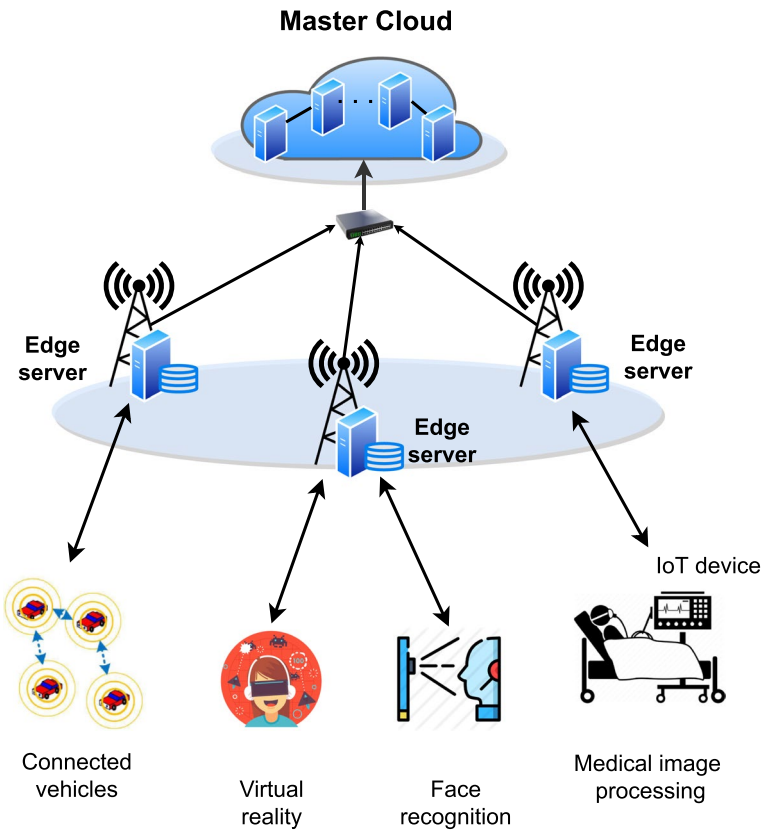


Fig. 1 Real-life applications of service caching in MEC

network where the average latency and energy have been optimized. Here, either one of the objectives is optimized by associating binary weights that create unfairness in the result. In [18], monetary cost and execution delay has been optimized using the particle swarm optimization (PSO) algorithm for a vehicular network. However, addressing mobile energy consumption still remains an issue. Three prime objectives, that is, execution time, energy consumed, and monetary cost have been optimized in a multi-user multi-server environment using a multi-objective evolutionary algorithm (MOEA/D) combining simple additive weighting (SAW) and multi-attribute decision making (MDM) in [19]. This work too lacks collaboration among servers and cache resource allocation which can be crucial to addressing QoE.

This research endeavors to bridge notable gaps that have persisted in the existing body of knowledge in the MEC environment. In a dynamic environment, where heterogeneous mobile devices and edge servers are involved in optimizing multiple objectives simultaneously, no existing solutions can effectively address the problem. Several challenges are encountered while optimizing conflicting objectives together in a complex environment where multiple real-time applications operate on different user devices. Firstly, real-time applications require faster processing than others. If they are computationally expensive, offloading associated data and codes frequently creates a significant overhead. Secondly, handling offloading decisions while executing tasks can slow down the services of edge servers, especially if the resources of the edge servers become saturated, thus degrading QoE. Thirdly, since multiple objective parameters are targeted for optimization, they can be conflicting in nature. Thus, an exhaustive exploration of potential solution combinations becomes imperative. Most of the studies done so far have opted for single-objective optimization associating scalar weights to multiple objective parameters. Some of these depend on multiple decision criteria for selecting solutions [19]. The parameters for such decision-making variables require meticulous fine-tuning and the environment saturated with real-time applications cannot afford to create extra overhead as such. Finally without service caching, every request for a particular service or content would need to travel from the user's device to the edge server or even further to the cloud, resulting in higher latency. This delay can be especially problematic for real-time delay-sensitive applications.

In this paper, we investigate a problem of joint optimization of task execution time, energy, and resource usage cost while offloading tasks in a CoMEC network. A task offloading framework based on grey **WOL**f optimization that exploits **VER**tical collaboration **IN E**dge computing, namely **WOLVERINE** system is devised to solve the problem. The WOLVERINE stands out from other task-offloading frameworks due to its innovative features and advantages. Traditional task offloading frameworks suffer from several drawbacks, which can be categorized into three main areas: 1) lack of reproducibility of offloaded application codes, 2) lack of collaboration among the edge servers, and 3) inability to optimize multiple crucial parameters simultaneously. These limitations have negative implications for network systems, resulting in decreased QoE, underutilized resources, and suboptimal network performance. In response to these challenges, WOLVERINE introduces a novel task offloading scheme for real-life computationally intensive applications, utilizing an evolutionary algorithm. This scheme addresses the collaboration among servers and leverages cached application code to minimize time, energy, and resource costs in edge computing environments. The main contributions of the WOLVERINE framework are listed below:

- We design a collaborative task offloading framework that effectively utilizes cached and computational resources to enhance user QoE in a CoMEC system where real-time applications are executed.
- We formulate the problem of jointly optimizing latency, energy, and resource usage cost as a Multi-objective Linear Programming (MOLP) problem.
- Due to the NP-hardness of the above MOLP, we exploit Binary Multi-Objective Grey Wolf Optimization (BMOGWO), a meta-heuristic evolutionary algorithm, to develop a polynomial time solution to the problem, namely WOLVERINE.
- The experimental results depict that the proposed WOLVERINE system outperforms in terms of execution latency, energy, and resource cost in comparison with [19] by 33.33%, 35%, and 40%, respectively.

The rest of this paper is organized as follows. "Related works" section illustrates the major existing works. "System model" section describes the system model of WOLVERINE. "Design details of WOLVERINE" section elaborates the computational model, multi-objective problem formulation, and meta-heuristic task offloading scheme. "Performance evaluation" section describes the environmental setup and results of experimental analysis. Finally, "Conclusion" section summarizes the key outcomes of our work and some future research directions.

Related works

Several works in the field of collaborative edge computing have been done, including optimal task caching and task allocation while optimizing a single objective

function, trade-offs between two or more objectives, and multi-objective optimization.

The first category of works in the literature focused on single-objective optimization in collaborative edge computing, for example, energy, time, or resource cost allocation. In [2], a genetic algorithm based on a data-aware task allocation strategy has been proposed that considers the network congestion control for allocating sub-tasks. In [20], the authors have focused on the reduction of energy consumption for task assignments by considering the heterogeneity of users using a heuristic-based greedy approach. An architecture has been proposed in [21] that considers unloading resource-intensive tasks from client devices in the cooperative edge space or to the remote cloud depending on users' desire and resource availability. An AI-driven intelligent Resource Allocation Framework (iRAF) [17] has been designed to solve complex resource allocation problems considering the current network states and task characteristics. Another group of authors in [22] have utilized a deep reinforcement learning method to solve computation offloading and resource allocation problems in a blockchain-based multi-UAV-assisted dynamic environment.

Computation offloading that focuses on the minimization of system cost comprising the trade-off between energy and task execution delay in the form of a weighted sum has been proposed in [15]. Collaboration among MEC servers for (data) cache and computational resource allocation are noteworthy in [15]. However, caching the content or code of applications is not enough due to the limited computational capacity of user devices as well as the delay associated with transmitting cached data or code. Hence the idea of jointly task offloading and caching needs to be considered. In [16], a joint service caching, task offloading, and system resource allocation scheme to minimize system cost comprising of time and energy have been formulated using a MILP problem. In [23], a priority-based task offloading and caching scheme is proposed for the MEC environment, where computing a task while reducing energy cost and delay time efficiently is the main priority. A new low-complexity hyper-heuristic algorithm has been proposed in [24], where content caching is performed along with computation offloading in an MEC network to optimize the service latency for all ground IoT devices. Mobility and user preference-aware content-caching in MEC are orchestrated in [25]. The authors in [26] introduce an enhanced binary PSO algorithm, which is designed for optimizing task offloading and content caching in MEC networks. It focuses on jointly optimizing task completion delay and energy consumption. Additionally, an enhanced binary particle swarm optimization (BPSO) algorithm is proposed for content caching in parallel task offloading scenarios. An alternating-iterative algorithm has been developed in [27] for jointly optimizing task caching and offloading in a resource-constraint environment to minimize energy consumption. Here task caching indicates caching of a completed application and relevant data. Subsequently, in [4], content caching, computation offloading, and resource allocation problems have been jointly considered to reduce users' overall task execution time. However, caching a complete application, i.e., content caching is often incompatible with user requirements. Hence, the idea of caching data codes for joint task offloading and data caching using the Lyapunov algorithm for minimizing task computation delay has been introduced in [28]. The authors have formalized joint service caching and task offloading decisions to minimize computation latency while keeping the total computation energy consumption low.

Multi-Objective Optimization problems are adopted for computation offloading in edge cloud by the authors of [29] which focused on the offloading probability of tasks to edge cloud from an MD. To optimize execution time, energy, and resource cost to maximize utility for resource providers in IoT networks, energy harvesting properties of unnamed aerial vehicles (UAV) are used in [30]. A deep reinforcement learning (DRL) based solution is used for this system network that is managed by blockchain. Multi-objective optimization problems have multiple Pareto-optimal solutions which are obtained by trade-offs. Hence, evolutionary algorithms can play a significant role in reaching a single-preferred solution [31]. In [32], time, energy, and cost were minimized for an edge cloud environment using the genetic algorithm NSGA-II. Minimization of average latency and energy consumption simultaneously for offloading tasks using the Cuckoo search algorithm has been proposed in [33]. In [34], Grey-Wolf Optimization is used to perform a trade-off between the minimization of energy consumption and response time in an MEC environment. An Improved Multi-Objective Grey Wolf Optimization (IMOGWO) is used for sub-task scheduling in an edge computing environment introduced in [35] to optimize makespan, load balance, and energy simultaneously. Computation time and cost minimization have been performed in [18] using the Particle Swarm Optimization (PSO) algorithm for a Vehicular Edge Computing (VEC) environment. In [19], a tri-objective problem has been considered in a multi-user and multi-server task offloading environment where an application is divided into multiple independent sub-tasks. A Multi-objective Evolutionary Algorithm based on decomposition (MOEA/D) has been developed for optimizing the time, cost, and energy expended in

Table 1 Summary of methods exploited

State-of-the-art works	Applied technique	Cache	Multi-objective optimization	Edge collaboration	Objective function
[17]	Monte Carlo Tree Search + Multi-Task Learning	✗	✗	✓	Traded-off latency & energy
[4]	Modified branch and bound + Modified generalized benders decomposition method	✗	✓	✗	Minimized task execution latency
[24]	Lagrange dual decomposition	✓	✓	✗	Minimized latency
[19]	Multiobjective Evolutionary Algorithm based on Decomposition	✗	✓	✗	Minimized latency, energy & cost
[38]	Collaborative Filtering + Cosine Similarity & Dynamic Time Wrapping	✓	✓	✗	Minimized latency & energy
WOLVERINE	BMOGWO	✓	✓	✓	Minimized latency, energy & cost

the execution of a particular sub-task. MOEA/D is also used to minimize latency and energy in [36] for the MEC environment, where the ordering of subtasks exists as a constraint. It is also used for minimization of latency and maximization of rewards for servers and tasks in [37]. However, the direct assignment of sub-tasks from mobile devices to a server is costly in terms of energy and offloading decision-making. The works mentioned above that addressed multi-objective optimization do not have a system environment similar to that of CoMEC handling real-life applications.

