Published: 10 May 2025
Online AM: 16 March 2025
Accepted: 02 February 2025
Revised: 22 November 2024
Received: 21 June 2024

RESEARCH-ARTICLE

Sustainable Dependent Sub-Tasks Orchestration at Extreme Edge Computing: A Partitioning-based Deep Reinforcement Learning Approach

ZAHRA SAFAVIFAR, University College Dublin, Dublin, Leinster, Ireland

CHARAFEDDINE MACHALIKH, Kasdi Merbah Ouargla University, Ouargla, Ouargla, Algeria

JUNFEI XIE, San Diego State University, San Diego, CA, United States

FATEMEH GOLPAYEGANI, University College Dublin, Dublin, Leinster, Ireland

Sustainable Dependent Sub-Tasks Orchestration at Extreme Edge Computing: A Partitioning-based Deep Reinforcement Learning Approach

ZAHRA SAFAVIFAR, Computer science, University College Dublin, Dublin, Ireland
CHARAFEDDINE MACHALIKH, New Technologies of Information and Telecommunication school, Universite Kasdi Merbah Ouargla, Ouargla, Algeria
JUNFEI XIE, Electrical and Computer Engineering school, San Diego State University, San Diego, United States
FATEMEH GOLPAYEGANI, University College Dublin, Dublin, Ireland

Extreme Edge Computing (EEC) promotes sustainable computing by reducing reliance on centralized data centres and decreasing their environmental impact. By using extreme edge devices to handle computing requests, the EEC reduces the energy demands for data transmission and execution, thereby reducing carbon footprints. However, EEC introduces challenges due to the mobile, heterogeneous, and resource-limited nature of these devices. Additionally, tasks are often complex and interdependent, complicating offloading and workload orchestration. The dynamicity of EEC systems, where both task generation and resources can be mobile, alongside task inter-dependencies, escalates the complexity of task offloading and workload management. To tackle these complexities, task partitioning emerges as a viable strategy. Moreover, in dynamic edge computing scenarios, resource demand remains unpredictable, emphasizing the critical need to optimize resource utilization efficiently.

In this article, we investigate the problem of tasks with inter-dependencies offloading in an EEC environment where mobile and resource-constrained edge devices are employed as computing resources. In this regard, a partitioning-based Deep Reinforcement Learning (DRL) for Dependent sub-Task Orchestration (DeTOrch) model is proposed. DeTOrch uses a state-of-the-art partitioning method for decomposing tasks and proposes a novel mobility task-orchestration mechanism to minimize the task completion time and maximize the use of edge devices' resource. The simulation results show that the proposed model can significantly improve the task success rate and decrease task completion time. In addition, in various scenarios with different levels of mobility, the proposed model outperforms the baselines while utilizing the resource of edge devices.

CCS Concepts: • **Computing methodologies → Planning under uncertainty;**

Additional Key Words and Phrases: Edge computing, dependent tasks orchestration, deep reinforcement learning (DRL), task partitioning, extreme edge computing

This publication has emanated from research supported in part by a grant from Science Foundation Ireland under Grant number 18/CRT/6183 and National Science Foundation under Grant CAREER-2048266.
Authors' Contact Information: Zahra Safavifar, Computer science, University College Dublin, Dublin, Dublin, Ireland; e-mail: zahra.safavifar@ucdconnect.ie; Charafeddine Machalikh, New Technologies of Information and Telecommunication school, Universite Kasdi Merbah Ouargla, Ouargla, Algeria; e-mail: mechalikh.charafeddine@univ-ouargla.dz; Junfei Xie, Electrical and Computer Engineering school, San Diego State University, San Diego, California, United States; e-mail: jxie4@sdsu.edu; Fatemeh Golpayegani, University College Dublin, Dublin, Ireland; e-mail: fatemeh.golpayegani@ucd.ie.

ACM 2834-5533/2025/05-ART11
https://doi.org/10.1145/3723037

ACM Reference Format:
Zahra Safavifar, Charafeddine Machalikh, Junfei Xie, and Fatemeh Golpayegani. 2025. Sustainable Dependent Sub-Tasks Orchestration at Extreme Edge Computing: A Partitioning-based Deep Reinforcement Learning Approach. *ACM J. Comput. Sustain. Soc.* 3, 2, Article 11 (May 2025), 31 pages. https://doi.org/10.1145/3723037

1 Introduction

Edge computing is a promising solution that brings computation closer to the end-user. It involves using high-capacity servers, such as Multi-Access **Mobile Edge Computing (MEC)** servers, to manage users' demands and workload at the network's edge [14]. The increasing development and popularity of advanced IoT applications, such as **augmented reality (AR)**, online gaming, and autonomous vehicle-related applications, have led to a significant rise in computing and communication services demands [26]. Therefore, providing scalable and adequate computational resources at the network's edge is crucial.

