IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 24 May 2024, accepted 14 June 2024, date of publication 17 June 2024, date of current version 24 June 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3415756

RESEARCH ARTICLE

Efficient Task Offloading Strategy for Energy-Constrained Edge Computing Environments: A Hybrid Optimization Approach

DEAFALLAH ALSADIE, (Member, IEEE)
Department of Computer Science and Artificial Intelligence, College of Computing, Umm Al-Qura University, Makkah 21961, Saudi Arabia
e-mail: dbsadie@uqu.edu.sa

ABSTRACT Edge Computing (EC) has emerged as a pivotal paradigm, offering solutions to address the challenges posed by latency-sensitive applications and to enhance overall network performance. In EC environments, efficient task offloading is crucial for minimizing latency and energy consumption while maximizing resource utilization. In this paper, we propose a hybrid task offloading approach (HybridTO) integrating Grey Wolf Optimizer and Particle Swarm Optimization. Our approach aims to optimize energy consumption and fulfil latency constraints in EC environments by taking into account various factors such as capacity constraints, proximity constraints, and latency requirements. Leveraging the collaborative capabilities inherent in EC servers, HybridTO offers a comprehensive solution to the task offloading problem. Through extensive simulations, we evaluate the performance of HybridTO against baseline approaches, demonstrating its superiority regarding energy usage, offloading utility and response delay, especially under conditions of limited resources. These results underscore the effectiveness of HybridTO as a promising solution for energy-efficient task offloading in EC environments, offering valuable insights for further research and development in this field.

INDEX TERMS Edge computing, task offloading, energy efficiency, optimization, hybrid algorithms.

I. INTRODUCTION

Edge Computing (EC) has emerged as a transformative paradigm aimed at addressing the escalating challenges posed by latency-sensitive applications and bolstering network performance by strategically situating computation and storage resources closer to the edge of the network [1]. As the technological landscape witnesses an exponential proliferation of Internet of Things (IoT) devices and experiences an escalating demand for real-time applications, the significance of EC in aligning with modern network requisites has become increasingly salient [7].

In traditional computing paradigms, tasks sensitive to delays have traditionally found their execution grounds on cloud platforms [2]. This convention has often resulted in significant latency issues arising from the transit of data between smart end devices (SEDs) and remote cloud servers (CSs). However, with the advent of EC, a promising paradigm that strategically situates computational resources closer to the network edge, the constraints of cloud-centric architectures are being aptly addressed [3]. This shift in computing architecture assumes paramount importance given the rapid proliferation of SEDs, equipped not only with the capacity to generate requests but also imbued with increasingly sophisticated computational capabilities, courtesy of advances in hardware technology [4]. Centralized cloud platforms may boast abundant resources, but their geographical remoteness from end-users often translates into prolonged transmission times, rendering them unsuitable for low-latency tasks [5]. Conversely, edge computing alleviates transmission latency by deploying resources in closer proximity to users, typically at the network edge. However, despite this inherent advantage, edge computing may lack the expansive computational prowess synonymous with centralized cloud environments.

The associate editor coordinating the review of this manuscript and approving it for publication was Qiang Li.

To optimally harness the benefits encapsulated within EC and foster efficient collaboration between edge and cloud resources, researchers have charted innovative paths through the proposition of cooperative architectures such as the edge-cloud architecture [6], [7]. This architectural framework orchestrates the synergistic utilization of each layer's unique strengths, facilitating the streamlined processing of a diverse array of tasks across the network infrastructure. However, navigating the landscape of executing intelligent tasks within the purview of the edge-cloud architecture presents several intricacies. Tasks laden with high resource demands and stringent latency requisites necessitate sophisticated offloading strategies to manoeuvre tasks between the edge and cloud strata effectively. Each SED, whether endowed with computational capabilities or not [4], finds itself at the crossroads of deciding whether to process tasks locally or offload them for remote processing [8].

Effectively managing the inherent complexity entailed in the execution of intelligent tasks and the subsequent offloading thereof mandates the adept handling of an array of constraints [2]. One such constraint stems from the presence of dependency constraints, which arise from the sequential nature of characterizing tasks within an intelligent job. Tasks such as face recognition exemplify this sequentiality, involving multiple consecutive steps like detecting the face, its pre-processing, extracting features, and face marking. The successful execution of specific tasks hinges upon the output generated by others, thereby complicating the decision-making process concerning offloading, as tasks may necessitate distribution across different servers to ensure seamless execution.

Moreover, service constraints add further complexity to task execution, as certain tasks depend on specific services like databases, libraries, or trained models [9]. However, the finite computational resources available at the edge and SEDs curtail the deployment of a limited subset of essential services, thereby exacerbating the complexities inherent in task-offloading decisions. While extant research has predominantly honed in on optimizing service deployment, scant attention has been devoted to the joint optimization of deployment and task offloading.

Efficient task offloading in EC environments remains a difficult challenge, particularly in scenarios beset by resource constraints and exacting latency prerequisites [10]. Task offloading necessitates intricate decision-making processes that weigh factors such as resource availability, network conditions, energy consumption, and application requisites. The literature abounds with a plethora of approaches aimed at tackling the task offloading conundrum in EC, spanning the gamut from heuristic-based methodologies to metaheuristic optimization techniques [11], [12], [13]. However, existing approaches often overlook the unique characteristics inherent to EC environments, such as resource constraints and the latent potential within multiple ESs.

In a bid to bridge this lacuna, this paper sets out to introduce an iterative methodology strategically designed to collectively optimize energy consumption, latency, and task offloading in EC environments. Acknowledging the NP-hard nature pervading these challenges [9], our approach endeavours to elevate the efficiency and efficacy of intelligent task execution within EC frameworks. Embracing this approach, we endeavour to engender the development of more streamlined and resource-efficient computing paradigms. In furtherance of this objective, we propose a hybrid task offloading strategy (HybridTO), hewn from the amalgamation of particle swarm optimization (PSO) [14] and grey wolf optimizer (GWO) [15], tailored for EC environments. The crux of our proposed strategy lies in optimizing energy consumption while seamlessly navigating latency constraints and resource availability. Unlike extant approaches, HybridTO meticulously takes into account the capacity constraints intrinsic to edge servers (ESs) and deftly harnesses the collaborative capabilities latent within multiple ES resources. By orchestrating intelligent swarm optimization techniques, HybridTO adeptly allocates tasks to EC servers, factoring in their computational prowess and proximity to end-users.

The main contributions of this paper can be distilled as follows:

1) Introduction of a hybrid model meticulously crafted to design an energy-efficient task offloading scheme tailored for EC environments. This model deftly navigates capacity constraints, proximity constraints, and latency requirements, all while harnessing the collaborative potential inherent within EC servers or between EC servers and the central cloud, thereby ensuring minimum latency and heightened computing power.
2) Proposal of a hybrid approach grounded in PSO and GWO to seamlessly resolve the optimization problem at hand. This approach optimizes resource allocation, spanning sub-carriers, power, and bandwidth for offloading, with the overarching objective of minimizing energy consumption while efficaciously meeting delay requirements.
3) Conducting extensive simulations to evaluate the performance metrics of the proposed strategy meticulously. By testing across various parameters, our findings clearly demonstrate the superiority of HybridTO over baseline approaches, particularly in terms of energy utilization, response delay, and offloading efficiency, especially in resource-constrained environments.

The subsequent sections of this paper are organized as follows: Section II presents a comprehensive review of pertinent literature, pinpointing areas necessitating further exploration. Subsequently, Section III delineates the problem formulation and the foundational models employed in our investigation. In Section IV, we delve into a comprehensive exposition of the tailored HybridTO algorithm crafted for optimizing task offloading efficiency within edge computing environments. Following that, Section V elucidates the

experimental configuration, the resultant discoveries, and their detailed scrutiny. Finally, Section VI encapsulates the overarching insights gleaned from this research endeavour.

II. RELATED WORK

In EC environments, where the proliferation of SEDs is increasing, numerous research efforts have addressed the task offloading challenge to enhance service quality and optimize resource utilization. For instance, Sang et al. [16] introduced a heuristic offloading algorithm aimed at fostering cooperation among EC resources by initially routing tasks to the cloud. While this approach enhances user satisfaction, it may inadvertently compromise overall system performance. Similarly, Wang et al. [11] proposed the Fastest Response First (FRFOA) and Load Balance Offloading Algorithm (LBOA), with FRFOA prioritizing task offloading to ESs with minimal response time, and LBOA distributing tasks to ESs capable of accommodating the most significant number of concurrent tasks. However, these heuristic-based methods often yield suboptimal solutions due to their reliance on local search strategies.