The summary of the state-of-the-art works has been listed in Tables 1 and 2. Most of the existing literature works have either performed single-objective optimization or weighted optimization in multi-user multi-server networks with and without cache or have performed multi-objective optimization without caching and collaboration among servers. The problem of jointly optimizing three basic objectives: execution latency, device energy, and resource cost has not yet been resolved in the CoMEC system incorporating service caching. The generation of Pareto-optimal solutions for optimizing multiple objectives simultaneously in a resource-constrained environment where servers collaborate and cache service is yet to be done. These observations have driven us to design a task offloading framework in the CoMEC environment for generating Pareto-optimal solutions for multi-objective optimization by exploiting service caching of computational resources.

Table 2 Summary of targeted performance parameters

State-of-the-art works	Execution time	Energy consumption	Monetary cost
[20]	✓	✗	✗
[4]	✗	✓	✗
[17, 22, 33, 34, 38]	✓	✓	✗
[18, 19], WOLVERINE	✓	✓	✓

System model

In this section, we describe the different entities of a CoMEC network and the interactions among them.

Entities of CoMEC network

We consider a CoMEC network consisting of a set of collaborative edge servers (CESs), $\mathbb{E}$ and a set of mobile devices (MDs), $\mathbb{U}$, as shown in Fig. 2. Each mobile device $k \in \mathbb{U}$ is connected with one edge server $j \in \mathbb{E}$, which is termed as its primary edge server (PES). Let τ be the set of M tasks arrived at a PES from mobile devices. Each task $i \in \tau$ is denoted by a four-parameter tuple, $\langle b_i, \mathcal{B}_i, T_i^{max}, \delta_i \rangle$, where b_i is the input data size, $\mathcal{B}_i$ is the size of related data codes, T_i^{max} is the task deadline and δ_i is the task budget. In this work, data code is considered to consist of application-related program code, system settings, and related databases/libraries.

Each mobile device k has computational resources and each edge server j is considered to consist of both computational and cached resources. Table 3 contains major notations. A task generated from an MD can be executed either on the MD itself or at any edge server where edge servers are borrowing resources from the cloud while needed.

Collaboration among entities

Upon receiving a set of task requests, τ from the mobile devices, the PES communicates with the other CESs for task-related information and checks the availability of the resources, i.e., cached and computational resources required for the execution of the tasks. After getting the resource availability information, the PES runs the WOLVERINE task allocation decision algorithm and determines the appropriate resource providers to execute the tasks considering their requirements. If none of the servers has enough resources to complete a task, it is forwarded to the master cloud for execution, implementing a vertical collaborative computation environment.

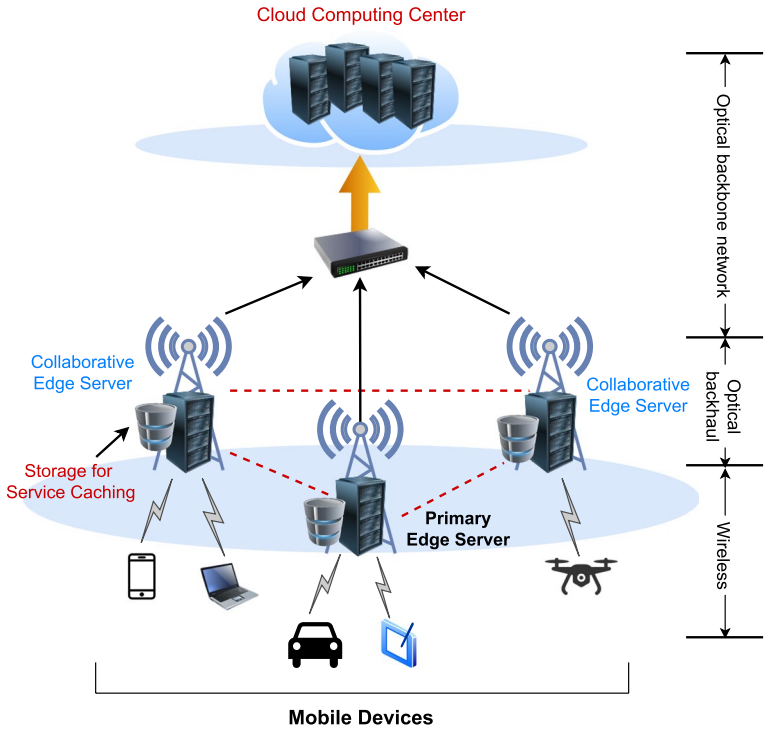


Fig. 2 The structure of CoMEC network over fiber-wireless connection

Table 3 Description of notations

Notation	Description
$\mathbb{U}$	Set of mobile devices in the system
τ, $\mathbb{E}$	Set of tasks and set of servers, respectively
c_i	Required CPU-cycle to complete task $i \in \tau$
μ_i^k	% of CPU-cycles allocated to task $i \in \tau$ by MD $k \in \mathbb{U}$
λ_{ij}	% of CPU-cycles allocated to task $i \in \tau$ by server $j \in \mathbb{E}$
B_{ij}	Radio bandwidth allocated to task i by server j
p^k	Transmission power of MD $k \in \mathbb{U}$
f^k	CPU-cycle frequency of MD $k \in \mathbb{U}$
f^j	CPU-cycle frequency of server $j \in \mathbb{E}$
b_i	Size of input data of task $i \in \tau$
$\mathcal{B}_i$	Size of the data code related to task $i \in \tau$
σ_{ij}	Cached resource availability for task i at server j
γ_j	Per unit CPU-cycle cost of server $j \in \mathbb{E}$
η_j	Per unit storage cost of server $j \in \mathbb{E}$
χ^w	Position vector of wolf $w \in P$
x_d^w	Position of wolf $w \in P$ at d^{th} dimension

Design details of WOLVERINE

In this section, we unfold different design components of WOLVERINE. First, we present a computational model of the proposed WOLVERINE system, then we formulate the task offloading problem as a multi-objective optimization problem; and finally, we devise a binary multi-objective grey wolf optimization-based solution.

Computational model of WOLVERINE

In this section, we unfold different design components of WOLVERINE. First, we present a computational model of the proposed WOLVERINE system, then we formulate the task offloading problem as a multi-objective optimization problem; and finally, we devise a binary multi-objective grey wolf optimization-based solution.

Figure 3 depicts the functional modules of the proposed WOLVERINE system, where an individual module is responsible for performing a specific function. The main functional modules of the PES can be grouped into two categories: the PES service module and the CES service module. The PES service module handles the task requests from the MDs and determines the optimal task offloading policy with the help of the CES service module. The responsibility of the CES service module is to manage collaboration between the PES and the CESs. Note that any collaborative edge server can work as a primary server by installing the PES service module to achieve the corresponding functionalities. The functionalities of each module are described below:

- **Task Profiler** receives the task-offloading requests from the MD first and then checks for the required

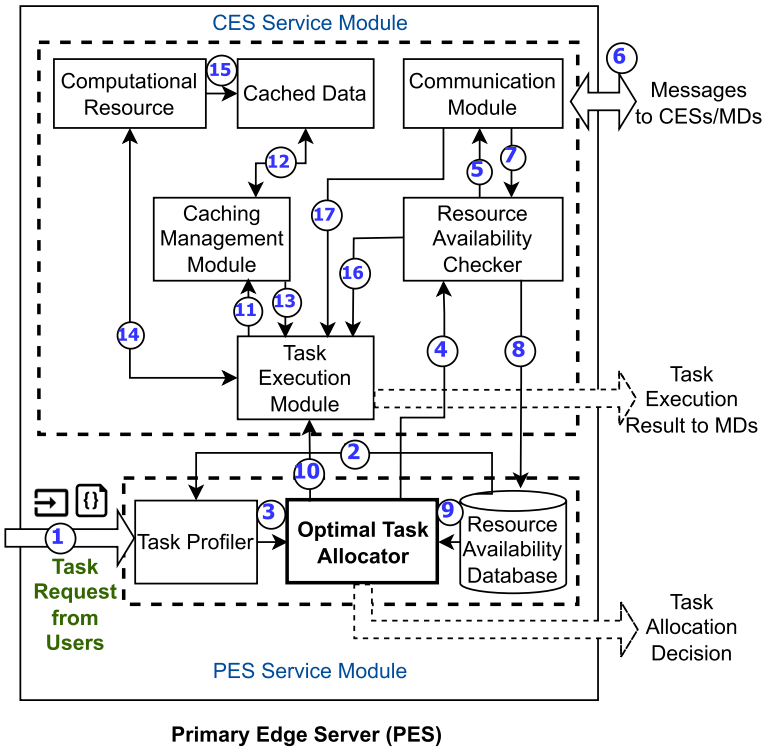


Fig. 3 Computational framework of WOLVERINE

cached resources for each task using the Resource Availability Database (Path 2) and propagates the task and resource data to the Optimal Task Allocator module (Path 3) for optimal resource allocation.

- **Optimal Task Allocator** is the core computational block of the PES service module. It collects the task's descriptions from the task profiler, queries the resource availability of the Collaborative Edge Servers (CES) to the Resource Availability Checker (Path 4) whose result comes through the Resource Availability Database (Path 5-6-7-8-9), formulates the WOLVERINE task offloading problem and communicates the associated task offloading decision vectors to the MDs.
- **Resource Availability Database** records the availability of the computational and cached resources of the CES that comes through the Communication Module and Resource Availability Checker (Path 6-7-8).
- **Resource Availability Checker** queries resources to other neighboring CESs and updates the cached and computational resources periodically or when triggered by the Optimal Task Allocator (Path 16).
- **Task Execution Module** executes the computational tasks offloaded to it by utilizing the available computational resources (Path 14) and cached data administered by Caching Management Module (Path 11-12-13).
- **Caching Management Module** supplies the cached data to the Task Execution Module from the Cached Data module (Path 12-13) and maintains the cached data repository by performing maintenance functions.
- **Cached Data Repository** stores the cached data code from the Computational Resource module for further use (Path 15).
- **Computational Resources** module stores the server's available resources, such as CPU cycle and memory, for usage by the Task Execution Module.
- **Communication Module** establishes collaboration among multiple edge servers and acts as a communication medium between the server and the MDs to share task data and computational results.

Multi-objective problem formulation

In this section, we calculate total latency T_{ij}, energy consumption E_{ij} and monetary cost C_{ij} for offloading task $i \in \tau$ to edge server $j \in \mathbb{E}$ or for local computation. Finally, we formulate the task offloading problem of WOLVERINE as a multi-objective optimization problem.

Calculation of T_{ij}

Two different cases for calculating T_{ij}:

In the first case, the mobile device executes the task locally, thus, experiencing no communication delay. So, the task computation delay, t_{ij}^k for executing task $i \in \tau$ on the mobile device $k \in \mathbb{U}$ locally is calculated as,

$$t_{ij}^k = \frac{c_i}{\mu_i^k \times f^k}. \tag{1}$$

Here, c_i is the number of computation cycles required to compute the task, μ_i^k is the ratio of CPU cycles allocated by k^{th} mobile device to complete i^{th} task and f^k is the CPU-cycle frequency of k^{th} mobile devices.