However, increasing the number of MEC or other high-capacity servers poses sustainability-related concerns due to high maintenance costs and increased CO_2 emissions. Recent studies indicate that data centers account for about 1% of global electricity usage, which is projected to increase as digital services grow [16, 28]. This has led to an urgent call for sustainable computing solutions to reduce energy consumption and produce carbon emissions across computing infrastructures. In response, researchers have proposed sustainability-focused IoT applications, such as sustainable smart homes [5], smart agricultural applications [10], and other similar machine learning-based applications [32, 42, 43]. Although these applications contribute to sustainability, they cannot fully address the environmental impact caused by increasing computational demands. This challenge highlights the need for a sustainable computing paradigm, moving from centralized computing models to decentralized approaches. **Extreme Edge Computing (EEC)** supports this transition by minimizing the use of high-capacity servers at the edge and harnessing the distributed vast computational power present at edge devices, such as smartphones, laptops, and tablets. [4, 34]. However, the EEC environment is highly dynamic; edge devices are heterogeneous, resource-constraint, and mobile in nature. Therefore, it is necessary to employ a workload orchestrator that can operate in such an environment with a high level of dynamicity [35].

In real-world scenarios, various IoT applications need to run composite tasks consisting of multiple sub-tasks that may depend on each other. These sub-tasks are required to be executed in a specific sequence within a minimum span of time [26]. For composite tasks offloading into the MEC, task partitioning is an approach to reduce latency when a given computational task can be partitioned into multiple sub-tasks and assigned to different resources [12]. In addition, some of the sub-tasks can be executed in parallel, which helps reduce the task's total latency and completion time.

A composite task consists of sub-tasks with inter-dependencies, often represented as **directed acyclic graphs (DAGs)**. Researchers typically use graph theory-based partitioning approaches to divide an application sub-tasks into different clusters. Considering the heterogeneity of devices, the main aim of proposing partitioning models is to reduce completion time [3, 12, 13, 24]. Decreasing energy consumption is also considered by References [42, 44]. The author in Reference [13] proposed a partitioning model to minimize the cost of mobile devices, which includes computation delay, energy consumption, and the price paid to the server.

Previous research has made significant progress in dependent sub-task offloading in edge computing. However, most focus has been on offloading tasks to high-volume stationary servers such as MEC servers, with less emphasis on scenarios involving edge devices' computing resources in EEC. In EEC scenarios, devices at the edge of the network with no or restricted computational

capabilities struggle to complete delay-sensitive large tasks. However, edge devices used as computing resources are more resource-constrained and heterogeneous compared to MEC servers. In addition, task generators and resource devices can be mobile in the EEC environments, which increases the risk of task failure. Offloading composite tasks with varying dependencies on various mobile devices even increases failure risk. As for the dependent sub-task offloading process, we have to deal with more than two mobile devices communicating with each other over varying time intervals. In Reference [30], the authors proposed an optimization model for dependent sub-tasks offloading where mobile edge devices are involved as a helper or computing resource. They use historical data to predict the mobile devices' destination and offload the sub-tasks only on devices that are located in the offloading range. However, they assume all edge devices are homogeneous in terms of computing resources and also consider that all mobile devices are moving at the same speed. This is not the case in the real world, where mobile devices are heterogeneous and can move at various speeds and directions.

In this article, we investigate the offloading of composite tasks in the EEC environment consisting of various sub-tasks that may be dependent on each other and need to be executed in a specific order. A novel partitioned-based **Deep Reinforcement Learning (DRL)** approach to mobility-aware **Dependent sub-Tasks Orchestration (DeTOrch)** is proposed, which aims to minimize the composite task completion time through decreasing the sub-tasks waiting time while utilizing the computing capacity of mobile edge devices as the primary resources and minimizing the usage of high-capacity servers. This model proposes a heuristic trajectory prediction algorithm using basic location data to predict the likelihood of device failure caused by device mobility. We conducted extensive simulations, and the results show that the proposed DeTOrch scheme performs highly reliably in an EEC environment with a high degree of device mobility. Moreover, it is efficient in terms of devices and network energy usage and task completion time.