To address this limitation, several studies have explored meta-heuristic approaches endowed with global search capabilities. Wang et al. [17] and Gao et al. [18] employed PSO, with the latter incorporating Levy Flight movement patterns to enhance exploration. Additionally, Wang et al. [19] utilized Genetic Algorithms (GA) to optimize similar objectives. Chakraborty and Mazumdar [20] leveraged GA to curtail energy consumption during task executions while adhering to latency constraints. Furthermore, Yadav et al. [21] employed a multi-objective Grey Wolf Optimization (MOGWO) technique to optimize energy consumption and computational time in fog computing scenarios. Xu et al. [22] proposed the multiple approaches using a tunicate swarm optimization algorithm (M-TSA), an improved iteration of TSA that integrates multiple strategies. This enhanced algorithm aims to optimize the weighted sum of time delay and energy consumption in edge-to-cloud systems. Significantly, these investigations predominantly concentrated on employing single meta-heuristic approaches, overlooking the potential synergistic benefits achievable through the combination of different meta-heuristics to enhance overall performance.

From a system architecture perspective, prior research on task offloading has predominantly concentrated on single-layer architectures [2]. Tasks were primarily offloaded to CSs, with a limited exploration into other offloading scenarios within EC architectures [23], [24], [25]. These investigations commonly operated under the assumption that SEDs lacked computational capabilities, thus necessitating task processing solely on external servers. For instance, Chen et al. [24] introduced a novel task offloading framework tailored for EC within a Software-Defined Ultra-Dense Network (SD-UDN), where the SD-UDN controller gathered system data to inform offloading decisions. Similarly, Qin et al. [25] delved into offloading determinations within a large-scale CC environment, accounting for the constrained computational capacity of CSs and exploring task transfers between servers to enhance response quality. Moreover, other investigations tackled multi-user multi-CAP EC networks, formulating task offloading as a Markov decision process and devising offloading strategies leveraging deep Q networks [23].

In recent years, there has been a notable emphasis on exploring task offloading strategies within a two-layer architectural framework, mainly focusing on the edge-cloud architecture [26], [27], [28] and device-edge system architecture [29], [30]. For instance, Sun et al. [27] proposed an innovative joint offloading scheme that optimizes resource usage by predicting resource occupancy, thus determining the most suitable destination for task offloading, whether to a remote cloud server or an edge node. This approach becomes particularly relevant in scenarios where physical devices are interconnected via various communication channels like optical fibre or wireless links. Similarly, Shah-Mansouri et al. [26] introduced a computational offloading game aimed at enhancing the Quality of Experience (QoE) within an edge-cloud system. This game-like approach focuses on strategically offloading tasks to either local edge nodes or remote cloud servers via access points, depending on factors such as task requirements, network conditions, and available resources.

Another noteworthy strategy involves the application of convex theory, as demonstrated in the work by Ren et al. [28]. By leveraging convex optimization principles, decisions are made regarding whether a given task should be exclusively processed at the edge node or if collaboration with the cloud is necessary to achieve optimal performance and resource utilization. It's important to highlight that task offloading decisions are not exclusively limited to local edge nodes or remote cloud servers. Edge servers possess the capability to communicate with each other and exchange data, enabling tasks to be offloaded to non-local servers when necessary to optimize resource usage and enhance system performance. This collaborative nature of EC architectures allows for dynamic and flexible task-offloading strategies tailored to the specific requirements and constraints of diverse applications and network environments. Interestingly, some studies have begun to consider the computing capacity of terminals. Kao et al. [30] explored optimal task assignments to local devices or remote servers, while other research [8], [29] focused on offloading tasks to local device or edge computing environments, assuming that devices possessed the capability to process jobs. However, many of these studies have assumed uniform strong capacity across all SEDs and allocated resources solely based on task requests. There has been relatively less exploration into task offloading within a three-tier architecture, such as device-edge-cloud systems [32]. In these architectures, algorithms are developed to combine convex programs with progressive rounding techniques to offload dependent tasks while incorporating service caching mechanisms effectively. This approach allows for more efficient resource utilization and task execution by

TABLE 1. Comparative Analysis of Task Offloading Approaches in Edge Computing Environments.

Study	Approach	Optimization Objectives	Focus Area	Performance Metrics	Strengths	Limitations
Sang et al. [16]	Heuristic	Enhanced cooperativeness, user satisfaction	EC resource allocation	User satisfaction, task response time	Addresses cooperativeness	Suboptimal performance due to local search
Wang et al. [11]	Heuristic (FRFOA, LBOA)	Minimize response time, Load balancing	Task offloading in MEC	Response time, task distribution	Improved load balancing	Limited by local search strategies
Wang et al. [17]	Meta-heuristic (PSO)	Energy efficiency, latency reduction	Task offloading optimization	Energy consumption, latency	Global search capability	Lack of synergy with other meta-heuristics
Gao et al. [18]	Meta-heuristic (PSO with Levy Flight)	Energy optimization, enhanced exploration	Task offloading in MEC	Energy consumption, exploration efficiency	Enhanced exploration capability	Limited application in complex scenarios
Chakraborty et al. [20]	Meta-heuristic (GA)	Energy consumption reduction, latency constraints	Task offloading optimization	Energy consumption, latency satisfaction	Effective energy optimization	Limited to single optimization objective
Yadav et al. [21]	Meta-heuristic (MOGWO)	Energy consumption optimization, computational time	Fog computing	Energy consumption, computational time	Multi-objective optimization	Complexity in parameter tuning
Xu et al. [22]	Meta-heuristic (M-TSA)	Weighted sum optimization of delay and energy	Edge-to-cloud systems	Delay, energy consumption	Integration of various strategies	Complexity in convergence
Chen et al. [24]	Heuristic	Task offloading in SD-UDN	EC architecture	Offloading decisions, system information	Utilizes network information	Limited scalability
Qin et al. [25]	Heuristic	Task offloading in mobile CC	EC architecture	Task transfer, response quality	Improves response quality	Limited exploration of offloading scenarios
Li et al. [23]	Heuristic	Task offloading in multiuser multi-CAP MEC	EC architecture	Offloading strategies, Markov decision process	Utilizes DQN for offloading	Limited scalability
Sun et al. [27]	Heuristic	Joint offloading scheme based on resource occupancy	Edge-cloud architecture	Offloading decisions, QoE	Utilizes resource prediction	Limited by prediction accuracy
Ren et al. [28]	Heuristic	Convex theory-based task offloading	Edge-cloud architecture	Task processing decisions	Utilizes convex theory	Limited applicability to complex scenarios
Shah-Mansouri et al. [26]	Heuristic	Computation offloading game for QoE maximization	Edge-cloud architecture	Offloading decisions, QoE	Maximizes QoE	Limited to local edge nodes or remote cloud servers
Kao et al. [30]	Heuristic	Optimal task assignments to local devices or remote	Device-edge system architecture	Task allocation decisions	Considers computing capacity	Assumes uniform strong capacity
Zhan et al. [29]	Heuristic	Task offloading with deep reinforcement learning	Device-edge system architecture	Offloading decisions, energy harvesting	Utilizes reinforcement learning	Limited scalability
Sundar et al. [31]	Heuristic	Task offloading with DAG modeling	Device-edge system architecture	Offloading decisions, task dependencies	Models task dependencies	Limited scalability

considering the interdependencies between tasks and leveraging caching to minimize data transmission and processing overhead.

Cheng et al. [13] introduced a novel approach that combines offloading and resource provisioning to minimize overall energy consumption in time-sensitive applications across multi-mobile EC networks. Wang et al. [33] explored task offloading in edge computing environments, explicitly focusing on optimizing both delay and energy consumption simultaneously. They tackled the problem using mixed integer nonlinear programming and applied relaxation techniques to solve it effectively. Similarly, Cao et al. [12] put forward a joint optimization strategy for computation and communication resource provisioning in EC settings aimed at reducing energy usage while ensuring latency constraints are met. Their research highlighted the benefits of a cooperative approach in improving performance and energy efficiency compared to non-collaborative designs. Notably, numerous

prior studies have also proposed energy-efficient methods in EC contexts.

From the standpoint of task attribution, existing research has predominantly concentrated on managing wholly individual tasks that cannot be divided further [2]. In such scenarios, the primary objective has been to determine the optimal execution location for each task based on node information, primarily to meet task delay requirements. However, this approach often leads to redundant deployment of identical services across different nodes, resulting in inefficient resource utilization.

As Deep Neural Network (DNN) technology has progressed, tasks have become divisible into segments, offering opportunities for more detailed task allocation strategies, whether vertically or horizontally. Consequently, many recent studies have explored task offloading mechanisms in scenarios involving task splitting, where tasks are divided into smaller components to be executed across multiple nodes or layers. Task splitting introduces additional complexity due to the need to satisfy precedence constraints, which dictate the order in which tasks must be executed and data transfer requirements between task segments.