For the second case, the input data and/or data code are offloaded to the MEC servers. If the data code is cached at the offloading server, then only the input data needs to be transmitted; otherwise, the device sends the input data along with the code to the server. For wireless transmission between the mobile device and collaborative edge server that follows Orthogonal Multiple Access (OMA), we consider the Rayleigh channel, and the transmission rate is calculated as,

$$r_{ij} = B_{ij} \times \log_2(1 + \frac{p^k \times h^k}{N_0}), \tag{2}$$

where, B_{ij} is the allocated radio bandwidth, p^k is the transmission power, h^k is the channel gain ($k \in \mathbb{U}$) and N_0 is the variance complex of white Gaussian channel noise. Now, we calculate the communication latency, t_{ij}^c for offloading task i to edge server j as follows,

$$t_{ij}^c = \frac{\sigma_{ij} \times b_i + (1 - \sigma_{ij}) \times (b_i + \mathcal{B}_i)}{r_{ij}}, \tag{3}$$

where, $\sigma_{ij} \in \{0, 1\}$. Its value is 1 when the cached resources i.e., data code available in the offloading server, otherwise 0. Here, b_i and $\mathcal{B}_i$ denote the size of the input parameters and data code, respectively. Next, we calculate the execution time of task i at the edge server j as,

$$t_{ij}^e = \frac{c_i}{\lambda_{ij} \times f^j}, \tag{4}$$

where, λ_{ij} is the resource of server j allocated to task i and f^j is the total resource of the j^{th} MEC. Finally, we calculate the total latency for completing task i using the following equation:

$$T_{ij} = \begin{cases} t_{ij}^k & \text{if task is executed locally} \\ t_{ij}^c + t_{ij}^e & \text{if task is executed on server.} \end{cases} \tag{5}$$

When calculating execution latency for real-time computation-intensive applications in edge computing, addressing delivery or downloading latency is crucial. However, in this particular scenario, the emphasis is placed more on upload speeds and network latency rather than download times. Besides, the execution result has typically limited data size and thus it has negligible impact on resource parameters.

Calculation of E_{ij}

For calculating total energy consumption E_{ij}, two possible cases have been brought under consideration. In the first case, the mobile device executes the task locally. Hence, we consider only task computation energy and it is calculated as follows,

$$E_{ij}^k = \kappa \times \left(f^k\right)^2 \times c_i, \tag{6}$$

where, κ is a co-efficient that depends on device's chip architecture [17] and f^k is the CPU-cycle frequency of k^{th} mobile device.

For the second case, the task is executed at the server, hence, task computation energy is ignored. Thus, the energy the device expends due to transmitting input data and/or code to the MEC server is calculated as,

$$E_{ij}^c = p^k \times t_{ij}^c, \tag{7}$$

where, p^k is the power of k^{th} mobile device and t_{ij}^c is the time required to transmit i^{th} task to j^{th} server. Now, the total energy consumption for offloading task i to server j is calculated as,

$$E_{ij}^k = \begin{cases} E_{ij}^k & \text{if task is executed locally} \\ E_{ij}^c & \text{if task is executed in an edge server.} \end{cases} \tag{8}$$

The overall energy consumption can include the energy consumed for transmitting the tasks to the servers. We have prioritized device energy consumption owing to the limited battery resources and computational capabilities of user devices. As a result, energy consumed for executing tasks by servers has been less emphasized.

Calculation of C_{ij}

Similar to latency and energy, the calculation of monetary cost for task computation can also have two possible cases. If the device performs the task locally instead of offloading, then it incurs no monetary cost. In the case of offloading, the cost of computational resources, i.e., CPU cycle and/or storage resources, i.e., memory, sums up the total monetary cost. For executing i^{th} task at j^{th} server, the storage cost is calculated as follows,

$$v_{ij}^s = \sigma_{ij} \times (\eta_j \times \mathcal{B}_i). \tag{9}$$

Here, $\sigma_{ij} \in \{0, 1\}$ determines the availability of cached resource. If the value of σ_{ij} is 1, storage cost will be incurred

for the device; otherwise, no storage cost is required. η_j is the storage cost of per bit resource. Next, we calculate the cost of computing i^{th} task at j^{th} server as follows,

$$v_{ij}^{c} = \gamma_j \times c_i, \tag{10}$$

where, γ_j is the unit CPU cycle cost of server j. Finally, the total monetary cost for executing task i at server j can be calculated as,

$$C_{ij} = \begin{cases} 0 & \text{if task is executed locally} \\ v_{ij}^{c} + v_{ij}^{s} & \text{if task is executed on server} \end{cases} \tag{11}$$

We have not considered cloud servers in our problem formulation. Although cloud server adds significant benefits related to scalability, server-health management, backup, and service provisioning capabilities, they can create hindrances in real-time application environments due to long-distance communication where exceptional QoE needs to be achieved. Uploading and executing tasks in the cloud require extra latency and energy, which impeded performance. Hence execution of tasks in user mobile devices and edge servers adds leverage to network performance. Cloud servers are typically utilized within an edge server network only when all other edge resources are overwhelmed or during network malfunctions.

Objective function formulation

Our aim is to execute each task $i \in \tau$ at local or remote resource $j \in \mathbb{E}$ so as to minimize the total execution latency, energy expenditure, and incurred monetary cost. Thus, WOLVERINE formulates the task execution problem as a multi-objective minimization problem as follows,

$$Minimize\ \{T(\overrightarrow{\chi}_w), E(\overrightarrow{\chi}_w), C(\overrightarrow{\chi}_w)\} \tag{12}$$

where,

$$T(\overrightarrow{\chi}_w) = \sum_{i\in\tau}\sum_{j\in\mathbb{E}}(X_{ij} \times T_{ij}), \tag{13}$$

$$E(\overrightarrow{\chi}_w) = \sum_{i\in\tau}\sum_{j\in\mathbb{E}}(X_{ij} \times E_{ij}), \tag{14}$$

$$C(\overrightarrow{\chi}_w) = \sum_{i\in\tau}\sum_{j\in\mathbb{E}}(X_{ij} \times C_{ij}). \tag{15}$$

Here, X_{ij} is a binary decision variable whose value is 1 if task i is allocated to edge server j, otherwise 0. And $X_{ij} \in \overrightarrow{\chi}_w$, where $\overrightarrow{\chi}_w$ is a D-dimensional vector, $\overrightarrow{\chi}_w = (x_1, x_2, ..., x_D)$. Each entry $x_d \in \overrightarrow{\chi}_w$ corresponds to the aforementioned decision variable $X_{ij}, \forall i \in \tau, \forall j \in \mathbb{E}$. $T(\overrightarrow{\chi}_w)$, $E(\overrightarrow{\chi}_w)$, and $C(\overrightarrow{\chi}_w)$ denotes the objective functions related to task execution latency, execution energy, and monetary cost respectively. Equation (12), which is a multi-objective linear optimization problem, is subject to the following constraints:

- **Assignment Constraint:** Task will be executed in either an edge server or in the user device. No partial assignment of tasks to multiple servers will be done.

$$\sum_{i\in\tau}\sum_{j\in\mathbb{E}} X_{ij} \leq 1 \tag{16}$$

- **Budget Constraint**: Constraint (17) denotes that the monetary cost of task t for executing it to server j cannot exceed the task budget, δ_i.

$$C_{ij} \leq \delta_i, \quad \forall i \in \tau, j \in \mathbb{E} \tag{17}$$

- **Energy Constraint**: Constraint (18) refers to the energy expenditure of a device in executing a task is limited by a threshold, E_i^{max}.

$$E_{ij} \leq E_{ij}^{max}, \quad \forall i \in \tau, j \in \mathbb{E} \tag{18}$$

- **Latency Constraint**: Constraint (19) denotes that a task t needs to be completed within its deadline, T_i^{max}.

$$T_{ij} \leq T_i^{max}, \quad \forall i \in \tau, j \in \mathbb{E} \tag{19}$$

Theorem 1 *The WOLVERINE task offloading problem formulated in Eq. (12) is NP-hard.*

Proof

The WOLVERINE task offloading problem aims at minimizing three objectives, yielding a set of Pareto optimal solutions. The optimization problem in Eq. (12) can be regarded as an assignment problem. To prove the NP-hardness of the WOLVERINE task offloading problem, we first convert Generalized Assignment Problem (GAP), a well-known NP-hard problem [39], into a multi-objective problem. The GAP assigns M tasks to N agents to minimize the overall assignment costs as follows:

$$Minimize \sum_{i\in M}\sum_{j\in J}(\mathbb{C}_{ij} \times X_{ij}) \tag{20}$$

Subject to:

$$\sum_{i\in M} A_{ij} \times X_{ij} \leq \mathbb{B}_j, \quad \forall j \in N \tag{21}$$

$$X_{ij} \in \{0, 1\}, \quad \forall i \in M, \forall j \in N \tag{22}$$

$$\sum_{j \in N} X_{ij} = 1, \quad \forall i \in M \tag{23}$$

Here, $\mathbb{C}$ indicates the assignment cost of task $m \in M$ to an agent $n \in N$, A is the resource capacity function that indicates the resource used by task $m \in M$ and $\mathbb{B}$ indicates the available capacity of an agent. To convert GAP to a multi-objective assignment problem, we first consider a bi-objective assignment problem where resource and cost constraints of GAP is to be satisfied by converting the three objectives of WOLVERINE to a single one as follows:

Minimize

$$\overrightarrow{Z} = (z_1, z_2) = \left(\sum_{i \in \tau} u \left(\sum_{j \in \mathbb{E}} \mathbb{R}_{ij} X_{ij} - \mathbb{B}_i \right), \sum_{i \in \tau} \sum_{j \in \mathbb{E}} \left(X_{ij} \times \mathcal{Z}_{ij} \right) \right) \tag{24}$$

where,

$$\begin{aligned} \mathcal{Z}(\tau, \mathbb{E}) &= \epsilon_1 T(\tau, \mathbb{E}) + \epsilon_2 E(\tau, \mathbb{E}) + \epsilon_3 C(\tau, \mathbb{E}) \\ &= \epsilon_1 \sum_{i \in \tau} \sum_{j \in \mathbb{E}} (X_{ij} \times E_{ij}) + \epsilon_2 \sum_{i \in \tau} \sum_{j \in \mathbb{E}} (X_{ij} \times T_{ij}) + \epsilon_3 \sum_{i \in \tau} \sum_{j \in \mathbb{E}} (X_{ij} \times C_{ij}) \end{aligned} \tag{25}$$

Subject to:

$$\sum_{j \in \mathbb{E}} X_{ij} \leq n_j, \quad \forall j \in \mathbb{E} \tag{26}$$

$$X_{ij} \in \{0, 1\}, \quad \forall i \in \tau, \forall j \in \mathbb{E} \tag{27}$$

$$\sum_{j \in \mathbb{E}} X_{ij} = 1, \quad \forall i \in \tau \tag{28}$$

Here, the value of ϵ_i is chosen in such a way that minimizing $\mathcal{Z}_{ij}$ yields the same result as the multi-objective functions. The function $u(X)$ is defined as, $u(X) = 1$ if $x \geq 0$ and 0 otherwise.

Note that, we do not consider resource limitation constraints of GAP as the constraints of the multi-objective optimization problem, rather we consider it as an objective to be optimized. If the resource limitation constraints are satisfied, then z_1 is equal to zero and the cost of assignment z_2 will be considered. If there exists a better solution in GAP, a better solution also exists in the corresponding multi-objective problem. We consider two feasible solution, $\overrightarrow{Z_1}$ and $\overrightarrow{Z_2}$ where cost $z_1 < z_2$. These two costs produce solutions (0,z_1) and (0,z_2) in multi-objective assignment problems. If we consider lexicographical minimum, then $z_1 < z_2$. Hence (0,z_1) is a better solution. Thus, GAP is convertible to a multi-objective assignment problem.

Therefore, it is shown that GAP can be converted to a multi-objective optimization problem. Since GAP is a well-known NP-hard problem, the WOLVERINE task offloading problem is also an NP-hard one. □

Meta-heuristic task offloading

As the number of MDs or servers increases, the WOLVERINE system experiences exponential growth in execution time. Many 5G applications can not tolerate a single second of delay. The proposed WOLVERINE framework attempts to optimize multiple objectives, such as minimizing latency, reducing energy consumption, and minimizing monetary costs. These objectives can be conflicting, meaning that improving one objective may degrade the other. Pareto optimal solutions help find a set of solutions where no single objective can be improved without worsening at least one other objective. Evolutionary algorithms help in solving problems that involve Pareto-optimality as the solution choice is based on the population approach [31]. Therefore, in this section, we develop a smart task offloading policy using Binary Multi-Objective Grey Wolf Optimization that determines the suitable set of resources to allocate the computational tasks in polynomial time.