The rest of this article is organized as follows: Related work is summarized in Section 2. The system model is represented in Section 4, and the problem modeling is described in Section 5. Mobility-aware DeTOrch is presented in Section 6. The proposed work is evaluated in Section 7. The article concludes by giving an outline of the future directions of this work in Section 8.

2 Related Work

Sustainability has become an important consideration in the design and operation of computing systems, especially as the demand for energy-efficient and environmentally responsible solutions continues to rise [11]. Researchers across various domains actively explore and propose innovative strategies to enhance sustainability, including energy-aware computing architectures [22], resource-efficient algorithms [8, 33], green energy production [27, 41], green data centers [1], and implementing edge computing solutions [17]. In edge computing environments with limited resources and critical energy concerns, sustainable practices are essential for long-term viability. Considering sustainability in task offloading and orchestration solutions enhances performance while minimizing environmental impact, promoting energy-efficient computing [36].

There have been a large number of methods for offloading dependent sub-tasks in edge computing environments with varying criteria and assumptions. In Reference [2], the authors proposed a model for scheduling delay-sensitive and computationally intensive inter-dependent tasks for parallel or sequential offloading to multiple MEC servers. Their approach aims to optimize both latency and offloading failure probability. They developed heuristic solutions based on conflict graph models and genetic algorithms. An improved genetic algorithm-based dispersed scheduling algorithm has been introduced by Reference [25] to minimize the average task computing latency while ensuring transmission reliability. To design this algorithm, various environment characteristics, such as differences in computing power, communication modes, task computation latency, queu-

ing latency, transmission latency, and random generation of computational tasks, are considered. A situation-aware orchestration of resource allocation and task scheduling to achieve collaborative rendering in a distributed IoT system is proposed by Reference [7]. This work optimally aligns resource capacity and task demands by adapting to dynamic system conditions and acknowledging the interplay between diverse tasks and heterogeneous resources. It uses a genetic algorithm to solve the problem of redundant task scheduling. The authors in Reference [39] have proposed a model that tackles the problem of offloading dependent sub-tasks in a Mobile Edge Computing system. Their model aims to minimize three objectives simultaneously: the application completion time, the energy consumption of the mobile device, and edge computing fee. This is formulated as a **multi-objective Markov Decision Process (MOMDP)** that uses a vector-valued reward, where each reward component corresponds to an objective. Additionally, they have proposed a multi-objective reinforcement learning-based schema to ensure that more important preferences are given priority during Q-network training, which helps maintain previously learned policies effectively.

Task partitioning is an approach that has been used by a number of researchers to reduce the latency of a composite task in edge computing environments, particularly in the MEC. Graph theory-based partitioning algorithm is used to group several tasks into individual clusters. Each cluster is composed of parent-child tasks and is considered as a single job [21]. Then they schedule each partition using the **Bacterial Foraging Optimization Algorithm (BFOA)** to find the schedules for the partitioned workflows and attempt to find the near-optimal schedule with minimum schedule length. The **Mobile Computation Offloading Partitioning (MCOP)** algorithm, described in Reference [44], is designed to enable dynamic application partitioning and computation offloading in mobile environments. MCOP leverages weighted call graphs to optimize the application partitioning process, considering various cost models and tradeoffs between saving time/energy and reducing transmission costs/delay. The algorithm selects which parts of the application are to be offloaded to the cloud/edge server and which parts are to be executed locally to minimize energy consumption or reduce execution time. The authors in Reference [12] propose a joint optimization approach for task partitioning and user association in MEC systems to reduce user latency. They used a mixed integer programming approach to solve the problem, which can be divided into two cases: independent and dependent tasks. They derived optimal solutions for task partitioning in both cases and proposed a user association scheme that achieves near-optimal results using dual decomposition. The authors of Reference [23] proposed a real-time heuristic algorithm based on dynamic programming that makes joint decisions for task partitioning and offloading, as well as system and application-level adaptation. Reference [13] proposed an approach for partitioning and offloading DNN tasks in multi-user MEC networks. The authors use layer-level computation partitioning to break down the DNN tasks for local or server computation. The approach aimed to minimize each MD's cost by formulating the optimal DNN-task partitioning and offloading strategy. They designed two distributed algorithms based on aggregative game theory to achieve this optimal strategy.