Some research efforts have focused explicitly on splittable and parallel computing task offloading mechanisms, particularly in the context of horizontally splitting DNN tasks. These studies aim to determine the optimal execution ratio and location for each segment of a task, considering factors such as computational capabilities and network conditions. As an example, Zhang et al. [29] utilized a deep reinforcement learning method to dynamically ascertain the offloading ratio for tasks among devices featuring dynamic energy-related capabilities. Similarly, other researchers [34] have proposed evolutionary algorithms for task separation in cloud-edge collaboration scenarios, aiming to maximize resource utilization while minimizing execution latency.

On the other hand, there is a growing body of literature addressing splittable and dependent computing task offloading mechanisms, particularly in the context of vertically splitting DNN tasks. In such scenarios, task offloading decisions must consider dependencies between task segments, ensuring that tasks are executed in the correct sequence to maintain integrity and accuracy. Some studies, such as those conducted by Sundar et al. [31] have modeled task dependencies using directed acyclic graphs (DAGs), enabling more sophisticated offloading decisions based on task interdependencies.

However, despite these advancements, existing research often overlooks the intricacies of data transmission between adjacent task segments and the overall architecture within which task offloading occurs. In particular, numerous investigations concentrate solely on choosing the edge node for executing each task segment within the edge layer, often overlooking the broader architecture encompassing terminals, edge, and cloud layers. Therefore, there is a need for comprehensive investigations that consider the entire task offloading process within complex multi-layer architectures to ensure optimal resource utilization and task execution efficiency.

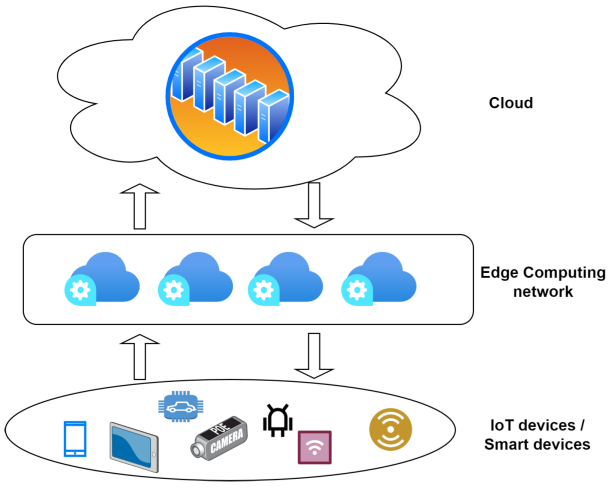

FIGURE 1. Edge-Cloud Computing Architecture.

This study diverges from existing research in several key aspects. Firstly, while many prior studies overlook EC-specific characteristics like resource limitations, our proposed strategy takes into account capacity constraints, including both computational and transmission limitations, as well as latency requirements. By doing so, it effectively allocates EC resources to minimize energy consumption while ensuring timely task completion. Secondly, while existing works typically focus on optimizing task offloading decisions within single-user or multi-user EC environments or between EC servers and the central cloud, our approach emphasizes a cooperative strategy among multiple EC resources or between EC servers and the central cloud. This cooperative approach enhances overall capacity utilization and improves system performance. Thirdly, our proposed strategy leverages intelligent swarm optimization techniques, requiring minimal information from the EC system. This makes our approach suitable for decentralized implementations and ensures scalability and adaptability in various EC environments.

Table 1 provides an overview of the discussed studies, outlining their approach, optimization objectives, focus area, performance metrics, strengths, and limitations.

III. UNDERLYING MODELS AND PROBLEM FORMULATION

In this section, we provide an overview of the network architecture and its constituent elements as depicted in Figure 1.

In the depicted Edge-Cloud environment setup, illustrated in Figure 1, a dynamic ecosystem unfolds, encompassing a multitude of SEDs, ESs, and a cloud infrastructure equipped with a plethora of CSs. Within this ecosystem, users engage in a diverse range of tasks using their devices.

While SEDs possess local computational capabilities to handle specific tasks, they often encounter limitations, especially those lacking sufficient computing resources, such as environmental sensors.

As a result, specific tasks are delegated to ESs for processing, necessitating the transfer of input data from the SED to the ES. This task offloading process is contingent upon the existence of a network connection between the device and the ES. Typically, an ES provides a localized network, such as 5G or WiFi, limiting its service coverage to devices within its range.

In scenarios where users generate numerous task requests that exceed the capacities of both devices and ESs, tasks with flexible timing requirements can be forwarded to the cloud. The cloud infrastructure operates over a vast area network (WAN) and extends coverage to all devices within the network. Upon receiving offloaded tasks, the cloud assigns resources in the form of CS to manage the processing requirements.

Therefore, task offloading entails the crucial task of determining the appropriate computing node (SED, ES, or CS) for processing each task based on factors such as task requirements, device capabilities, network conditions, and system constraints.

A. SYSTEM MODEL

E2C framework comprises of computing nodes: SED, ES, and CS, denoted as nodes *SED* (for devices, $1 \leq i \leq$ SEDs), *ES* (for ES, $SED+1 \leq j \leq SED+E$), and *CS* (for CS, $SED+ES+1 \leq i \leq SED+ES+CS$). Each node n_i possesses a processing capacity p_i and provides a network bandwidth rate bw_i. Binary constants $a_{i,j}$, where $1 \leq i \leq$ SED, $SED+1 \leq j \leq SED+E$, are utilized to signify the coverage of ES. Specifically, $a_{i,j} = 1$ if device n_i is within the coverage of ES n_j, and $a_{i,j} = 0$ otherwise. As a result, for a given task t_i, the bandwidth rate $bw_{i,j}$ when offloaded to ES n_j is determined by the expression $bw_{i,j} = a_{i,j} \cdot bw_j$, leveraging the signal power and Gaussian noise characteristics of the network.

The system receives a total of TS tasks (t_k, $1 \leq k \leq TS$) from the SEDs. Binary constants $m_{i,k}$, where $1 \leq i \leq$ SED and $1 \leq k \leq TS$, indicate the task-device relationships. Specifically, $m_{i,k} = 1$ denotes that t_k is initiated by n_i, while $m_{i,k} = 0$ indicates otherwise. Each task t_k is characterized by its computing size c_k, input data volume p_k, and a deadline d_k, representing the maximum processing finish time. This study prioritizes strict deadline requirements, postponing consideration of soft deadline constraints for future exploration. Essentially, if a task's deadline can be met, it will undergo processing within the E2C framework; otherwise, it is rejected, as no benefit arises from handling overdue tasks. Without loss of generality, the paper assumes $d_k \leq d_{k+1}$ for $1 \leq k \leq TS-1$.

To model the task offloading problem in Edge-Cloud environment, binary variables $x_{i,k}$ are introduced to denote offloading decisions, as defined in Equation 1:

$$x_{i,k} = \begin{cases} 1, & \text{if } t_k \text{ is processed by } n_i \\ 0, & \text{otherwise} \end{cases} \tag{1}$$

where $1 \leq i \leq SED+ES+CS$, $1 \leq k \leq TS$.

B. MODEL FOR TASK PROCESSING

Tasks undergo three distinct processing scenarios: (i) local processing utilizing the resources of their respective SED and (ii) offloading to an ES that covers the device. Subsequently, this paper delineates the task-processing procedures corresponding to these three scenarios.

1) LOCAL TASK PROCESSING

In the initial scenario, transmission delays are nonexistent as data remain stored locally upon collection by the SED, aligning with environmental conditions and user behaviour. Consequently, the task's processing time equates to its computing time. However, task execution commences solely upon the availability of computing resources, coinciding with the completion of preceding tasks by the device. To optimize the acceptance ratio, defined as the ratio between accepted and total task numbers, the Earliest Deadline First (EDF) strategy has been demonstrated to offer an optimal solution. Hence, to formulate the task offloading optimization problem, this study derives the completion time for each task by adhering to an EDF order for each device. Hence, the task completion duration for tasks processed locally can be calculated using Equation 2, where $c_k \cdot p_i$ signifies the processing time of t_k handled by n_i, and $\sum_{k'=1}^{k} x_{i,k'} \cdot \frac{c_{k'}}{p_i}$ represents the cumulative processing time of tasks, including t_k, as well as tasks with earlier deadlines within n_i. Each task processed locally is confined to its respective SED, as indicated by Equation 3.

$$f_k^{SED} = \sum_{i=1}^{SED} m_{i,k} \left(\sum_{k'=1}^{k} (x_{i,k'} \cdot \frac{c_{k'}}{p_i}) \right), \quad 1 \leq k \leq TS \tag{2}$$

$$x_{i,k} \leq m_{i,k}, \quad 1 \leq i \leq SED, \quad 1 \leq k \leq TS \tag{3}$$

The energy utilization cost can be obtained as per Eq. 4 [35]

$$E^{SED} = EN^{SED} (p_{SED})^2 C_k \tag{4}$$

where EN^{SED} is the energy consumed per CPU cycle and c_k (MIPS) denotes the workload of SED.