Preliminaries

The Grey Wolf Optimization (GWO) [40] is a bio-inspired meta-heuristic algorithm that is designed based

on the social leadership and hunting techniques found in grey wolves. To mathematically model the social hierarchy of the wolves, the fittest solution is considered the alpha (α) wolf. The second and third best solutions are named beta (β) and delta (δ) wolves, respectively. The leader selection and position updating of the rest of the search agents are done in each iteration, eventually converging to a set of Pareto-optimal solutions. Binary Multi-objective Grey Wolf Optimization (BMOGWO) is a special variant of MOGWO that allows search agents to move in a binary space instead of a continuous spectrum [41]. In our specific case, where we aim to optimize execution time, energy consumption by devices, and monetary cost simultaneously, the BMOGWO algorithm demonstrates superior performance compared to other evolutionary algorithms such as MOPSO, BAT, and WHALE optimization algorithms [42–44] for tackling multi-objective problems, efficiently addressing the optimization of multiple objectives concurrently. It also outperforms Ant-Colony Optimization (ACO) and Whale Optimization (WO) in scenarios where task offloading is required to edge servers [34]. The BMOGWO also surpasses other evolutionary algorithms in scenarios where Pareto-optimal solutions are generated due to better performance in the exploration of solution space and prevention of convergence to local optima [45].

Defining the position vector

We consider a population of wolves denoted by P where each wolf, $w \in P$ represents a candidate solution [46]. The position of a wolf w in the search space is denoted by a D-dimensional binary position vector $\overrightarrow{\chi}_w$ where $D = \tau \times \mathbb{E}$. The D-dimensional vector is denoted by $\overrightarrow{\chi}_w = (x_1, x_2, ..., x_D)$ where each entry $x_d \in \overrightarrow{\chi}_w$ corresponds to a decision variable $X_{ij}, \forall i \in \tau, \forall j \in \mathbb{E}$ such that, $d = (i-1) \times \mathbb{E} + j$.

Updating positions of the wolves

In GWO, the position of each ω wolf is updated by considering the positions of α, β, and δ wolves. Let $\overrightarrow{\chi}_\alpha$, $\overrightarrow{\chi}_\beta$, $\overrightarrow{\chi}_\delta$ and $\overrightarrow{\chi}_\omega$ denote the position of α, β, δ and ω wolves, respectively. Now we calculate the distance of the ω wolf from the other three leader wolves as follows.

$$\overrightarrow{\mathcal{D}_\alpha} = \mid \overrightarrow{C_1}.\overrightarrow{\chi_\alpha} - \overrightarrow{\chi_\omega} \mid, \tag{29}$$

$$\overrightarrow{\mathcal{D}_\beta} = \mid \overrightarrow{C_2}.\overrightarrow{\chi_\beta} - \overrightarrow{\chi_\omega} \mid, \tag{30}$$

$$\overrightarrow{\mathcal{D}_\delta} = \mid \overrightarrow{C_3}.\overrightarrow{\chi_\delta} - \overrightarrow{\chi_\omega} \mid . \tag{31}$$

Here, $\overrightarrow{C}$ is a position vector with values in the range [0, 2]. The position vector associates weight to each prey item, in our case, the three best solutions. The value of C is chosen randomly to favor exploration by introducing randomness in the algorithm's behavior. This vector controls the effect of prey, in this case, the effect of the three best solutions on the updating search agents. $\mid \overrightarrow{C} \mid > 1$ emphasizes the effect of best solutions more on the ω wolves whereas $\mid \overrightarrow{C} \mid < 1$ de-emphasizes the effect. This prevents local optimum convergence and ensures that the entire search space is covered. Besides, the random selection of values in C emphasizes exploration not only in the initial stages but also during the final iterations [40]. The value of C is determined as $\overrightarrow{C} = 2.\overrightarrow{r_2}$, where $\overrightarrow{r_2} \in [0, 1]$. Now the updated position of the ω wolf with respect to alpha, beta, and delta wolves is calculated as follows.

$$\overrightarrow{\chi}_{\alpha,\omega} = \overrightarrow{\chi}_\alpha - \overrightarrow{A_1}.(\overrightarrow{\mathcal{D}_\alpha}), \tag{32}$$

$$\overrightarrow{\chi}_{\beta,\omega} = \overrightarrow{\chi}_\beta - \overrightarrow{A_2}.(\overrightarrow{\mathcal{D}_\beta}), \tag{33}$$

$$\overrightarrow{\chi}_{\delta,\omega} = \overrightarrow{\chi}_\delta - \overrightarrow{A_3}.(\overrightarrow{\mathcal{D}_\delta}). \tag{34}$$

Here, $\overrightarrow{A}$ is the co-efficient vector that governs convergence or divergence towards the prey, or the best solutions, and has values in the range [-1,1]. The formula for calculating A's value is $\overrightarrow{A} = 2\overrightarrow{a}.\overrightarrow{r_1} - \overrightarrow{a}$, where $\overrightarrow{r_1} \in [0, 1]$. The search space exploration is managed using the $\overrightarrow{a}$ parameter. By averaging $\overrightarrow{\chi}_{\alpha,\omega}$, $\overrightarrow{\chi}_{\beta,\omega}$ and $\overrightarrow{\chi}_{\delta,\omega}$ in $\overrightarrow{\chi}_w$, the ultimate position of the ω wolf is now determined.

Note that each entry $x_d \in \overrightarrow{\chi}_w$ corresponds to a binary decision variable of the MOLP problem and is only allowed to have a value of either 0 or 1 as follows.

$$x_d = \begin{cases} 1 & \text{if } sigmoid(x_d) \geq rand() \\ 0 & \text{otherwise} \end{cases} \tag{35}$$

where, $sigmoid(x_d)$ is defined as,

$$sigmoid(x_d) = \frac{1}{1 + e^{-x_d}}. \tag{36}$$

Here, the *rand*() function provides a uniformly distributed random number in the range of [0, 1] that improves search space exploration with the goal of avoiding local optima. The convergence and diversity of the Pareto-front generated by MOGWO for Pareto-optimal solutions in tri-objective problem are higher than that of Multi-Objective Particle Swarm Optimization (MOPSO) [45]. Here, convergence indicates how close the obtained solutions are to the true Pareto-front. Diversity demonstrates how thoroughly the search space has been explored. It shows how much an algorithm is comparing

the trade-offs and setting the wide range of options. Higher diversity indicates a greater number of options have been explored through a different balance between the objective parameters. Grey-Wolf Optimization strikes a balance between the two of these. It converges toward the true Pareto-front by iteratively computing the solutions. As the algorithm progresses, the positions of α, β, and δ wolves are updated based on their fitness values. These three best solutions found so far guide the search process toward finding better solutions and helps to converge towards Pareto-front through optimal trade-offs. The exploration and randomness of Grey Wolf Optimization prevent convergence to local optima and provides a better exploration of a wide range of trade-offs.

Algorithm 1 Archive controller

Input: X, $\mathcal{A}$, P
Output: $\mathcal{A}$
1: **for all** $w \in P$ **do**
2: $\Omega \leftarrow \emptyset$
3: $flag \leftarrow 0$
4: **for all** $\rho \in \mathcal{A}$ **do**
5: **if** $(\overrightarrow{\chi}_w.T < \rho.T \;\&\; \overrightarrow{\chi}_w.E \leq \rho.E \;\&\; \overrightarrow{\chi}_w.C \leq \rho.C) \;||\; (\overrightarrow{\chi}_w.T \leq \rho.T \;\&\; \overrightarrow{\chi}_w.E < \rho.E \;\&\; \overrightarrow{\chi}_w.C \leq \rho.C) \;||\; (\overrightarrow{\chi}_w.T \leq \rho.T \;\&\; \overrightarrow{\chi}_w.E \leq \rho.E \;\&\; \overrightarrow{\chi}_w.C < \rho.C)$ **then**
6: $\Omega \leftarrow \Omega \cup \rho$
7: **else if** $(\rho.T < \overrightarrow{\chi}_w.T \;\&\; \rho.E \leq \overrightarrow{\chi}_w.E \;\&\; \rho.C \leq \overrightarrow{\chi}_w.C) \;||\; (\rho.T \leq \overrightarrow{\chi}_w.T \;\&\; \rho.E < \overrightarrow{\chi}_w.E \;\&\; \rho.C \leq \overrightarrow{\chi}_w.C) \;||\; (\rho.T \leq \overrightarrow{\chi}_w.T \;\&\; \rho.E \leq \overrightarrow{\chi}_w.E \;\&\; \rho.C < \overrightarrow{\chi}_w.C)$ **then**
8: $flag = 1$
9: **end if**
10: **end for**
11: **if** $flag == 0$ **then**
12: UpdateArchive($\mathcal{A}$, Ω)
13: **end if**
14: **end for**
15: **return** $\mathcal{A}$

Controlling the archive

For incorporating multi-objective optimization in GWO, an archive of fixed size is used. It is a simple storage for storing or retrieving Pareto-dominant solutions obtained so far, which is shown in Algorithm 1. In line 1, for each $w \in P$, a set Ω is initialized that stores the archive solutions dominated by $\overrightarrow{\chi}_w$. A flag is also initialized to check if any solution from the archive dominates $\overrightarrow{\chi}_w$. Line 5 checks for the archive members dominated by $\overrightarrow{\chi}_w$ and the dominated members are added to Ω. Line 7 checks the opposite and sets the flag to 1. In case there is no archive member that dominated $\overrightarrow{\chi}_w$, i.e., flag = 0, the archive is updated using procedure $UpdateArchive(\mathcal{A}, \Omega)$ in line 12. Lines 2-13 iterate for every member of the population and the updated archive is returned.

In Algorithm 2, $UpdateArchive(\mathcal{A}, \Omega)$ procedure is summarized. In lines 2-4, the dominated solutions are removed from the archive. The capacity of the archive is checked in line 5. If it is not full, then the current non-dominated solution is added to the archive in line 6; otherwise, the solution from the most crowded segment is removed and the current non-dominated solution is added to the archive in line 9. In line 12, if a particular solution is an outlier, the grid is updated adaptively to cover the new solution.

Adaptive grid mechanism

An adaptive grid made of hypercubes [47] is generated using the archive, where the dimension of each hypercube is equal to the number of optimization objectives. The grid mechanism divides the objective space of the problem into a grid. Each hypercube is interpreted as a geographical region that contains the solutions [47]. For our WOLVERINE task offloading problem, which has three objectives, therefore, the adaptive grid consists of three-dimensional hypercubes. The boundary of the objective/target space at t-th iteration is determined as ($minT_t, minE_t, minC_t$ and $maxT_t, maxE_t, maxC_t$). Now, we calculate the modulus of the grid using the same approach [47] as follows,

$$\Delta T_t = \frac{maxT_t - minT_t}{M}, \tag{37}$$

$$\Delta E_t = \frac{maxE_t - minE_t}{M}, \tag{38}$$

$$\Delta C_t = \frac{maxC_t - minC_t}{M}. \tag{39}$$

Here, M is an integer that determines the number of segments in each dimension of the objective space. Therefore, the total number of hypercubes is M^3.

We employ a strategy in which non-dominated solutions are removed from the most crowded segments of the archive and leader selection is performed from the less crowded segments [45]. Both of these operations are based on probabilities to avoid local optima in search spaces. The solution density in each segment plays an important role in calculating these probabilities [47]. The more non-dominated solutions there are in a segment, the higher the probability of removing one solution and the lower the probability of choosing a leader. The probability of choosing the i-th segment to remove a solution is calculated as follows:

$$\mathcal{P}_i = \frac{N_i}{c}, c = \max(N_i) \tag{40}$$

where N_i is the number of obtained pareto-optimal solutions in i-th segment. Note that Eq. (40) assigns a higher probability to a crowded segment. On the other hand, the probability of selecting a leader from the archive is calculated in the opposite manner. The roulette-wheel approach is used for the selection based on the likelihood for each hypercube [45], as expressed by the following equation:

$$\mathcal{Q}_i = \frac{1}{N_i + 1}. \tag{41}$$

From Eq. (41), it is clear that a segment with fewer solutions has a higher probability of being chosen as the leader.