The authors in Reference [3] introduced **Dirichlet deep deterministic policy gradient (D3PG)**, a DRL approach derived from DDPG. The model flexibly partitions tasks, offloads subtasks to edge servers, and adjusts computational frequencies to maximize task completion, minimize energy consumption, and reduce total time. Reference [24] proposed EDGEVISION, a system framework for partitioning and orchestrating computer vision applications on heterogeneous edge computing platforms that consider both CPUs and GPUs. The EDGEVISION abstracts the heterogeneous hardware resources and task run-time environments and divides the application into individual tasks to be orchestrated and deployed into the heterogeneous edge nodes.

They also propose two scheduling algorithms to minimize the processing latency and the overall system cost.

Most of these studies have focused primarily on the heterogeneity of devices and tasks, with only a few taking into account the mobility of devices. An edge device selection and task offloading method considering mobility in a wireless **Distributed Edge Computing (DEC)** environment is proposed by Reference [19]. They formulate the problem of minimizing the response time for task offloading of mobile devices. While they considered the mobility of devices, they only considered tasks that were independent. In Reference [26], the authors proposed a scheme for offloading tasks in mobile applications with task-dependency requirements in multi-slot MEC systems. They aimed to minimize task failure by considering various delay constraints, different dependencies, online arrival patterns, and execution order of all tasks. To achieve this, they developed a migration-enabled multi-priority task sequencing algorithm, which determines the optimal task execution order by deducing five necessary theorems. Additionally, they developed a DDPG-based learning algorithm to adapt to online environments. They considered the mobile devices that offload dependent tasks on stationary MEC servers.

The **Energy-Efficient Task Offloading (EETO)** algorithm is developed by Reference [30] for dependent sub-tasks in a cooperative edge computing environment. In this setting, mobile end devices contribute to computation and communication relay support within a sliced-edge network. EETO accommodates arbitrary task dependencies characterized by a general task dependency graph and employs deep learning for trajectory prediction using users' historical data. The study formulates the task offloading optimization problem as a **quadratically constrained quadratic programming (QCQP)** problem, utilizing a solution strategy involving semidefinite relaxation and stochastic mapping to obtain task offloading decisions. Although they took into account the mobility of both the task generator and the resource devices, they did not consider the heterogeneity of devices and the different speeds of mobile devices.

Many studies have explored the offloading of dependent sub-tasks relying on heuristic or metaheuristic algorithms, such as those presented in References [2, 7, 23, 25], as the problem is known to be NP-hard. However, these algorithms often lack the ability to fully adapt to dynamic edge computing scenarios due to their high complexity and time-consuming nature [39]. To address this issue, some researchers have employed Reinforcement Learning techniques, as seen in studies such as References [3, 15, 39]. A few studies also consider the mobility of devices, such as References [19, 26, 30, 37]. However, none of these models take into account the orchestration of dependent subtasks in the EEC environment, particularly when both the task generator and resources are mobile and can move at varying speeds and directions. This study aims to propose a solution for dependent task orchestration that maximizes the use of mobile edge devices as a computational resource and minimizes composite task completion time. The following section explains the motivation scenario of this article.

3 Motivation Scenario

As a motivation scenario, consider a community in a remote developing region. The region suffers from water shortages, while agriculture is the main source of income. In addition, this area has hard water, which has damaged the pipe system and further complicates access to clean water. To address these challenges, local authorities have proposed implementing sustainable smart irrigation [10] and water softener systems [5]. These systems aim to optimize water use while preventing further pipe scaling and environmental harm.

Setting up these systems requires IoT sensors to monitor water quality, soil moisture, and local demand, in addition to computational resources capable of running **machine learning (ML)** applications for data-driven optimization. Since this community is far from any developed area,

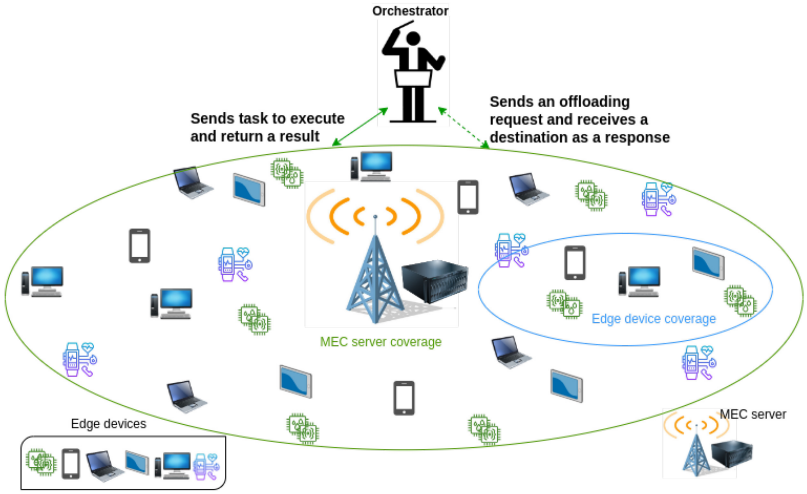


Fig. 1. Orchestration area.

it only has a nearby MEC server to offload the computing task and data. However, this is already mostly occupied with other applications, and there is no budget for setting up a new high-capacity server. Therefore, using available computing resources at edge devices—such as personal computers, laptops, and smartphones—would be the only reasonable option.