2) ES TASK PROCESSING

During transfer of a task to an ES, its processing time includes both the duration for computing latencies and input data transmission. In our investigation, we neglect the time needed for output data transfer, assuming that the output size resulting from computation is typically smaller compared to the input data size for a given task.

For task t_k offloaded to ES n_i ($SED+1 \leq i \leq SED+ES$), the data transfer period is determined by $p_k / \sum_{j=1}^{SED} m_{j,k} \cdot bw_{j,i}$,

the term $\sum_{j=1}^{SED} m_{j,k} \cdot bw_{j,i}$ denotes the data transfer rate between the ES and the SED initiating t_k, and the computing latency is $\frac{c_k}{p_i}$. To avoid performance conflicts, data transfers for the tasks offloaded to an ES are executed in sequence. Thus, following the EDF processing order, the total time for data transfer for task t_k when offloaded to an ES can be calculated using Equation 5.

$$f_k^{ES\_NET} = \sum_{i=SED+1}^{SED+ES} \left(x_{i,k} \cdot \sum_{k'=1}^{k} x_{i,k'} \cdot \left(\frac{p_k}{\sum_{j=1}^{SED} m_{j,k} \cdot bw_{j,i}} \right) \right) \tag{5}$$

where $1 \leq k \leq TS$.

Once a task is offloaded to an ES, computing can begin only after the input data transmission has been completed. Therefore, the task's finish time on an ES is determined by Eq. 6, where $\max_{1=k'<k} x_{i,k'} \cdot f_{ES_k}$ represents the finish time of tasks processed before t_k.

$$f_k^{ES} = \sum_{i=SED+1}^{SED+ES} \left(x_{i,k} \cdot \left(\max \left(f_k^{ES_N ET} \cdot \max_{1=k'<k} x_{i,k'} \cdot f_k^{ES} \right) + \frac{c_k}{p_i} \right) \right) \tag{6}$$

where $1 \leq k \leq TS$.

Each task can only be offloaded to an ES that has a network connection to its corresponding device. Therefore, Equation 7 must be fulfilled.

$$x_{i,k} = \sum_{i'=1}^{SED} m_{i',k} \cdot a_{i',i}, \quad SED + 1 \leq i \leq SED + ES, \quad 1 \leq k \leq TS \tag{7}$$

Energy-Cost computations are performed as follows:

Let $C = \{c_1, c_2, \ldots, c_\omega, \ldots, C\}$ denotes the set of manageable sub-carriers (SubC) with respective bandwidth bw_ω to transfer task k to an ES. The values of $\omega \in C, k \in J, s \in \Phi$ indicate the offloading points like $\omega_{ks} = 1$ shows that task k is offloaded to an ES s by SubC ω; otherwise, $\omega_{ks} = 0$. In addition, $R = \{P_{\omega ks} : P_{\omega ks} \in [0, P_{\text{budget}}], \omega \in C, k \in J, s \in \Phi\}$ represents power allocation matrix. Here the term $P_{\omega ks}$ shows power assigned to task k uploaded to ES. Therefore, the task uploading strategy must satisfy the constraint expressed in Eq. 8:

$$\sum_{s \in \Phi} \sum_{\omega \in C} \Omega_{\omega ks} \leq 1, \quad k \in J \tag{8}$$

Let $P_{SED,ES}$ denote the power required for transferring a job k to ES, in Watts, and $P_{s,j}$ represent the power needed for communication burden between ESs. The energy consumed for transmiting ($E_{SED,ES}$) is calculated as $E_{SED,ES} = P_{SED,ES} \cdot D_{SED,ES} = P_{SED,ES} \cdot \frac{s_k}{R} \cdot \eta$. Similarly, the energy consumed for communication burden between ECs can be expressed as $E_{ES,j} = P_{ES,j} \cdot \frac{s_k}{R_s}$. If the task is not forwarded to other nearby servers, then $P_{ES,j} = 0$. Additionally, the execution energy cost for tasks that get offloaded to a specific server is denoted as E_e, which can be calculated by $E_e = E_{\text{ser}} \cdot (f_{\text{ser}})^2 \cdot c_k$, where E_{ser} represents the energy cost/CPU cycle. Hence, the total energy cost (E_{off}) for the offloaded task can be determined using Equation 9.

$$E_{\text{off}} = E_{SED,ES} + E_{ES,j} + E_e \tag{9}$$

C. PROBLEM FORMULATION

This research aims to discover a design that optimizes energy efficiency by reducing the system's overall energy expenditure, all while ensuring adherence to capacity and delay limitations. This objective is formally expressed in Eq. 10.

$$\min_{x_k, y_k} \sum_{k=1}^{N} \left(\sum_{i=1}^{Q} x_k E_L + \sum_{s=1}^{S} y_k E_{\text{off}} \right) \tag{10}$$

Here, N variable represents the total number of tasks within the system that need to be processed. Q and S variables represent the number of local execution resources and offloaded resources, respectively. These resources could be processing units (CPUs, GPUs) or memory units, depending on the specific system being analyzed. The equation iterates through each task (k) from 1 to N. Each task considers two options: local execution or offloading. The decision variables x_k and y_k determine which option is chosen for each task.

The decision variables are x_k and y_k, which determine whether a task is executed locally or offloaded to an edge server, respectively. Below are the typical constraints that might be associated with these variables in the context of capacity and delay limitations:

$$x_k, y_k \in \{0, 1\} \quad \forall k \in \{1, 2, \ldots, N\} \tag{11}$$

This constraint ensures that each task k is either executed locally ($x_k = 1$) or offloaded ($y_k = 1$), but not both.

$$x_k + y_k = 1 \quad \forall k \in \{1, 2, \ldots, N\} \tag{12}$$

This ensures that each task k is either assigned to local execution or offloaded, but not both.

$$\sum_{k=1}^{N} x_k \leq C_{\text{local}} \tag{13}$$

This limits the total number of tasks that can be processed locally to the local processing capacity C_{local}.

$$\sum_{k=1}^{N} y_k \leq C_{\text{offload}} \tag{14}$$

This limits the total number of tasks that can be offloaded to the edge servers' processing capacity C_{offload}.

$$T_{\text{local},k} x_k \leq D_k \quad \forall k \in \{1, 2, \ldots, N\} \tag{15}$$

This ensures that the processing time for tasks executed locally does not exceed their respective deadlines D_k.

$$T_{\text{offload},k} y_k \leq D_k \quad \forall k \in \{1, 2, \ldots, N\} \tag{16}$$

This ensures that the processing time for tasks offloaded to edge servers does not exceed their respective deadlines D_k.

$$\sum_{k=1}^{N} x_k R_{\text{local},k} \leq R_{\text{local,max}} \tag{17}$$

This ensures that the total resource consumption of tasks executed locally does not exceed the maximum available local resources $R_{\text{local,max}}$.

$$\sum_{k=1}^{N} y_k R_{\text{offload},k} \leq R_{\text{offload,max}} \tag{18}$$

This ensures that the total resource consumption of tasks offloaded to edge servers does not exceed the maximum available offload resources $R_{\text{offload,max}}$.

These constraints collectively ensure that the optimization problem respects the limitations on capacity and delay, while striving to minimize the overall energy consumption of the system.

IV. THE PROPOSED TASK OFFLOADING ALGORITHM

The optimization problem defined in Eq. 10 presents a formidable challenge due to its nonlinear nature and NP-hard complexity. Finding the optimal solution to such a problem is notoriously difficult and time-consuming. Hence, it becomes imperative to devise an efficient method to address this challenge effectively.

Several methodologies have been devised to address NP-hard problems, encompassing techniques like PSO, GA, ACO, and GWO. Among these, PSO stands out for its widespread use in hybrid algorithms, thanks to its efficient global optimization capabilities, rapid convergence rate, and ease of implementation. On the other hand, GWO has proven to be effective in solving diverse real-world problems, including resource allocation and scheduling.

In our study, we propose the Hybrid Task Offloading (HybridTO) approach to tackle the optimization problem at hand. HybridTO is a hybrid method that synergistically combines the strengths of PSO and GWO. By combining the unique attributes of these methods, HybridTO seeks to augment the exploration capabilities across the solution space and enhance convergence rates towards optimal solutions.

The PSO algorithm is renowned for its efficiency in global optimization and straightforward implementation [14]. Meanwhile, GWO excels in exploring the search space by emulating the leadership hierarchy observed in grey wolf packs [15]. By integrating these methods, our goal is to leverage their combined strengths to achieve superior results, faster convergence, and more comprehensive exploration of the global optimum.

The PSO algorithm, as a stochastic swarm technique, derives inspiration from animal behaviours, especially in search and hunting, to solve global optimization problems. In our context, we apply this algorithm within an edge computing environment, where we aim to offload N tasks having the requirement of low latency and resources to ESs for processing. Here, each SED is conceptualized as a particle, leading to the representation of the problem as N-dimensional vectors.