Algorithm 2 Algorithm for updating archive

```
1: procedure UpdateArchive(A, Ω)
2:   for all ρ' ∈ Ω do
3:     A ← A \ ρ'
4:   end for
5:   if A is not full then
6:     A ← A ∪ χ⃗_w
7:   else if A is full then
8:     Remove one solution from the most crowded segment using Eq. (40)
9:     A ← A ∪ χ⃗_w
10:  end if
11:  if any newly added solution to A is located outside the grid then
12:    Update grid to cover the new solution
13:  end if
14: end procedure
```

Algorithm 3 BMOGWO based task offloading

Input: Set of tasks τ, Set of servers $\mathbb{E}$
Output: Task allocation matrix X

1: Initialize parameters a, A and C
2: $\mathcal{A} \leftarrow \emptyset$
3: **for all** $w \in P$ **do**
4: Initialize random position vector $\overrightarrow{\chi}_w$
5: Calculate fitness values of $\overrightarrow{\chi}_w$ using Eqs. (13)-(15)
6: **end for**
7: Get updated archive using Algorithm 1
8: Select leaders $\overrightarrow{\chi}_\alpha$, $\overrightarrow{\chi}_\beta$ and $\overrightarrow{\chi}_\delta$ from $\mathcal{A}$ using Eq. (41)
9: $t \leftarrow 0$
10: **while** $(t < I_{max})$ **do**
11: **for all** $w \in P$ **do**
12: **for all** $x_d \in \overrightarrow{\chi}_w$ **do**
13: Update x_d by Eqs. (29)-(35)
14: **end for**
15: **end for**
16: Update parameters a, A and C
17: **for all** $w \in P$ **do**
18: Calculate fitness values of $\overrightarrow{\chi}_w$ using Eqs. (13)-(15)
19: **end for**
20: Get updated archive using Algorithm 1
21: Select leaders $\overrightarrow{\chi}_\alpha$, $\overrightarrow{\chi}_\beta$ and $\overrightarrow{\chi}_\delta$ from $\mathcal{A}$ using Eq. (41)
22: $t = t + 1$
23: **end while**
24: **for all** dimension $x_d \in \overrightarrow{\chi}_\alpha$ **do**
25: $i \leftarrow \lceil \frac{d}{N} \rceil$, $j \leftarrow \{d - (i-1) \times N\}$
26: $X_{i,j} \leftarrow x_d$
27: **end for**
28: **return** X

BMOGWO-based task execution

The steps of the BMOGWO-based task execution scheme of WOLVERINE are presented in Algorithm 3. First, we initialize the archive in line 2. Next, we initialize a population of random position vectors and calculate their fitness values in lines 4 and 5. The archive is populated with a set of non-dominated solutions generated using Algorithm 1 in line 7. Line 8 selects three different leaders using a grid mechanism. For each dimension of every wolf, the positions are updated in line 13. Parameters a, A, and C are updated in line 16. Next, we calculate the fitness values of the updated position vectors in line 18 and update the archive with updated positions using the Algorithm 1 in line 20. Hence, from the updated archive, three new leaders are selected using Eq. (41) in line 21. Lines 11-21 repeat until a maximum number of iterations I_{max} is reached. Finally, the value of entry x_d of the best solution $\overrightarrow{\chi}_\alpha$ is assigned to the corresponding decision variable in lines 25-26 and the decision vector X is returned.

Complexity analysis

In this section, we analyze the complexity of the three algorithms used in WOLVERINE. In Algorithm 2, Line 3 is enclosed within a loop that iterates $|\mathcal{A}|$ times in the worst case. Line 8 requires M^3 time. The rest of the statements are of constant time complexity. Thus, the overall complexity of Algorithm 2 is $O(|\mathcal{A}| + M^3)$. Next, we define the complexity of Algorithm 1. Lines 5-9 are enclosed within a loop that iterates $|\mathcal{A}|$ times. Line 12 updates the archive using Algorithm 2 that takes $O(|\mathcal{A}| + M^3)$. Lines 2-13 are also enclosed within a loop that takes $|P|$ times. Hence, the computational complexity of Algorithm 1 is $O(|P| \times (|\mathcal{A}| + M^3))$. Finally, we analyze the complexity of Algorithm 3. Lines 4 and 5 are enclosed within a loop that iterates for $|P|$ times. Line 7 updates the archive that requires $O(|P| \times (|\mathcal{A}| + M^3)))$ times. Line 13 is enclosed within a nested loop that iterates $|P| \times |\overrightarrow{\chi}|$ times. Line 18 is enclosed in another loop that iterates for $|P|$ times. Line 20 again calls Algorithm 2. Lines 11-22 are also enclosed within a loop that iterates for I_{max} times. The rest of the algorithm takes constant time to run. Thus, the total computational complexity of Algorithm 3 is $O(|P| \times (|\mathcal{A}| + M^3 + I_{max} \times (\overrightarrow{\chi} + |\mathcal{A}| + M^3))) \approx O(M^3)$.

Convergence analysis

In this section, we analyze the convergence of the developed WOLVERINE system, which is measured using Inverted Generational Distance (IGD). IGD is a metric used for assessing the quality of a set of solutions produced by an optimization algorithm, particularly in the context of multi-objective optimization. It measures the convergence and diversity of the obtained solutions concerning the true Pareto front, which represents the optimal trade-off between conflicting objectives. The IGD

metric calculates the average distance from each point in the obtained solution set to the nearest point in the true Pareto front. A lower IGD value indicates a better convergence and diversity of the obtained solutions.

If the IGD value between the obtained Pareto front ρ and the true Pareto front $\rho*$ is $IGD(\rho, \rho*)$, then the convergence ratio (CR) $\mathcal{C}$ can be defined as,

$$\mathcal{C} = \frac{IGD(\rho_{t+1}, \rho*) - IGD(\rho_t, \rho*)}{IGD(\rho_t, \rho*)}, \tag{42}$$

where, ρ_t and ρ_{t+1} denote the Pareto Front value after t and $(t+1)$ iterations, respectively.

Theorem 2 *The convergence ratio $\mathcal{C}$ of the developed BMOGWO-based WOLVERINE system is bounded by $g(P, \overrightarrow{C}, \overrightarrow{A}, \tau, \mathbb{U}, \mathbb{E}, t)$.*

Proof

This proof can be done by inductive hypothesis. We need to proof that CR $\mathcal{C} \leq g(P, \overrightarrow{C}, \overrightarrow{A}, \tau, \mathbb{U}, \mathbb{E}, t)$. It can be mathematically denoted as,

$$IGD(\rho_{t+1}, \rho*) - IGD(\rho_t, \rho*) \leq g(P, \overrightarrow{C}, \overrightarrow{A}, \tau, \mathbb{U}, \mathbb{E}, t) \times IGD(\rho_t, \rho*). \tag{43}$$

Here, $g(P, \overrightarrow{C}, \overrightarrow{A}, \tau, \mathbb{U}, \mathbb{E}, t)$ indicates the upper bound of the solution, where the solution of the algorithm is the farthest from the true Pareto front ρ^*, which can be mathematically represented as follows,

$$g(P, \overrightarrow{C}, \overrightarrow{A}, \tau, \mathbb{U}, \mathbb{E}, t) = \max_{\hat{\varrho} \in \rho_t} \min_{\varrho^* \in \rho^*} d(\hat{\varrho}, \varrho^*), \tag{44}$$

where, $d(\hat{\varrho}, \varrho^*)$ denotes the distance between the two solutions $\hat{\varrho}$, and ϱ^* in the solution space. The IGD value of solution ρ_t after iteration t can be calculated similar to [48] as follows,

$$IGD(\rho_{t+1}, \rho^*) = \frac{1}{|\rho^*|} \sum_{\hat{\varrho} \in \rho_t} \min_{\varrho^* \in \rho^*} d(\hat{\varrho}, \varrho^*), \tag{45}$$

Basis Step: Let us assume that ρ_0 denotes the initial Pareto front approximation and $IGD(\rho_0, \rho*)$ be the initial IGD value. Then, Eq. (43) can be modified as follows,

$$IGD(\rho_1, \rho*) - IGD(\rho_0, \rho*) \leq g(P_0, \overrightarrow{C_0}, \overrightarrow{A_0}, \tau, \mathbb{U}, \mathbb{E}, 0) \times IGD(\rho_0, \rho*), \tag{46}$$

where, $P_0, \overrightarrow{C_0}, \overrightarrow{A_0}$ denote the initial population size, position vector, and co-efficient vector, respectively. Equation (46) confirms that the induction hypothesis holds true for the base step.

Inductive Step: Assume that the theorem holds up to the t-th iteration i.e., $IGD(\rho_t, \rho*) \leq g(P, \overrightarrow{C}, \overrightarrow{A}, \tau, \mathbb{U}, \mathbb{E}, t)$. Now, we need to express the improvement in performance from iteration t to $t+1$, which can be mathematically represented as,

$$\begin{aligned} &IGD(\rho_{t+1}, \rho*) - IGD(\rho_t, \rho*) \leq h(P, \overrightarrow{C}, \overrightarrow{A}, \tau, \mathbb{U}, \mathbb{E}, t), \\ &IGD(\rho_{t+1}, \rho*) \leq h(P, \overrightarrow{C}, \overrightarrow{A}, \tau, \mathbb{U}, \mathbb{E}, t) + IGD(\rho_t, \rho*) \end{aligned} \tag{47}$$

where, $h(.)$ denotes the improvement function. As $IGD(\rho_t, \rho*) \leq g(P, \overrightarrow{C}, \overrightarrow{A}, \tau, \mathbb{U}, \mathbb{E}, t)$, therefore, $IGD(\rho_{t+1}, \rho*) \leq g(P, \overrightarrow{C}, \overrightarrow{A}, \tau, \mathbb{U}, \mathbb{E}, t)$ in Eq. (47). Thus it confirms that Eq. (43) holds true for all t and convergence ratio $\mathcal{C}$ of the developed WOLVERINE system is bounded by $g(P, \overrightarrow{C}, \overrightarrow{A}, \tau, \mathbb{U}, \mathbb{E}, t)$. □

Performance evaluation

In this section, the performance of our proposed multi-objective task offloading with the caching approach is compared with some of the existing strategies in the literature: MGBD [4], iRAF [17] and MOEA/D [19]. The work presented in [4] focuses on jointly addressing the content caching, computation offloading, and resource allocation problem to reduce users' overall task execution time. An AI-driven resource allocation framework (iRAF) has been developed in [17] to tackle intricate resource allocation problems by considering current network conditions and parameters to optimize either execution time or energy consumption. In a multi-user and multi-server task offloading environment, a tri-objective problem is addressed in [19], where time, device energy, and cost are optimized using Multi-Objective Evolutionary Algorithm (MOEA/D). However, caching the data codes has not been considered in this work. The environmental setup, performance metrics, and results are discussed below.

Environmental setup

We have implemented our proposed algorithm and performed empirical numerical evaluation using Python 3.6.0 [49]. For evaluation purposes, we consider a

scenario where a stationary edge server is centered in a $1000 \times 1000m^2$ urban area. A number of collaborative edge servers are randomly located around the primary edge server and several mobile devices are connected to the edge servers. The path loss model between the mobile devices and servers is assumed to follow a log-normal distribution. In addition to the above metrics, we model packet loss on each path using the Gilbert loss model [50] and the channels handle the re-transmission of lost packets using TCP protocol. 20 channels are employed, each with a bandwidth of 2MHz. Our study is focused on real-time, delay-sensitive, and computation-intensive applications, including interactive video gaming, AR/VR applications, medical image processing, and face recognition. The task arrivals pattern follows a Poisson distribution. The whole experiment has been run 50 times and the average of all these results is taken to plot each graph. Major environment setup parameters used in this paper are shown in Table 4. In our simulation setting environment, we have ensured that resources are allocated proportionately across different systems. All the methods from the literature were implemented and performance metrics data were collected in a system environment consistent with that of ours.