Water softening and irrigation scheduling require precise timing and resource efficiency. This can be achieved by employing feedback control systems that dynamically adjust based on real-time data from sensors. Although these applications are designed based on sustainability criteria to use less energy for computing, as ML-based applications, they still need to process vast amounts of data in real-time. Their tasks often exceed the capabilities of individual edge devices. As a result, the processing tasks need to be broken down into smaller partitions and distributed across multiple devices. However, edge devices are heterogeneous and have limited resources. They may be carried by community members who are moving by walking or micro-mobility vehicles such as bicycles.

In such a scenario, a workload orchestration system is needed to find the optimum device for each offloaded task. The goal is to enhance the use of edge device computing resources and reduce the reliance on MEC servers, all while ensuring that the processing results meet the predefined **Service Level Agreements (SLAs)** and **Quality of Service (QoS)** requirements.

4 System Model

This section provides a detailed description of how edge devices interconnect with each other in an EEC environment considered in this article. It also examines the conceptual modeling of the diverse resources used by these edge devices.

4.1 Device Model

Consider an EEC environment comprising a MEC server and a variety of heterogeneous edge devices (i.e., laptops, smartphones, IoT sensors, and wearable). The total number of edge devices, d_j with computing capability, is denoted as D. Consequently, there exist $D+1$ computing resources denoted as $\mathcal{D}_{\mathcal{R}} = \{d_j | j \in \{1, 2, \ldots, D+1\}\}$. All devices in the area are connected to a network with a high computing-powered orchestrator, $O_{\mathcal{T}}$, placed on device d_o, as shown in Figure 1. We assume that the communication range of the MEC server covers the entire area, while edge devices can only communicate with other devices within a limited range. Each edge device is characterized

by a set of parameters: $\boldsymbol{d}_j = [d_j^{Id}, d_j^{Comp}, d^{Cu}, d_j^{Nc}, d_j^{B}, d_j^{D\lambda}, d_j^{S\lambda}, d_j^{Mo}, d_j^{Qu}, d_j^{V}, d_j^{X}, d_j^{Y}]$. Here, d_j^{Id} is the identifier of a device d_j, and d_j^{Comp} refers to the CPU power per core of the device, measured in **million instructions per second (MIPS)**. d^{Cu} refers to current CPU utilization of device. d_j^{Nc} represents the number of CPU cores. d_j^{B} indicates the remaining battery level, set at 100 for non-battery-based devices. $d_j^{D\lambda}$ denotes the dynamic energy consumption per second during the computing process, and $d_j^{S\lambda}$ refers to the static energy required to stay connected to the network and remain powered on. Additionally, d_j^{Mo} is a binary value representing the device's mobility state, and d_j^{Qu} is the number of tasks waiting for execution at the device.

Furthermore, we assume that each device $\boldsymbol{d}_j$ is allocated equal bandwidth of $\mathcal{W}$ (measured in Hz) on the network to communicate with the orchestrator $O_{\mathcal{T}}$. Hence, the edge devices can send requests and receive responses with equal network priorities.