Algorithm 1 Particle Swarm Optimization (PSO)

```
Initialize population of particles randomly within search space
Initialize velocity of particles randomly within predefined range
Set individual best position (p_best) of each particle to its initial position
Set global best position (g_best) to the position of the particle with the best fitness value
while termination condition not met do
    for each particle do
        Update velocity using:
            velocity = w · velocity + c1 · rand() · (p_best − position) + c2 · rand() · (g_best − position)
        Update position using:
            position = position + velocity
        if fitness of new position is better than p_best then
            Update p_best to new position
        end if
        if fitness of new position is better than g_best then
            Update g_best to new position
        end if
    end for
end while
return g_best as the solution
```

The pseudo-code of the PSO algorithm used in our study is outlined in Algorithm 1. This algorithm iteratively updates the positions of particles in the search space based on their individual and collective experiences, gradually converging towards optimal solutions.

The GWO algorithm is a sophisticated optimization technique inspired by the social structure and hunting behaviour observed in grey wolf packs. By emulating the hierarchical organization and cooperative hunting strategies of these animals, the GWO algorithm seeks to navigate complex search spaces and find optimal solutions efficiently.

In the GWO algorithm, the population is divided into four distinct groups, each representing a different level of leadership within the pack. These groups are denoted as α, β, δ, and ω, mirroring the hierarchical structure observed in wolf packs.

At the top of the hierarchy is the α wolf, which assumes the role of the leader responsible for making overarching decisions for the pack. The β wolves occupy the second-highest position in the hierarchy, acting as assistants to the alpha and providing guidance to lower-ranking members. The δ wolves hold the third rank and support the higher-ranking wolves by assisting in boundary observation, group defence, and hunting activities. Finally, the omega wolf occupies the lowest position in the hierarchy and is required to follow the directives of higher-ranking members.

Algorithm 2 Grey Wolf Optimization (GWO)

Initialize grey wolf population X^*, X^{alpha}, X^{beta}, X^{delta} within search space
Initialize positions of grey wolves randomly
Evaluate fitness of each grey wolf
while termination criterion not met **do**
  for all grey wolves X_i **do**
    if fitness(X_i) < fitness(X^{alpha}) **then**
      $X^{\text{alpha}} \leftarrow X_i$
    end if
    if fitness(X_i) > fitness(X^{alpha}) and fitness(X_i) < fitness(X^{beta}) **then**
      $X^{\text{beta}} \leftarrow X_i$
    end if
    if fitness(X_i) > fitness(X^{alpha}), fitness(X_i) > fitness(X^{beta}), and fitness(X_i) < fitness(X^{delta}) **then**
      $X^{\text{delta}} \leftarrow X_i$
    end if
  end for
  for all grey wolves X_i **do**
    for each dimension d **do**
      Update position using equation:
      $X_i[d] \leftarrow X^{\text{alpha}}[d] - \text{a}_{\text{howl}} \cdot \text{rand}() \cdot |X^{\text{alpha}}[d] - 2 \cdot X_i[d]|$
      $X_i[d] \leftarrow (X_i[d] + X^{\text{beta}}[d])/2$
      $X_i[d] \leftarrow (X_i[d] + X^{\text{alpha}}[d])/2$
      $X_i[d] \leftarrow (X_i[d] + X^{\text{delta}}[d])/2$
    end for
  end for
  Evaluate fitness of each updated grey wolf
end while
return Position of the alpha grey wolf X^{alpha} as the solution

The GWO algorithm employs three primary hunting strategies - searching, encircling, and attacking - to explore and exploit the search space effectively. These strategies enable the grey wolf pack to collaboratively navigate the environment, identify promising regions, and converge towards optimal solutions. Through the coordinated efforts of the α, β, δ, and ω, wolves, the GWO algorithm iteratively refines its search and converges towards high-quality solutions.

In this work, we propose HybridTO, a novel hybrid algorithm combining PSO and GWO to optimize task offloading strategies in energy-constrained edge computing environments. HybridTO leverages the complementary strengths of PSO and GWO to enhance optimization performance. PSO is renowned for its robust global search capabilities, which allow it to efficiently explore the search space to avoid local optima. At the same time, GWO excels in local search, providing robust exploitation to fine-tune solutions. By integrating these two methods, HybridTO effectively balances exploration and exploitation, potentially leading to superior optimization performance. This hybrid approach often achieves faster convergence and higher-quality solutions, particularly in complex and multimodal optimization problems. However, this performance enhancement comes with increased complexity. HybridTO requires additional mechanisms to coordinate PSO and GWO, resulting in higher computational costs and more intricate parameter tuning. Despite these challenges, the improvement in solution quality and convergence speed justifies the added complexity, making HybridTO a powerful tool for optimization in energy-constrained edge computing environments.

The HybridTO algorithm, as outlined in Algorithm 3, orchestrates the process of optimizing energy consumption and resource allocation for efficient task offloading. Let's delve into the details of each step:

1) Initialization: The algorithm starts by gathering all necessary information and setting the required parameters. It initializes the sub-carrier assignment during the first iteration, assigns random initial velocities and locations to each particle/device, and evaluates their corresponding fitness values. Moreover, it keeps track of a vector, $pBest_i$, to monitor each particle's best locations individually, along with a $gBest$ vector to record the best result in the current population. The decision regarding offloading is subsequently made by assessing the task processing costs and taking into account energy preferences and delay.
2) Iterative Execution Loop: The algorithm enters an iterative loop, continuing until the termination criteria are met. During each iteration of the loop, the global best and individual best values for every particle are adjusted. Moreover, it updates the velocity $V_i(t+1)$ and location $\lambda_i(t+1)$ for each particle based on their previous values and the best values found so far.
3) Return Optimized Results: Upon completion of the iterative loop, the algorithm provides energy usage results. These results provide valuable insights required for efficiently allocating resources, including sub-carriers and power consumption, for the offloading process.
4) Resource Allocation and Offloading Decision: The controller server responds to the service requester (end-user) based on the optimized results obtained from the algorithm. The SED owning the task determines whether the task must be calculated locally or offloaded to ES. Afterwards, the SEDs request the necessary resources from the controller server, which oversees the resource allocation table and handles task scheduling. Upon receiving the request, the controller server assigns the optimal ES within the collaborative environment. Subsequently, the SEDs transfer the task to the chosen ES for execution.
5) Optimization Phases: The algorithm improves resource allocation and energy consumption through two main steps. Initially, it sets up the locations of the SEDs, which are depicted as a vector with a dimensionality of size N.

Algorithm 3 HybridTO Algorithm

Initialize: Set required parameters, determine preliminary sub-carrier assignment.
Randomly initialize velocities and locations for each particle/device.
Evaluate fitness values for each particle/device.
Initialize $pBest_i$ vector to track individual particles' best locations.
Initialize $gBest$ vector to track particle with best result.
Make offloading decision based on task processing costs.
while Termination criteria not met **do**
    for Each particle/device **do**
        Update each and global best values.
        Update velocity $V_i(t+1)$ and location $\lambda_i(t+1)$.
    end for
end while
Return: Optimized energy utilization results.

Analyzing the time complexity of a hybrid PSO-GWO algorithm for task offloading in edge computing involves considering the individual complexities of PSO and GWO, along with the overhead introduced by combining them. PSO's complexity primarily depends on the number of particles (N) and the number of iterations (T), resulting in a complexity of $O(N \times T)$ due to position and velocity updates for each particle. Similarly, GWO's complexity, determined by the population size (N) and iterations (T), also stands at $O(N \times T)$ due to the calculations in its hunting, encircling, attacking, and fleeing phases. When combined, the hybrid algorithm's complexity depends on the integration method: sequential execution results in an additive complexity of $O(N \times T)$, while interleaved execution, involving additional overhead, maintains the same order of magnitude. Additional factors such as specific implementation details and the complexity of fitness function evaluations might slightly vary the overall complexity. However, the hybrid PSO-GWO algorithm is typically considered to have a time complexity of $O(N \times T)$, where N is the population size and T is the number of iterations.

By following this comprehensive process, the HybridTO algorithm efficiently manages task offloading in edge computing environments, leading to enhanced energy efficiency and resource utilization.

Combining PSO and GWO for task offloading in edge cloud environments offers distinct advantages over using either method alone. The hybrid approach leverages PSO's strong global exploration capabilities and GWO's efficient local exploitation, resulting in a more balanced search process that improves the likelihood of finding optimal solutions. This combination enhances convergence speed, as PSO's rapid initial exploration helps quickly identify promising regions, while GWO's precise local search refines these solutions more effectively. Additionally, the hybrid method mitigates the risk of getting trapped in local optima, a common issue with individual algorithms, thereby increasing robustness and reliability in complex, multimodal optimization landscapes. The integration of these algorithms also adapts better to the intricate and dynamic scenarios typical of edge cloud environments, where tasks, resource constraints, and network conditions are highly variable. This adaptability leads to more efficient resource utilization and energy savings, which are critical in energy-constrained edge devices. Overall, the combined PSO and GWO approach provides superior performance in terms of solution quality, convergence speed, and robustness, making it a more robust and versatile tool for optimizing task offloading in edge cloud environments.