Performance metrics

We have measured the performance of our algorithm based on the following metrics:

- ***Average latency*** is defined as the ratio of the total delay experienced by the tasks to the number of tasks.
- ***Average Energy Consumption*** is the average amount of energy consumed by each edge device.
- ***Average Cost Savings*** is calculated as the difference between a device's budget and the monetary cost paid by it divided by the number of tasks. The higher value indicates a higher system performance.
- ***Task Completion Reliability (TCR)*** is the ratio of the number of tasks completed to the submitted ones.

Table 4 Evaluation parameters

Parameter	Value
Simulation area	$1000 \times 1000m^2$
Number of mobile devices	[10-260]
Number of collaborative edge servers	[2-12]
The transmit power	[32*mW*-197*mW*]
A coefficient that affects local energy consumption (κ)	10^{-26}
The required CPU cycles to complete task	[6×10^9-9×10^{10}]Hz
The CPU-cycle frequency of MD	300MHz
The computation capability of edge servers	[3.19GHz-19.14GHz]
The bandwidth of one channel	2MHz
Size of input data	[3MB-50MB]
Number of Iteration (I_{max})	50
Population size	50

Result analysis

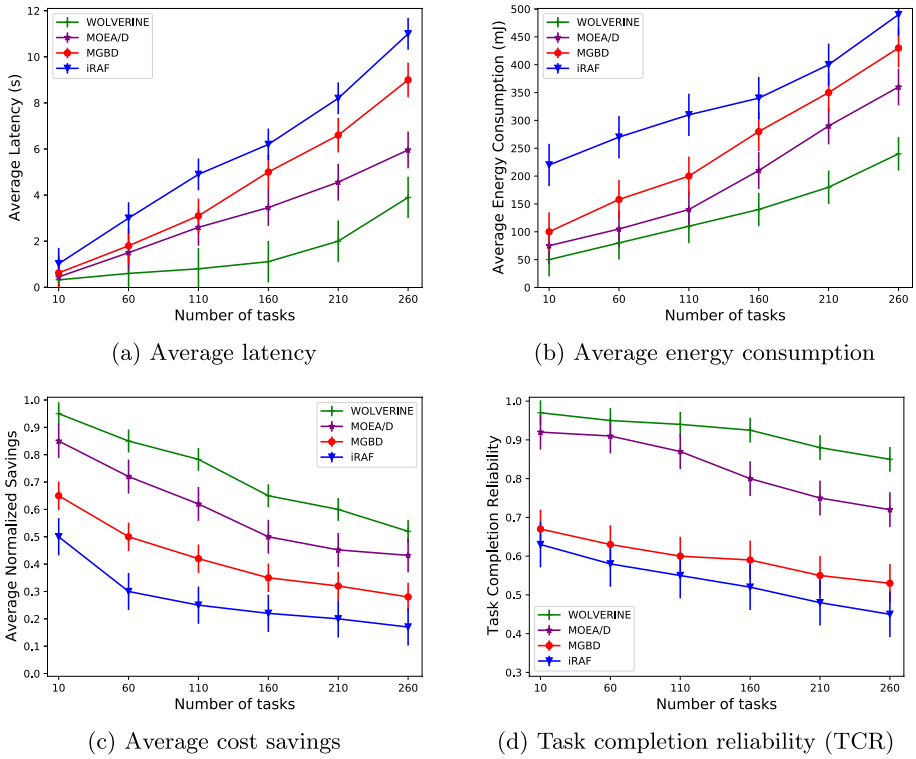
In this section, we have discussed the performance of our proposed system by varying the number of tasks, the number of servers in the system, and the average computation power per task.

Impact of a varying number of tasks

In this experiment, we vary the number of tasks of the overall network system from 10 to 250 and keep the number of servers fixed at 12. The result and comparison are shown in Fig. 4.

Figure 4(a) shows that as the number of tasks increases, the average latency also increases. Initially, latency increases slowly for a smaller number of tasks. However, as the number of tasks exceeds 160, latency increases exponentially. Latency is lower in MGBD and WOLVERINE cases than the iRAF because the former two have implemented caching. In the case of MOEA/D, the performance is close to the WOLVERINE. A single mobile device user decomposes an application into multiple independent sub-tasks and offloads them to various servers, depending on resource availability. However, as the sub-tasks are executed in parallel, the total latency considered for completing a task is the maximum latency among the sub-tasks, and a risk of high delay remains in case the system reaches its saturation point. Besides, the absence of server-to-server collaboration makes it difficult to share sub-tasks. Our proposed WOLVERINE exploits both collaborative edge computing and caching. Therefore, if the required data for a specific task is not cached at a server or computational resources are not present, the server can pass the task to another collaborative server where the task data is cached already, which decreases the service delay significantly. Therefore, our proposed WOLVERINE outperforms the state-of-the-art approaches.

The impact of varying numbers of tasks on average energy consumption is depicted in Fig. 4(b). With the increasing number of tasks, the energy toll is also increasing because a large number of tasks need to share the same bandwidth and require higher latency to reach the edge. Both WOLVERINE and MGBD perform better than iRAF because of exploiting caching, which helps the system's users reduce backhaul latency and energy. However, the energy consumption gap increases significantly



(a) Average latency

(b) Average energy consumption

(c) Average cost savings

(d) Task completion reliability (TCR)

Fig. 4 Impacts of a varying number of tasks

between WOLVERINE and MGBD when the number of tasks rises from 110 to 160 in the network, as MGBD needs to request the cloud for task processing owing to the unavailability of resources. For MOEA/D, a higher number of tasks means sub-tasks are executed in mobile devices more frequently, which increases overall energy consumption in the system. Besides, when sub-tasks are offloaded to multiple servers, the data code needs to be offloaded as well. Thus, energy expended for offloading data code to edge servers also occurs as an overhead, and offloading tasks frequently also incurs some communication costs with the increasing number of sub-tasks. For WOLVERINE, all the tasks requested can be cached either at different servers or the data code for computing the tasks needs to get transmitted at the servers; that is, no access to the cloud is necessary, thus, reducing energy consumption. Besides, collaboration among servers facilitates lower energy consumption.

In Fig. 4(c), we can observe the impact on average cost savings when task numbers are varied. As the number of tasks escalates, the average cost savings is reduced as the number of offloading tasks is increased, which in turn increases monetary costs for memory and computation. For the iRAF and MGBD, with increasing tasks, the cost goes up faster than WOLVERINE. The reason behind the increasing cost in the case of iRAF is the use of DNN and Monte Carlo Tree, which incur memory and computation costs. For MGBD, with an increasing number of tasks, device budget savings decrease due to the lack of collaboration among servers and offloading to the cloud in case of unavailability of server resources. In the case of MOEA/D, the cost is lower when the number of tasks is high as many of those are locally executed. However, offloading to multiple servers from a single user device can incur higher costs in terms of memory and computation depending on the availability of server resources. The proposed WOLVERINE offers a higher percentage (50%-95%) of savings compared to MGBD and iRAF due to exploiting service caching, binary offloading, and collaboration among servers.

In Fig. 4(d), we see that increasing the number of tasks reduces Task Completion Reliability (TCR) in all of the methods. This happens due to the scarcity of resources and the delay sensitivity of tasks. For the iRAF, the TCR falls steadily when the number of tasks increases from 10 to 110 but falls sharply with increasing tasks from 110 to 260. As the iRAF allows partial offloading, therefore, with the increasing number of tasks, the tendency to offload the greater portion of a task is also increased, which in turn also enhances the task drop rate. The higher task drop in iRAF is the higher training time using the DNN and Monte Carlo Tree, creating a latency overhead that

may cause many applications to exceed their deadlines. For MGBD and WOLVERINE, the TCR falls gradually with an increasing number of tasks due to caching. However, a depth-first-search tree is constructed in MGBD, which incurs some overhead, resulting in crossing the deadline for some tasks. Hence, TCR is lower in comparison to WOLVERINE. For MOEA/D, with the gradually rising number of tasks, the drop rate of sub-tasks can be increased due to the lack of resources and higher queuing delay of mobile devices. Since collaboration among edge servers and caching cater to the task completion rate better, WOLVERINE performs better than MOEA/D in system environments that contain rapidly offloaded tasks.

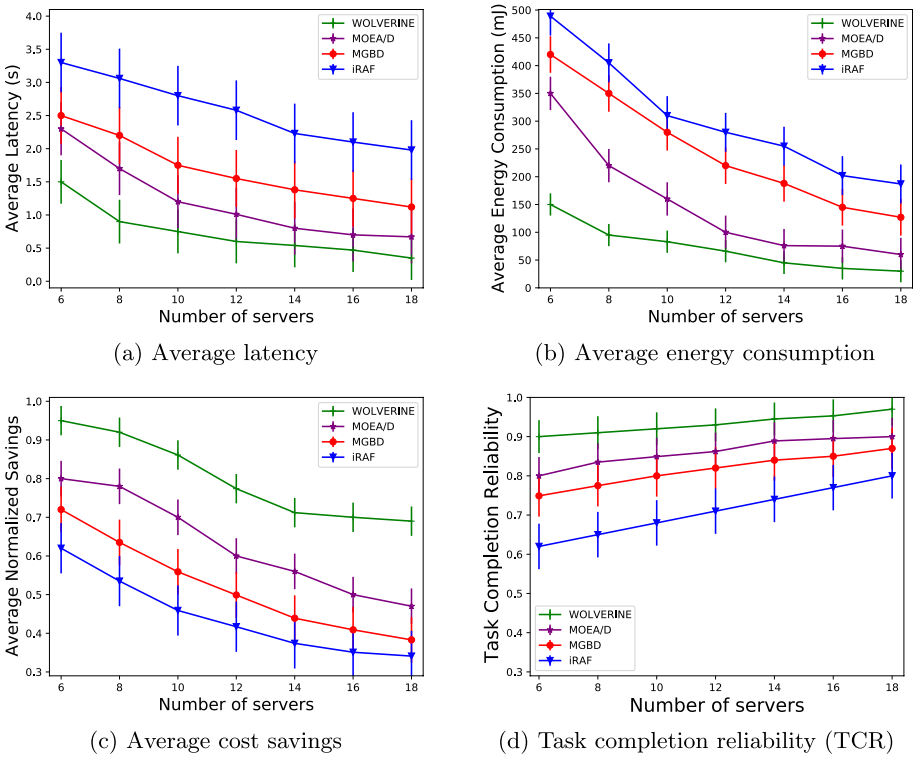
Impact of a varying number of servers

The impact of varying numbers of servers on the objective parameters is represented in the graphs of Fig. 5. For this scenario, the number of tasks is fixed at 50.

For a fixed number of tasks, as the number of resources increases, the average latency decreases for all schemes as shown in Fig. 5(a). The iRAF has higher latency in comparison to both MGBD and WOLVERINE because of the higher computational time of DNN and Monte Carlo Tree. In the case of MGBD, the construction of the search tree and exhaustive searching procedure affect the overall latency. In the case of MOEA/D, a task is disintegrated into multiple sub-tasks, which incurs higher latency overhead for server-to-device communication, and sometimes it faces difficulty to find the most suitable server for executing some sub-tasks. On the other hand, WOLVERINE exhibits better performance with the increasing number of servers as it is a joint implementation of edge server collaboration and caching.

WOLVERINE also performs better in terms of energy consumption, as depicted in Fig. 5(b). With the increasing number of servers, the energy consumption of MDs is significantly minimized in all studies. In WOLVERINE, more tasks are offloaded to the edge servers when the number of collaborative edge servers increases along with the increasing availability of cached data. That is why, up to a certain increment in the number of servers, energy consumption reduces. After that, the energy level hits a plateau or does not decrease significantly as the amount of cached data and computational resources increase with the increasing number of servers.