4.2 Composite Task Model

A task $\boldsymbol{T}_i$ generated by any edge device is a task consisting of K sub-tasks, denoted as $\boldsymbol{T}_i = \{\boldsymbol{\zeta}_{i,k} | k \in \{1, 2, \ldots, K\}, i \in \{1, 2, \ldots, I\}\}$, which are required to execute in a specific order. I refers to the total number of composite tasks which is generated in the system. Each sub-task is characterized by an array $\boldsymbol{\zeta}_{i,k} = [\zeta_{i,k}^{Id}, \zeta_{i,k}^{Rq}, \zeta_{i,k}^{Siz}, \zeta_{i,k}^{Lat}, \zeta_{i,k}^{Clat}, \zeta_{i,k}^{Sd}, \zeta_{i,k}^{Rd}]$. Here, $\zeta_{i,k}^{Id}$ denotes the identifier of the sub-task, $\boldsymbol{\zeta}_{i,k}$, and $\zeta_{i,k}^{Rq}$ represents the set of prerequisite sub-tasks that must complete execution (including compilation and result generation) before $\zeta_{i,k}$ can begin processing. $\zeta_{i,k}^{Siz}$ is the size of the sub-task. $\zeta_{i,k}^{Lat}$ represents the overall latency that the sub-task $\boldsymbol{\zeta}_{i,k}$ can tolerate. Additionally, $\zeta_{i,k}^{Clat}$ refers to the computing latency the sub-task $\boldsymbol{\zeta}_{i,k}$ should meet during execution on the device. $\zeta_{i,k}^{Sd}$ and $\zeta_{i,k}^{Rd}$ denote the total size of the offloading request and result data transmitted over the network, respectively.

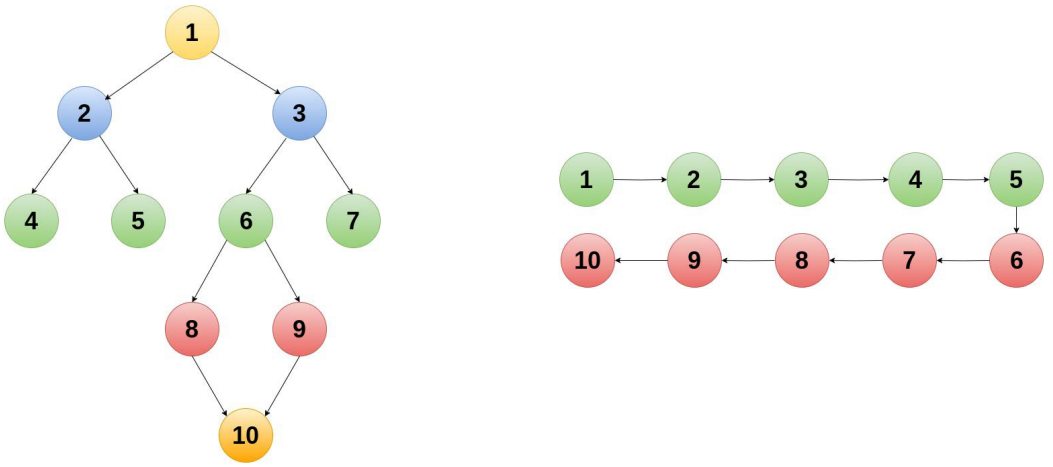
4.2.1 Tasks Dependency Model. The task dependency in this study is presented as a DAG, $G = (T, E)$, where T represents sub-tasks set as vertices, and E represents dependency among the sub-tasks as edges, $\zeta_{i,k}^{Rq} \rightarrow \zeta_{i,k}^{Id} \in E$ [20], as shown in Figure 2.

Composite tasks in this article are defined with two models of dependencies among the sub-tasks, namely, *sequential* and *general*. In sequential dependency, each sub-task depends on the completion of only one previous sub-task and general dependency in which a sub-task may depend on the completion of one or more of the previous sub-tasks [2] (see Figure 2 for illustration.)

4.2.2 Composite Task Partitioning Model. We consider composite task $\boldsymbol{T}_i$, consisting of K sub-tasks labeled from 1 to K, which can have sequential or general dependencies between them. Using the algorithm is explained in Section 6.1, each composite task T_i is divided into N partitions $P_i = \{p_{i,n} | n \in \{1, 2, \ldots, N\}, i \in \{1, 2, \ldots, I\}\}$ where $N \leq K$. Each partition is represented as $\boldsymbol{p}_{i,n} = [p_{i,n}^{Id}, p_{i,n}^{Ds}]$, which consist of m number of sub-tasks when $(1 \leq m \leq K)$. $p_{i,n}^{Id}$ is the identifier of the partition and $p_{i,n}^{Ds}$ is the dependency score that is calculated by counting all sub-tasks in other partitions that depend on sub-tasks in partition $p_{i,n}$.

4.3 Computing Model

Suppose the orchestrator $O_{\mathcal{T}}$ selects device $\boldsymbol{d}_j$ as an offloading destination for sub-tasks of partition $\boldsymbol{p}_{i,n}$. Each sub-task, $\boldsymbol{\zeta}_{i,k}$, is required to wait for a specific time interval $\tau_{i,k,j}^{Wait}$. This interval is the sum of two components: the duration for which the sub-task has to wait until the previous related tasks are completed and the time the task has to spend in the queue on the destination device before its execution.