V. EXPERIMENT AND RESULTS

In this section, we delve into a comprehensive evaluation of the effectiveness of our proposed HybridTO approach through a series of meticulously designed experiments. We kick-start by elucidating the intricate details of the experimental setup meticulously crafted for these assessments. Following this, we introduce and expound upon the metrics meticulously chosen to gauge and scrutinize the performance of our proposition. Lastly, we meticulously present and dissect the experimental results, unravelling invaluable insights and noteworthy observations gleaned during the rigorous evaluation phase.

A. EXPERIMENTAL SETUP AND PERFORMANCE METRICS

The table outlines the critical parameters employed in the experimental setup to evaluate the performance of the proposed approach, as depicted in Table 2. These parameters provide crucial insights into the configuration and characteristics of the environment under investigation.

Included in the table are essential specifications such as the number of ESs, the uplink transmission power ($P_{SED,ES}$), system bandwidth (bw), ES execution rate (f_{ser}), SEDs execution rate (f_{dev}), input data size (SZ_k), workload (W_k), and the average bandwidth of sub-carriers.

For instance, the experimental setup considers 15 ESs, each operating with an uplink transmission power of 20 dBm. The system bandwidth is set at 20 MHz to facilitate data transmission and processing. The ES execution rate varies within the range of 2 to 5 GHz, while SEDs execute tasks at a fixed rate of 1 GHz. The input data size ranges from 20 to 100 MB, indicating the volume of data processed by the system. Additionally, the workload, measured in CPU cycles, spans from 1×10^7 to 2.5×10^7, representing the computational demand imposed by the tasks. Lastly, the sub-carrier average bandwidth is determined to be 15 kHz, providing a crucial parameter for communication and resource allocation.

These meticulously defined parameters collectively establish the experimental framework and conditions necessary to evaluate the efficacy and performance of the proposed approach under various scenarios and configurations.

The evaluation of the proposed HybridTO approach involved a comprehensive analysis of multiple parameters

TABLE 2. Experimental Setup Parameters.

Parameter	Specification
Number of ESs	15
Bandwidth (bw)	20 MHz
Uplink transmission power ($P_{SED,ES}$)	20 dBm
ES execution rate (f_{ser})	[2,5] GHz
Input data size (SZ_k)	20 - 100 MB
SEDs execution rate (f_{dev})	1 GHz
Sub-carrier average bandwidth	15 kHz
Workload (W_k)	$(1-2.5)\times 10^7$ CPU cycles

critical to the performance of task offloading in edge computing environments. Key metrics such as processing time, energy consumption, and offloading utilization were meticulously scrutinized across varying task characteristics, including the number of tasks, task size, and transmission rate. These metrics were assessed using simulation results obtained through rigorous experimentation, with a comparative analysis conducted against established baseline approaches to provide a meaningful assessment of the proposed methodology's effectiveness.

In particular, the proposed HybridTO approach was subjected to evaluation alongside several baseline methods, each representing different offloading strategies commonly employed in edge computing scenarios. The baseline studies encompassed an optimal enumeration offloading strategy (referred to as Base_OES) [36], a random offloading technique grounded in dynamic programming with hamming distance termination (referred to as Base_DPH) [37], and a local execution strategy at the SED.

Base_DPH operates by making offloading decisions randomly based on resource availability, utilizing hamming-distance termination to guide the offloading process. On the other hand, Base_OES adopts an exhaustive enumeration approach, systematically exploring all potential offloading solutions to identify the optimal server for task offloading. These baseline methods serve as benchmarks against which the performance of the proposed HybridTO algorithm is assessed, facilitating a comprehensive understanding of its relative strengths and weaknesses.

B. RESULTS AND ANALYSIS

In this section, we thoroughly examine the simulation results to evaluate the advantages provided by the proposed approach across various parameters. These parameters include input data size, the number of ESs, processing capacity, and data uploading rate.

Fig. 2 presents a comparative analysis of the computational complexity between the proposed HybridTO and Base_OES concerning varying numbers of tasks while maintaining a constant task-input size. Base_OES, although capable of providing an optimal solution for offloading decisions, suffers from an exponential time complexity of $O(3^n)$, where

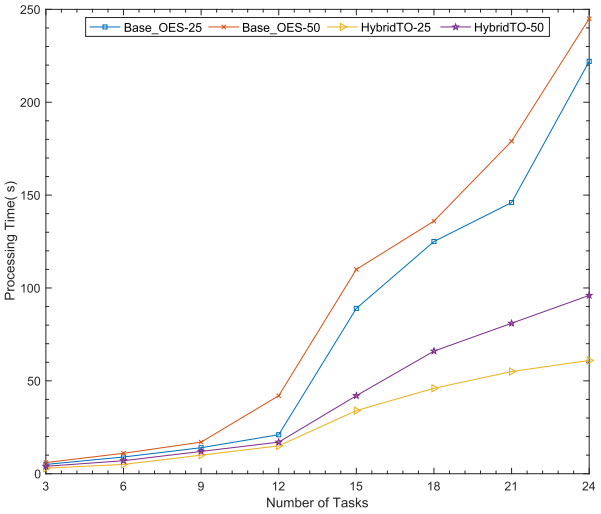

FIGURE 2. Processing Time Vs. Number of Tasks.

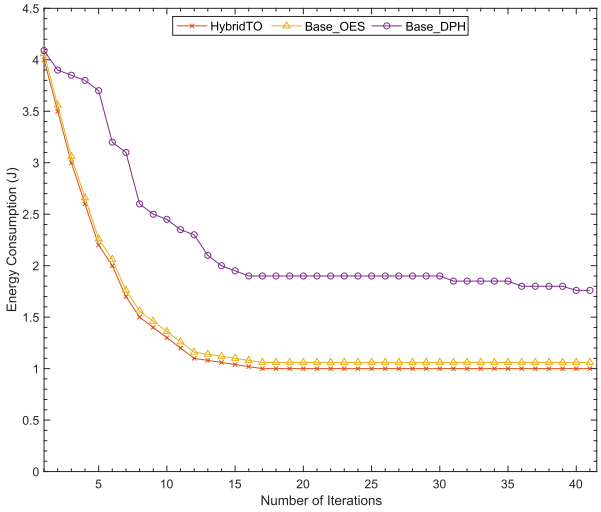

FIGURE 3. Energy Consumption Vs. Number of Iterations.

n represents the number of tasks. Consequently, Base_OES becomes impractical in scenarios involving a large number of tasks due to its substantial computational requirements. As such, in this study, we leverage Base_OES as a benchmark algorithm for comparison with the proposed HybridTO.

Simulation results demonstrate that the proposed HybridTO algorithm outperforms Base_OES regarding execution speed when the task-input size is set to 25 MB and 50 MB. Initially, for task counts ranging from 3 to 12, both Base_OES and HybridTO exhibit nearly identical execution times. However, as the number of tasks escalates, the proposed HybridTO begins to showcase significantly faster execution speeds compared to Base_OES, with the performance disparity widening considerably. This highlights the better performance of the proposed HybridTO over Base_OES, particularly in scenarios with numerous tasks.

In Figure 3, the evaluation of performance juxtaposes the proposed HybridTO algorithm with benchmark algorithms

regarding optimal energy usage, and statistical output for executing user tasks. A notable observation is that in comparison to the Base_DPH methods, the HybridTO algorithm exhibits results closely resembling those of the optimal Base_OES approach while showcasing superior energy utilization within fewer iterations, particularly up to 100 iterations. This achievement stems from the incorporation of a penalty function within the fitness function, which facilitates the identification of the optimal solution by simultaneously considering resource availability and task computation costs. Moreover, the inclusion of the penalty function expedites iterative convergence and enhances result accuracy, contributing to the algorithm's effectiveness in achieving optimal energy consumption levels.

In Figure 4, the influence of task size on energy consumption is illustrated. As the task size increases, the difference in energy consumption ratio decreases. This phenomenon occurs because more extensive input data requires more resources for both uploading and processing tasks, resulting in higher energy consumption compared to smaller tasks. However, all methods demonstrate a comparable trend of decreasing disparities in energy consumption ratio with an increase in task size. The HybridTO approach closely aligns with Base_OES while outperforming the baseline scheme Base_DPH. This underscores the significance of efficient energy utilization achieved by the proposed strategy, leading to enhanced data uploading rates and optimized energy savings for the system.