With the increasing number of servers for all schemes, the cost of allocating tasks is increased and the average cost savings is decreased, which is depicted in 5(c). In WOLVERINE, the cost of allocating tasks increases owing to memory cost and monetary cost for computation in various CoMEC servers. Nevertheless, the average savings remain greater than that of MGBD since the local computation of tasks also occurs here, which may incur no cost at all, and the cached resource size in MGBD is



(a) Average latency

(b) Average energy consumption

(c) Average cost savings

(d) Task completion reliability (TCR)

Fig. 5 Impacts of a varying number of servers

high along with the exhaustive search cost of DFS trees. On the other hand, iRAF involves DNN and Monte Carlo Tree in an optimization algorithm that occupies some extra memory. Therefore, the overall computation and memory cost is higher than our proposed method.

The impact on TCR (Task Completion Reliability) for varying numbers of servers is demonstrated in Fig. 5(d). For WOLVERINE and MGBD, the TCR escalates with the increasing number of servers due to exploited caching. However, the content caching in MGBD faces some resource constraint issues for highly resource-intensive applications. On the other hand, the aforementioned issues for iRAF may incur task drops due to exceeding the deadline in this scheme. For MOEA/D, TCR is relatively stable in comparison to the MGBD and iRAF as sub-tasks are offloaded more frequently with increasing tasks in the system. However, it is still not better as WOLVERINE due to the absence of server-to-server collaboration. For WOLVERINE, TCR improves due to incorporating caching as well as server collaboration.

Impact of caching

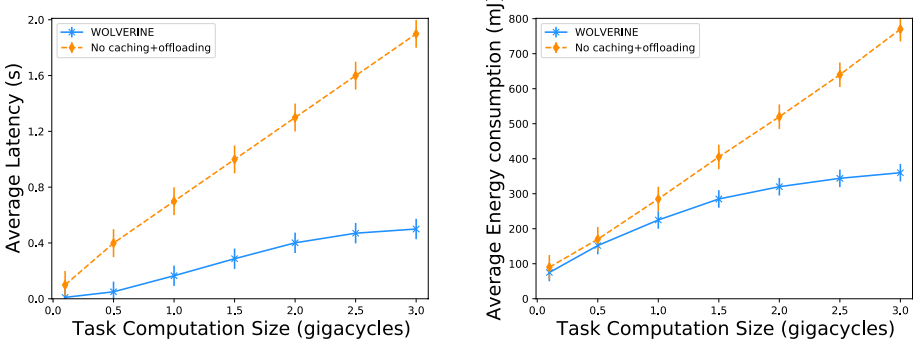
Caching the data code for computation-intensive tasks instead of the entire code itself creates a certain impact on objective parameters that are to be optimized and the impact is depicted here in Fig. 6. In this experiment, we varied the average computation per task while fixing the number of tasks and servers at 50 and 12, respectively.

Figure 6(a) indicates that if the average computation cycles per task increase, then average latency will increase exponentially without caching. Here, a considerable amount of time will be required for the computation of tasks along with the offloading of the tasks to collaborative servers. On the other hand, if caching is performed, then it is observed that the time required will be less as some of the data code is already available on the cached server; only the input data needs to be transmitted. Similarly, for average energy consumption, such changes are observed in Fig. 6(b), i.e., if the tasks are cached, less energy is wasted in communication overhead, which in turn reduces overall average energy consumption. Therefore, it is pretty clear from the experiments that service caching leverages task completion notably.

Impact of geographical proximity of users

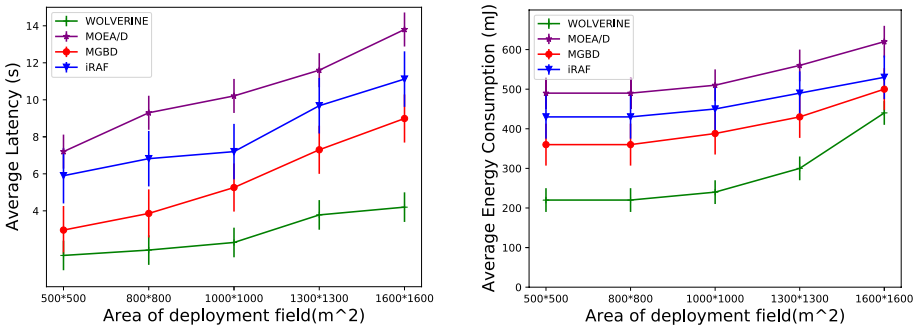
In this experiment, the geographical area is varied in an edge computing environment to measure the performance of user service latency and energy consumption. A larger area results in an augmented physical distance between edge servers and users, leading to extended transmission times and subsequently higher average latency. Additionally, expanded areas tend to experience heightened network congestion as a consequence of increased user traffic, exacerbating latency concerns. This congestion contributes to elevated communication overhead, necessitating higher transmission power and, consequently, increased energy consumption for devices. Furthermore, the scarcity of resources in an extended area often necessitates a greater execution of tasks by the devices themselves, resulting in amplified energy usage at the user end.

A gradual rise in average latency and device energy is observed for all the schemes shown in Figs. 7(a) and 7(b), respectively. However, for WOLVERINE, the increase in average latency and energy are significantly lower than the rest of the schemes. Since both collaborative edge computing and caching are exploited in this scheme, service delay and energy consumption are lowered as increased area multiplies the chances of finding appropriate edge servers and cached resources. For the rest of the schemes, the drawbacks for a higher value of energy and latency can be attributed to transmission to cloud, task dependency, higher computation, memory



(a) Average latency (b) Average energy consumption

Fig. 6 Impacts of caching on the performance



(a) Average latency (b) Average energy consumption

Fig. 7 Impacts of geographical proximity of users

overhead by algorithms themselves, and local optima convergence.

Ablation experiment

As a strategy to retain superior non-dominated solutions and to expand the exploration of a broader search space, the WOLVERINE system incorporates an adaptive grid mechanism. This technique facilitates leader selection and enhances the quality of solutions through probability-based elimination methods. To conduct an ablation experiment, we have adjusted the average number of computations per task, maintaining a fixed number of tasks and servers at 80 and 16, respectively. Subsequently, we have analyzed the effects on latency and energy consumption with and without the adaptive grid mechanism.

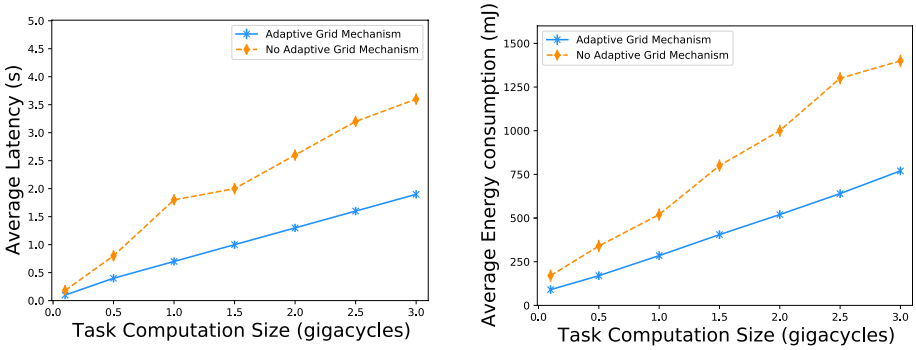
The graphs in Fig. 8 state that as the computation cycles per task increase, both average latency and average energy consumption experience an exponential rise when the adaptive grid mechanism is not utilized. Conversely, its inclusion leads to reduced latency and energy consumption. These are achieved by accelerated convergence and exploitation of the most effective solutions. It has also notably decreased the number of trial-and-error attempts.

Hypervolume and inverted generational distance

In this section, we measure the performance to evaluate the quality of Pareto-optimal solutions obtained by the developed WOLVERINE system.

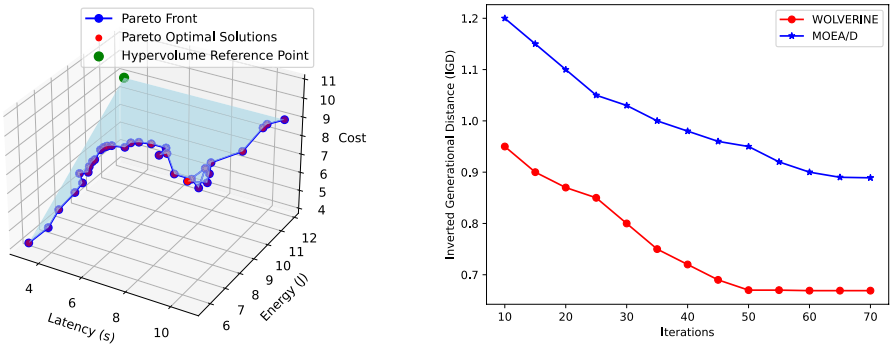
In multi-objective evolutionary algorithms (MOEAs), hypervolume is a commonly used performance metric, which measures the volume of the objective space that is dominated by the solutions in the Pareto front approximation. The hypervolume indicator assesses the effectiveness of a given set of solutions by calculating the volume of the objective space that it covers. It provides a single scalar value that represents the spread and diversity of the Pareto front approximation. Higher hypervolume values indicate better coverage and a more comprehensive representation of the Pareto front.

In Fig. 9(a), the hypervolume region of the Pareto front has been demonstrated in a 3D graph, which is computed based on the covered space by the non-dominated solutions, relative to a predefined reference point. This reference point represents an ideal state without any



(a) Average latency (b) Average energy consumption

Fig. 8 Impacts of adaptive grid mechanism



(a) Hypervolume region in 3D (b) Inverted generational distance (IGD)

Fig. 9 Quality measurements of WOLVERINE Pareto-optimal solutions

necessary trade-offs between objectives, which is marked green color in the graph for clarity. The shaded region, inclusive of the reference point, visually represents the hypervolume region, signifying the extent of the objective space covered by the set of non-dominated solutions. For hypervolume in this case, we have considered 10 servers with 60 tasks and the scalar value of hypervolume is 24.59 after 50 iterations. This is the highest value obtained which became steady after 50 iterations.

For convergence analysis of the developed WOLVERINE system, we have calculated IGD in terms of the number of iterations. Figure 9(b) illustrates how the IGD values change throughout the execution, indicating the performance and convergence of the algorithm. From this graph, we can observe that the IGD initially starts with a high value and gradually decreases throughout the first 50 iterations. Subsequently, it stabilizes at a particular value, indicating that convergence has been achieved. Higher values of IGD suggest that the solution set obtained for a certain number of iterations is not close to convergence. As the optimization algorithm explores more of the search space, lower values of IGD are obtained, signifying improved convergence and proximity to the true Pareto front. We have compared IGD values of MOEA/D with that of WOLVERINE. It is observed that IGD values for MOEA/D are higher than those of WOLVERINE for similar iteration numbers. The poorer distribution of the Pareto front in the case of MOEA/D can be attributed to its higher IGD values in comparison to WOLVERINE [45]. The value of IGD becomes stable after 50 iterations for WOLVERINE whereas for MOEA/D the value stabilizes after 65 iterations and at a higher value than that of WOLVERINE. Thus the graphical representations point towards a better convergence of WOLVERINE in comparison to MOEA/D, indicating that WOLVERINE achieves convergence faster.

Conclusion

This paper introduced an efficient task offloading framework, namely WOLVERINE, that brought about a collaboration among the edge servers to share computational resources while penetrating real-time applications in edge devices with optimal energy consumption and resource cost. The multi-objective optimization problem was proven to be NP-hard; therefore, we formulated a Binary Multi-objective Grey Wolf Optimization-based meta-heuristic solution that deduced the Pareto optimal solutions for time, energy, and cost objectives i.e., the tri-objective optimization problem in polynomial time. The performance analysis results carried out in Python and demonstrated significant performance improvement as high as 33.33%, 35%, and 40% in terms of execution latency, energy, and resource cost, respectively compared to the state-of-the-art.