(a) Task with general dependency.

(b) Task with sequential dependency.

Fig. 2. Composite tasks.

After waiting for $\tau_{i,k,j}^{Wait}$, the sub-task $\boldsymbol{\zeta}_{i,k}$ is executed. The execution time of sub-task $\boldsymbol{\zeta}_{i,k}$ running on the selected device $\boldsymbol{d}_j$ is calculated as:

$$\tau_{i,k,j}^{Exec} = \frac{\zeta_{i,k}^{Siz}}{d_j^{Comp}}. \tag{1}$$

The total task processing time $\tau_{i,k,j}^{Proc}$ for each sub-task $\boldsymbol{\zeta}_{i,k}$ from when an offloading request is submitted to the orchestrator till the result is returned is:

$$\tau_{i,k,j}^{Proc} = \tau_{i,n,j}^{Net} + \tau_{i,n,j}^{Srv} + \tau_{i,k,j}^{Wait} + \tau_{i,k,j}^{Exec}, \tag{2}$$

where $\tau_{i,n,j}^{Net}$ represents the time for sending a request and receiving a response over the network (network time) for each $\boldsymbol{p}_{i,n}$. $\tau_{i,n,j}^{Srv}$ denotes the orchestrator service time, which is the time taken by the orchestrator to find a destination device, $\boldsymbol{d}_j$, for partition, $\boldsymbol{p}_{i,n}$, which contains $\boldsymbol{\zeta}_{i,k}$.

As a QoS criterion, this study assumed that each sub-task, $\boldsymbol{\zeta}_{i,k}$, has a designated computing latency, $\zeta_{i,k}^{Clat}$. This represents the maximum latency that the sub-task can tolerate during execution. To meet this requirement, a sub-task with task size, $\zeta_{i,k}^{Siz}$, should be offloaded to a device with computing power, d_j^{Comp}, that is capable of executing it within a time, $\tau_{i,k,j}^{Exec}$, equal to or less than $\zeta_{i,k}^{Clat}$. To impose this constraint, we introduce a binary variable $\delta_{i,k,j}$ to represent the resource capability score, such that if $\boldsymbol{\zeta}_{i,k}$ is assigned to $\boldsymbol{d}_j$ that meets the above criterion, then $\delta_{i,k,j}$ is set to 1; otherwise, it is set to 0. Mathematically, we have

$$\delta_{i,k,j} = \begin{cases} +1 & \text{if } \tau_{i,k,j}^{Exec} \leq \zeta_{i,k}^{Clat} \\ 0 & \text{otherwise.} \end{cases} \tag{3}$$

Therefore, the Orchestrator, $O_{\mathcal{T}}$, should select a device, $\boldsymbol{d}_j$, for partition, $\boldsymbol{p}_{i,n}$, that satisfies the QoS requirements of all its sub-tasks. To consider the sub-task, $\boldsymbol{\zeta}_{i,k}$, as a success, the total processing time $\tau_{i,k,j}^{Proc}$ must be equal to or less than the total designed latency, $\zeta_{i,k}^{Lat}$, as shown on Equation

(4) if whether a sub-task has been completed successfully or not.

$$\Phi_{i,k,j} = \begin{cases} +1 & \text{if } \tau^{Proc}_{i,k,j} \leq \zeta^{Lat}_{i,k} \\ 0 & \text{otherwise.} \end{cases} \tag{4}$$

Furthermore, we use an indicator $\theta_{i,k,j}$ to specify whether a sub-task is offloaded to an edge device or the MEC server, as shown below:

$$\theta_{i,k,j} = \begin{cases} +1 & \text{if } \boldsymbol{d}_j \text{ is an Edge device} \\ 0 & \text{if } \boldsymbol{d}_j \text{ is an MEC server.} \end{cases} \tag{5}$$

Now, consider the composite task, $\boldsymbol{T}_i$, as a whole. Its execution is considered to be successful only if all of its sub-tasks are completed successfully and the completion time of the composite task is less than the maximum latency that is predefined for this task. Mathematically, we use Φ_i, as shown below, to indicate whether a composite task is successfully completed or not.

$$\Phi_i = \begin{cases} +1 & \text{if } \Phi_{i,k,j} = 1, \forall \zeta_{i,k} \in \boldsymbol{T}_i \\ 0 & \text{otherwise.} \end{cases} \tag{6}$$

Therefore,

$$\begin{aligned} \tau^{Proc}_i &= \sum_{k=1}^{K} \sum_{j=1}^{D+1} \tau^{Proc}_{i,k,j}, \\ \tau^{Proc} i,k,j &= \begin{cases} \tau^{Proc} i,k,j, & \text{if } \boldsymbol{\zeta} i,k \text{ is offloaded to device } d_j, \\ 0, & \text{otherwise.} \end{cases} \end{aligned} \tag{7}$$

is the total execution time of the composite task $\boldsymbol{T}_i$.