Figure 5 illustrates the relationship between the processing capacity required and total energy utilization for task completion. It is clear that as the processing capacity requirement increases, total energy utilization also increases across all approaches. Local execution shows higher energy usage in comparison to other methods, primarily due to the limitations in computing and energy resources at the SEDs. Transferring resource-intensive tasks to ESs presents a promising approach to conserving energy at the SEDs. In the proposed HybridTO approach, total energy utilization closely aligns with Base_OES and is lower than Base_DPH and Local Execution. Tasks are offloaded to ESs that utilize cooperative task processing and maximize resource usage at the network edge. This conserves energy that would otherwise be spent on transmitting jobs through the backhaul link to remote servers.

Moreover, the offloading decision takes into account capacity constraints, proximity, and time constraints, effectively minimizing transmission energy over long distances and enhancing overall energy efficiency. By intelligently allocating tasks based on these considerations, the HybridTO approach optimizes energy consumption while ensuring timely task completion and resource utilization.

Figure 6 illustrates a comparison between the HybridTO approach and two baseline methods, Base_OES, and Base_DPH, regarding offloading utility across different data transmission rates. Significantly, the offloading utility exhibits an upward trend as the transmission rate increases across all approaches, although nuanced differences are noticeable, particularly at lower transmission rates.

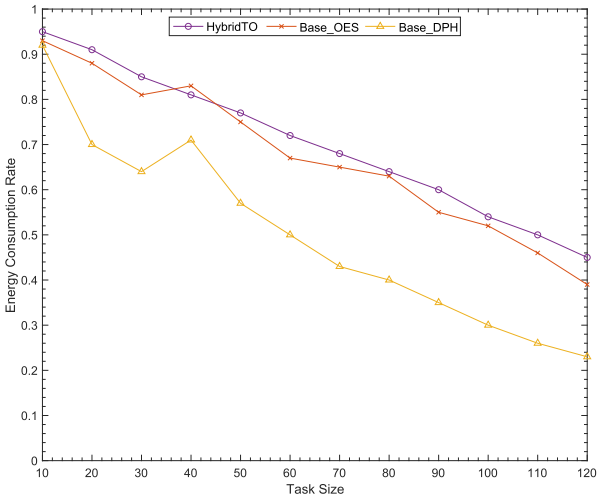

FIGURE 4. Energy Consumption Rate Vs. Task Size.

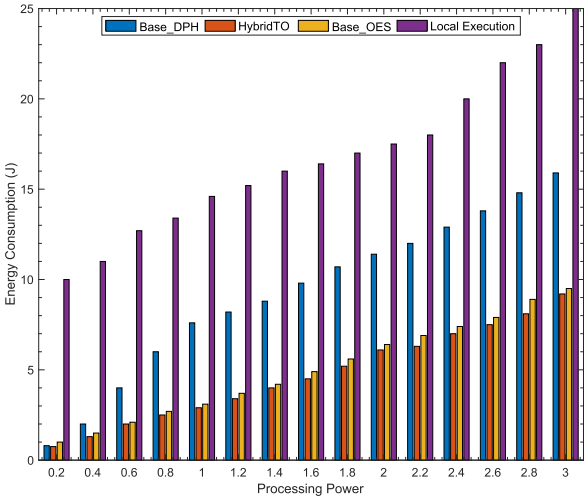

FIGURE 5. Energy Consumption Vs. Processing Power.

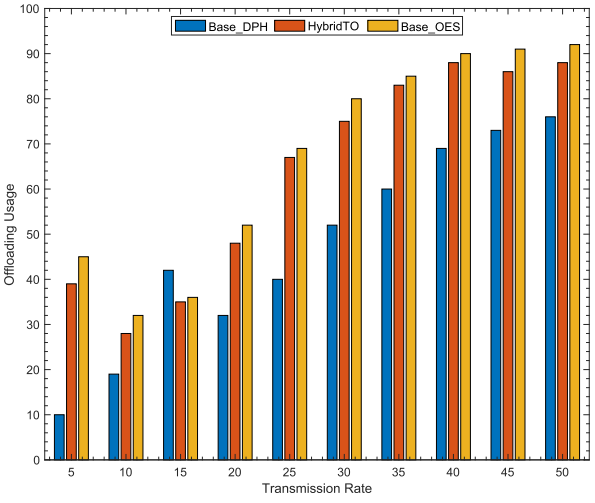

FIGURE 6. Offloading Usage Vs. Transmission Rate.

In each scenario, HybridTO matches the performance of Base_OES and outperforms other baseline methods like Base_DPH regarding offloading utility. This advantage stems from HybridTO's optimization strategy, which strategically utilizes ESs situated closer to end users for task execution. By reducing latency and resource consumption, such as energy and bandwidth, compared to remote cloud servers, HybridTO boosts the uploading rate of SEDs, ultimately enhancing the offloading utility. Furthermore, the cooperative capabilities inherent in ESs play a pivotal role in this context. These capabilities ensure the availability of ample resources at the edge, facilitating more efficient task offloading to ESs and minimizing resource consumption by SEDs. As a result, the HybridTO approach optimizes offloading utility by intelligently distributing tasks across the network infrastructure, leveraging both proximity and resource availability to enhance overall system performance.

VI. CONCLUSION AND FUTURE DIRECTIONS

In this study, we conducted a thorough exploration of energy-efficient task offloading strategies within EC environments. Through extensive experimentation and analysis, we scrutinized the performance of the proposed HybridTO approach against several baseline methods, namely Base_DPH, Base_OES, and Local Execution. Our comprehensive evaluation revealed that HybridTO offers substantial improvements over these baseline methods across various metrics, including energy utilization, response delay, and offloading utility. Particularly noteworthy is HybridTO's superiority in scenarios characterized by a large number of tasks or diverse task sizes.

One key finding of our analysis is the importance of leveraging ESs located closer to end users to minimize latency and resource consumption while maximizing offloading utility. By strategically distributing tasks across the network infrastructure and capitalizing on the proximity of ESs to end users, HybridTO demonstrates remarkable efficiency gains compared to traditional offloading approaches.

Overall, our results underscore HybridTO's potential as a highly effective solution for energy-efficient task offloading in EC environments. The implications of our findings extend to future research and development efforts in this domain, promising advancements in optimizing resource utilization and enhancing system performance.

Looking ahead, our focus will be on extending the applicability of HybridTO for energy-efficient task offloading in edge computing. This includes expanding network traffic datasets and using network emulation tools to create more realistic testing conditions, along with evaluating HybridTO's performance on diverse hardware platforms to ensure its scalability and efficiency across various edge computing environments. One critical extension of this work could involve exploring the integration of HybridTO with emerging technologies like Stacked Intelligent Metasurfaces (SIM) [38]. SIM utilizes intelligent metasurfaces to manipulate electromagnetic waves dynamically. While this technology is currently not directly applicable to network traffic classification, its potential for Over-The-Air (OTA) computing holds promise for future advancements in edge computing frameworks. By incorporating SIM capabilities, HybridTO could potentially offload specific computational tasks to the intelligent metasurfaces as data propagates through them. This could lead to benefits such as reduced energy consumption on traditional servers, improved task processing efficiency at the edge, and, ultimately, a more robust and efficient edge computing environment. Further research is needed to explore the feasibility and potential benefits of integrating SIM with HybridTO for network traffic classification tasks.

REFERENCES

[1] M. P. J. Mahenge, C. Li, and C. A. Sanga, "Energy-efficient task offloading strategy in mobile edge computing for resource-intensive mobile applications," *Digit. Commun. Netw.*, vol. 8, no. 6, pp. 1048–1058, Dec. 2022.

[2] M. Teng, X. Li, and K. Zhu, "Joint optimization of sequential task offloading and service deployment in end-edge-cloud system for energy efficiency," *IEEE Trans. Sustain. Comput.*, vol. 9, no. 3, pp. 1–16, May/Jun. 2024.

[3] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet Things J.*, vol. 3, no. 5, pp. 637–646, Oct. 2016.

[4] P. Dai, K. Hu, X. Wu, H. Xing, and Z. Yu, "Asynchronous deep reinforcement learning for data-driven task offloading in MEC-empowered vehicular networks," in *Proc. IEEE Conf. Comput. Commun.*, May 2021, pp. 1–10.

[5] C. Zeng, X. Wang, R. Zeng, Y. Li, J. Shi, and M. Huang, "Joint optimization of multi-dimensional resource allocation and task offloading for QoE enhancement in cloud-edge-end collaboration," *Future Gener. Comput. Syst.*, vol. 155, pp. 121–131, Jun. 2024.

[6] Y. Ding, K. Li, C. Liu, and K. Li, "A potential game theoretic approach to computation offloading strategy optimization in end-edge-cloud computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 33, no. 6, pp. 1503–1519, Jun. 2022.