An improved version of GWO can be exploited on the developed system through dynamic weight association to multiple objectives and modification to the convergence factor. New scopes can be added by considering data loss, security of executed tasks, and so on. Deployment of a deep-learning model to accurately predict the task arrival rate, allocate the tasks, and adjust the cache resources following that prediction can be interesting future works. Furthermore, we can enhance our current framework by hybridizing different evolutionary algorithms to address the strengths of these algorithms in a dynamic environment. Consideration of robustness and fault tolerances in case of points of failure also adds a new edge to our current work.

Authors' contributions

Idea Generation, Writing and Analysis - Nawmi Nujhat, Fahmida Haque Shanta, Palash Roy, Sujan Sarker Supervision and Evaluation - Md. Mamun-Or-Rashid, Md. Abdur Razzaque Formulation, Editing, and Analysis - Mohammad Mehedi Hassan and Giancarlo Fortino.

Funding
The authors are grateful to the UGC Research Project, University of Dhaka, Bangladesh for supporting grants. This work was also supported by King Saud University, Riyadh, Saudi Arabia, through the Researchers Supporting Project under Grant- RSP2024R18.

Availability of data and materials
The data will be provided upon request.

Declarations

Ethics approval and consent to participate
Not applicable.

Competing interests
The authors declare no competing interests.

Received: 25 July 2023 Accepted: 7 December 2023
Published online: 23 January 2024

References

1. Liang B, Gregory MA, Li S (2022) Multi-access edge computing fundamentals, services, enablers and challenges: a complete survey. J Netw Comput Appl 199(103):308
2. Sahni Y, Cao J, Yang L (2018) Data-aware task allocation for achieving low latency in collaborative edge computing. IEEE Internet Things J 6(2):3512–3524
3. Nandi PK, Reaj MRI, Sarker S, Razzaque MA, Rashid MM, Roy P, (2024) Task offloading to edge cloud balancing utility and cost for energy harvesting internet of things. J Netw Comput Appl 221:103766
4. Zhang J, Hu X, Ning Z, Ngai ECH, Zhou L, Wei J, Cheng J, Hu B, Leung VC (2018) Joint resource allocation for latency-sensitive services over mobile edge computing networks with caching. IEEE Internet Things J 6(3):4283–4294
5. Shafique K, Khawaja BA, Sabir F, Qazi S, Mustaqim M (2020) Internet of things (IoT) for next-generation smart systems: A review of current challenges, future trends and prospects for emerging 5G-IoT scenarios. IEEE Access 8:23022–23040
6. Ullah I, Lim HK, Seok YJ, Han YH (2023) Optimizing task offloading and resource allocation in edge-cloud networks: a DRL approach. J Cloud Comput 12(1):112
7. Ren J, Yu G, He Y, Li GY (2019) Collaborative cloud and edge computing for latency minimization. IEEE Trans Veh Technol 68(5):5031–5044
8. Puthal D, Mohanty SP, Wilson S, Choppali U (2021) Collaborative edge computing for smart villages [energy and security]. IEEE Consum Electron Mag 10(3):68–71
9. Chien WC, Weng HY, Lai CF (2020) Q-learning based collaborative cache allocation in mobile edge computing. Futur Gener Comput Syst 102:603–610
10. Xu J, Chen L, Zhou P (2018) Joint service caching and task offloading for mobile edge computing in dense networks. In: IEEE INFOCOM 2018-IEEE Conference on Computer Communications, IEEE, pp 207–215
11. Mach P, Becvar Z (2017) Mobile edge computing: a survey on architecture and computation offloading. IEEE Commun Surv Tutor 19(3):1628–1656
12. Li C, Tang J, Tang H, Luo Y (2019) Collaborative cache allocation and task scheduling for data-intensive applications in edge computing environment. Futur Gener Comput Syst 95:249–264
13. Alfakih T, Hassan MM, Gumaei A, Savaglio C, Fortino G (2020) Task offloading and resource allocation for mobile edge computing by deep reinforcement learning based on SARSA. IEEE Access 8:54074–54084
14. Alam MGR, Hassan MM, Uddin MZ, Almogren A, Fortino G (2019) Autonomic computation offloading in mobile edge for IoT applications. Futur Gener Comput Syst 90:149–157
15. Chen Z, Chen Z, Jia Y (2019) Integrated task caching, computation offloading and resource allocation for mobile edge computing. In: IEEE Global Commun. Conf. (GLOBECOM). IEEE, Waikoloa, pp 1–6
16. Bi S, Huang L, Zhang YJA (2020) Joint optimization of service caching placement and computation offloading in mobile edge computing systems. IEEE Trans Wirel Commun 19(7):4947–4963
17. Chen J, Chen S, Wang Q, Cao B, Feng G, Hu J (2019) iRAF: A deep reinforcement learning approach for collaborative mobile edge computing IoT networks. IEEE Internet Things J 6(4):7011–7024
18. Luo Q, Li C, Luan T, Shi W (2022) Minimizing the delay and cost of computation offloading for vehicular edge computing. IEEE Trans Serv Comput 1. 15(5):2897–2909.
19. Wang P, Li K, Xiao B, Li K (2022) Multiobjective optimization for joint task offloading, power assignment, and resource allocation in mobile edge computing. IEEE Internet Things J 9(14):11737–11748
20. Gu B, Chen Y, Liao H, Zhou Z, Zhang D (2018) A distributed and context-aware task assignment mechanism for collaborative mobile edge computing. Sensors 18(8):2423
21. Mahenge MP, Li C, Sanga CA Collaborative mobile edge and cloud computing: Tasks unloading for improving users' quality of experience in resource-intensive mobile applications. In: 2019 IEEE 4th Int. Conf. Comput. and Commun. Systems (ICCCS). IEEE, Singapore, pp 322–326
22. Mohammed A, Nahom H, Tewodros A, Habtamu Y, Hayelom G (2020) Deep reinforcement learning for computation offloading and resource allocation in blockchain-based multi-UAV-enabled mobile edge computing. In: 2020 17th International Computer Conference on Wavelet Active Media Technology and Information Processing (ICCWAMTIP). IEEE, Chengdu, pp 295–299
23. Nur FN, Islam S, Moon NN, Karim A, Azam S, Shanmugam B (2019) Priority-based offloading and caching in mobile edge cloud. J Commun Softw Syst 15(2):193–201
24. Hao Y, Song Z, Zheng Z, Zhang Q, Miao Z (2023) Joint communication, computing, and caching resource allocation in LEO satellite MEC networks. IEEE Access 11:6708–6716
25. Gul-E-Laraib, Zaman SKu, Maqsood T, Rehman F, Mustafa S, Khan MA, Gohar N, Algarni AD, Elmannai H (2023) Content caching in mobile edge computing based on user location and preferences using cosine similarity and collaborative filtering. Electronics 12(2)
26. Xiao Z, Shu J, Jiang H, Lui JC, Min G, Liu J, Dustdar S (2022) Multi-objective parallel task offloading and content caching in D2D-aided MEC networks. IEEE Trans Mob Comput 22(11):6599–6615
27. Hao Y, Chen M, Hu L, Hossain MS, Ghoneim A (2018) Energy efficient task caching and offloading for mobile edge computing. IEEE Access 6:11365–11373
28. Zhang N, Guo S, Dong Y, Liu D (2020) Joint task offloading and data caching in mobile edge computing networks. Comput Netw 182:107446
29. Liu L, Chang Z, Guo X, Ristaniemi T Multi-objective optimization for computation offloading in mobile-edge computing. In: 2017 IEEE Symposium Comput. and Commun. (ISCC). IEEE, Heraklion, pp 832–837
30. Seid AM, Lu J, Abishu HN, Ayall TA (2022) Blockchain-enabled task offloading with energy harvesting in multi-UAV-assisted IoT networks: A multi-agent DRL approach. IEEE J Sel Areas Commun 40(12):3517–3532
31. Deb K (2001) Multiobjective Optimization Using Evolutionary Algorithms. Wiley, New York
32. Afrin M, Jin J, Rahman A, Tian YC, Kulkarni A (2019) Multi-objective resource allocation for edge cloud based robotic workflow in smart factory. Future Gener Comput Syst 97:119–130
33. Song C, Zhou H (2020) Computation offloading optimization in mobile edge computing based on multi-objective cuckoo search algorithm. In: Proceedings of the 2020 the 4th International Conference on Innovation in Artificial Intelligence, pp 189–193
34. Abbas A (2021) Meta-heuristic-based offloading task optimization in mobile edge computing. Int J Distrib Sens Netw
35. Jiang K, Ni H, Han R, Wang X (2019) An improved multi-objective grey wolf optimizer for dependent task scheduling in edge computing. Int J Innov Comput Inf Control 15(6):2289–2304
36. Song F, Xing H, Luo S, Zhan D, Dai P, Qu R (2020) A multiobjective computation offloading algorithm for mobile-edge computing. IEEE Internet Things J 7(9):8780–8799
37. Gong Y, Bian K, Hao F, Sun Y, Wu Y (2023) Dependent tasks offloading in mobile edge computing: a multi-objective evolutionary optimization strategy. Futur Gener Comput Syst 148:314–325
38. Sardar Khaliq uz Z, Maqsood T, Rehman F, Mustafa S, Khan MA, Gohar N, Algarni AD, Elmannai H, (2023) Content caching in mobile edge

computing based on user location and preferences using cosine similarity and collaborative filtering. Electronics 12(2):284
39. Man TH (2005) An algorithm for multi-objective assignment problem. PhD thesis, The Chinese University of Hong Kong
40. Mirjalili S, Mirjalili SM, Lewis A (2014) Grey wolf optimizer. Adv Eng Softw 69:46–61
41. Shahjalal M, Farhana N, Roy P, Razzaque MA, Kaur K, Hassan MM (2022) A binary gray wolf optimization algorithm for deployment of virtual network functions in 5G hybrid cloud. Comput Commun 193:63–74
42. Hussein M (2021) Simulation-optimization for the planning of off-site construction projects: a comparative study of recent swarm intelligence metaheuristics. Sustainability 13(24):13551
43. Al-Imron CN (2022) An energy-efficient no idle permutations flow shop scheduling problem using grey wolf optimizer algorithm. Jurnal Ilmiah Teknik Industri 21(1)
44. Wei L (2022) Multi-objective gray wolf optimization algorithm for multi-agent pathfinding problem. In: 2022 IEEE 5th International Conference on Electronics Technology (ICET). IEEE, Chengdu
45. Mirjalili S, Saremi S, Mirjalili SM, Coelho LdS (2016) Multi-objective grey wolf optimizer: a novel algorithm for multi-criterion optimization. Expert Syst Appl 47:106–119
46. Roy P, Sarker S, Razzaque MA, Hassan MM, AlQahtani SA, Aloi G, Fortino G (2020) AI-enabled mobile multimedia service instance placement scheme in mobile edge computing. Comput Netw 182(107):573
47. Zhao F, He X, Zhang Y, Ma W, Zhang C A novel pareto archive evolution algorithm with adaptive grid strategy for multi-objective optimization problem. In: 2019 IEEE 23rd Int. Conf. Comput. Support. Coop. Work Des. (CSCWD), pp 301–306
48. Liu Y, Wei J, Li X, Li M (2019) Generational distance indicator-based evolutionary algorithm with an improved niching method for many-objective optimization problems. IEEE Access 7:63881–63891
49. Van Rossum G, Drake FL (2009) Python 3 Reference Manual. CreateSpace, Scotts Valley
50. Gilbert EN (1960) Capacity of a burst-noise channel. Bell Syst Tech J 39(5):1253–1265

Publisher's Note