[7] J. Almutairi and M. Aldossary, "A novel approach for IoT tasks offloading in edge-cloud environments," *J. Cloud Comput.*, vol. 10, no. 1, p. 28, Apr. 2021.

[8] Y. He, D. Zhai, R. Zhang, J. Du, G. S. Aujla, and H. Cao, "A mobile edge computing framework for task offloading and resource allocation in UAV-assisted VANETs," in *Proc. IEEE Conf. Comput. Commun. Workshops*, May 2021, pp. 1–6.

[9] S. Long, Y. Zhang, Q. Deng, T. Pei, J. Ouyang, and Z. Xia, "An efficient task offloading approach based on multi-objective evolutionary algorithm in cloud-edge collaborative environment," *IEEE Trans. Netw. Sci. Eng.*, vol. 10, no. 2, pp. 645–657, Mar. 2023.

[10] K. Thakur and G. Kumar, "Nature inspired techniques and applications in intrusion detection systems: Recent progress and updated perspective," *Arch. Comput. Methods Eng.*, vol. 28, no. 4, pp. 2897–2919, Jun. 2021.

[11] C. Wang, R. Guo, H. Yu, Y. Hu, C. Liu, and C. Deng, "Task offloading in cloud-edge collaboration-based cyber physical machine tool," *Robot. Comput.-Integr. Manuf.*, vol. 79, Feb. 2023, Art. no. 102439.

[12] X. Cao, F. Wang, J. Xu, R. Zhang, and S. Cui, "Joint computation and communication cooperation for mobile edge computing," in *Proc. 16th Int. Symp. Model. Optim. Mobile, Ad Hoc, Wireless Netw.*, May 2018, pp. 1–6.

[13] K. Cheng, Y. Teng, W. Sun, A. Liu, and X. Wang, "Energy-efficient joint offloading and wireless resource allocation strategy in multi-MEC server systems," in *Proc. IEEE Int. Conf. Commun.*, May 2018, pp. 1–6.

[14] D. Wang, D. Tan, and L. Liu, "Particle swarm optimization algorithm: An overview," *Soft Comput.*, vol. 22, no. 2, pp. 387–408, Jan. 2018.

[15] S. Mirjalili, S. M. Mirjalili, and A. Lewis, "Grey wolf optimizer," *Adv. Eng. Softw.*, vol. 69, pp. 46–61, Mar. 2014.

[16] Y. Sang, J. Cheng, B. Wang, and M. Chen, "A three-stage heuristic task scheduling for optimizing the service level agreement satisfaction in device-edge-cloud cooperative computing," *PeerJ Comput. Sci.*, vol. 8, p. e851, Jan. 2022.

[17] B. Wang, J. Cheng, J. Cao, C. Wang, and W. Huang, "Integer particle swarm optimization based task scheduling for device-edge-cloud cooperative computing to improve SLA satisfaction," *PeerJ Comput. Sci.*, vol. 8, p. e893, Feb. 2022.
[18] T. Gao, Q. Tang, J. Li, Y. Zhang, Y. Li, and J. Zhang, "A particle swarm optimization with lévy flight for service caching and task offloading in edge-cloud computing," *IEEE Access*, vol. 10, pp. 76636–76647, 2022.
[19] B. Wang, B. Lv, and Y. Song, "A hybrid genetic algorithm with integer coding for task offloading in edge-cloud cooperative computing," *IAENG Int. J. Comput. Sci.*, vol. 49, no. 2, pp. 503–510, 2022.
[20] S. Chakraborty and K. Mazumdar, "Sustainable task offloading decision using genetic algorithm in sensor mobile edge computing," *J. King Saud Univ. Comput. Inf. Sci.*, vol. 34, no. 4, pp. 1552–1568, Apr. 2022.
[21] J. Yadav and Suman, "E-MOGWO algorithm for computation offloading in fog computing," *Intell. Autom. Soft Comput.*, vol. 36, no. 1, pp. 1063–1078, 2023.
[22] Q. Xu, G. Zhang, and J. Wang, "Research on cloud-edge-end collaborative computing offloading strategy in the Internet of Vehicles based on the M-TSA algorithm," *Sensors*, vol. 23, no. 10, p. 4682, May 2023.
[23] C. Li, J. Xia, F. Liu, D. Li, L. Fan, G. K. Karagiannidis, and A. Nallanathan, "Dynamic offloading for multiuser muti-CAP MEC networks: A deep reinforcement learning approach," *IEEE Trans. Veh. Technol.*, vol. 70, no. 3, pp. 2922–2927, Mar. 2021.
[24] M. Chen and Y. Hao, "Task offloading for mobile edge computing in software defined ultra-dense network," *IEEE J. Sel. Areas Commun.*, vol. 36, no. 3, pp. 587–597, Mar. 2018.
[25] X. Qin, B. Li, and L. Ying, "Distributed threshold-based offloading for large-scale mobile cloud computing," in *Proc. IEEE Conf. Comput. Commun.*, May 2021, pp. 1–10.
[26] H. Shah-Mansouri and V. W. S. Wong, "Hierarchical fog-cloud computing for IoT systems: A computation offloading game," *IEEE Internet Things J.*, vol. 5, no. 4, pp. 3246–3257, Aug. 2018.
[27] Z. Sun, H. Yang, C. Li, Q. Yao, D. Wang, J. Zhang, and A. V. Vasilakos, "Cloud-edge collaboration in industrial Internet of Things: A joint offloading scheme based on resource prediction," *IEEE Internet Things J.*, vol. 9, no. 18, pp. 17014–17025, Sep. 2022.
[28] J. Ren, G. Yu, Y. He, and G. Y. Li, "Collaborative cloud and edge computing for latency minimization," *IEEE Trans. Veh. Technol.*, vol. 68, no. 5, pp. 5031–5044, May 2019.
[29] W. Zhan, C. Luo, J. Wang, C. Wang, G. Min, H. Duan, and Q. Zhu, "Deep-reinforcement-learning-based offloading scheduling for vehicular edge computing," *IEEE Internet Things J.*, vol. 7, no. 6, pp. 5449–5465, Jun. 2020.
[30] Y.-H. Kao, B. Krishnamachari, M.-R. Ra, and F. Bai, "Hermes: Latency optimal task assignment for resource-constrained mobile computing," *IEEE Trans. Mobile Comput.*, vol. 16, no. 11, pp. 3056–3069, Nov. 2017.
[31] S. Sundar and B. Liang, "Offloading dependent tasks with communication delay and deadline constraint," in *Proc. IEEE Conf. Comput. Commun.*, Apr. 2018, pp. 37–45.
[32] G. Zhao, H. Xu, Y. Zhao, C. Qiao, and L. Huang, "Offloading dependent tasks in mobile edge computing with service caching," in *Proc. IEEE Conf. Comput. Commun.*, Jul. 2020, pp. 1997–2006.
[33] Z. Wang, W. Liang, M. Huang, and Y. Ma, "Delay-energy joint optimization for task offloading in mobile edge computing," 2018, *arXiv:1804.10416*.
[34] Y. Laili, F. Guo, L. Ren, X. Li, Y. Li, and L. Zhang, "Parallel scheduling of large-scale tasks for industrial cloud–edge collaboration," *IEEE Internet Things J.*, vol. 10, no. 4, pp. 3231–3242, Feb. 2023.
[35] Y. Wen, W. Zhang, and H. Luo, "Energy-optimal mobile application execution: Taming resource-poor mobile devices with cloud clones," in *Proc. IEEE INFOCOM*, Mar. 2012, pp. 2716–2720.
[36] H. Guo, J. Liu, and H. Qin, "Collaborative mobile edge computation offloading for IoT over fiber-wireless networks," *IEEE Netw.*, vol. 32, no. 1, pp. 66–71, Jan. 2018.
[37] H. Shahzad and T. H. Szymanski, "A dynamic programming offloading algorithm for mobile cloud computing," in *Proc. IEEE Can. Conf. Electr. Comput. Eng.*, May 2016, pp. 1–5.
[38] J. An, C. Yuen, C. Xu, H. Li, D. W. K. Ng, M. D. Renzo, M. Debbah, and L. Hanzo, "Stacked intelligent metasurface-aided MIMO transceiver design," *IEEE Wireless Commun.*, vol. 13, no. 1, pp. 1–9, Apr. 2024.

DEAFALLAH ALSADIE (Member, IEEE) received the B.Sc. degree in computer science from Taibah University, in 2007, the M.A.Sc. degree in computer science from Latrobe University, Australia, in 2011, and the Ph.D. degree in computer science from RMIT University, Melbourne, VIC, Australia, in 2019. He is currently an Assistant Professor with the Department of Computer Science and Artificial Intelligence, College of Computing, Umm Al-Qura University, Makkah, Saudi Arabia. His research interests include scheduling and resource allocation for parallel and distributed computing systems, data centers, edge computing, and the Internet of Things.