4.4 Computational Energy Model

The energy model of this article is designed based on the model represented in Reference [18]. The transmission energy, $\lambda^{Trn}_{i,k,j}$, for sending an offloading request for task $\boldsymbol{\zeta}_{i,k}$ and receiving a response from $\boldsymbol{d}_j$ over the network is estimated as:

$$\lambda^{Trn}_{i,k,j} = \lambda_{tb} t^{Net}_{i,k,j} (\zeta^{Sd}_{i,k} + \zeta^{Rd}_{i,k}). \tag{8}$$

Here, λ_{tb} is transmission energy per bit (Watt) and $(\zeta^{Sd}_{i,k} + \zeta^{Rd}_{i,k})$ denotes the total size (bit) of the offloading request and result data transmitted over the network. The computing energy $\lambda^{Exec}_{i,k,j}$ consumed by the selected edge device $\boldsymbol{d}_j$ with computing power d^{Comp}_j to execute task $\boldsymbol{\zeta}_{i,k}$ with size $\zeta^{Siz}_{i,k}$ is calculated by:

$$\lambda^{Exec}_{i,k,j} = (d^{D\lambda}_j + d^{S\lambda}_j) t^{Exec}_{i,k,j}. \tag{9}$$

The $d^{D\lambda}_j$, as mentioned before, is dynamic energy that the device $\boldsymbol{d}_j$ consumes each second during the computing process. The $d^{S\lambda}_j$ refers to the static energy that a device needs to stay connected to the network and remain powered on. The $\tau^{Exec}_{i,k,j}$ is computing time that is calculated by Equation (1).

The energy required by the device d_o, which is the host of orchestrator $O_{\mathcal{T}}$ to select the optimal $\boldsymbol{d}_j$, is calculated by:

$$\lambda^{O}_{i,k,j} = (d^{D\lambda}_o + d^{S\lambda}_o) \tau^{Srv}_{i,k,j}. \tag{10}$$

The $d^{D\lambda}_o$ is dynamic energy that the orchestrator device d_o consumes each second during the computing process. The $d^{S\lambda}_o$ refers to the static energy that the orchestrator device needs to stay

connected to the network and remain powered on. Therefore, the total energy required to complete the task offloading cycle from an edge device through $O_{\mathcal{T}}$ to the selected $\boldsymbol{d}_j$ is given as:

$$E^{tot}_{i,k,j} = \lambda^{Trn}_{i,k,j} + \lambda^{Exec}_{i,k,j} + \lambda^{O}_{i,k,j}. \tag{11}$$

4.5 Mobility Model

In the EEC environment, both task generators and resources can be mobile. Mobile devices move in various directions and speeds within an area and can enter or leave the area. In this article, we developed our model for pedestrian environments, where devices are carried by pedestrians or individuals riding micro-mobility vehicles such as bicycles or scooters. The range of connection for edge devices is more restricted than edge MEC servers, as depicted in Figure 1. Devices can only communicate with devices that are located in their connection range. Such mobility behavior in the environment leads to an increasing risk of failure due to potential disconnections between devices when they move outside of the covered range.

Trajectory prediction is a well-known solution for mobility management in dynamic environments. In this article, we employ the mobility trajectory prediction approach proposed in Reference [38].

Consider a mobile device that is moving at a speed of d^V_j and at an angle of d^α_j. At timestamp t, the location of the mobile device is denoted as $(d^X_j(t), d^Y_j(t))$. In this work, we assume that the mobile devices move at a constant speed in a particular direction. Therefore, we can calculate the possible location of the mobile device at timestamp $t + \Delta t$ using the following equation [40]:

$$\begin{aligned} d^X_j(t + \Delta t) &= (d^V_j \Delta t) \cos(d^\alpha_j) \\ d^Y_j(t + \Delta t) &= (d^V_j \Delta t) \sin(d^\alpha_j). \end{aligned} \tag{12}$$

Since mobile devices might pause or change direction during the Δt time period, $(d^X_j(t+\Delta t), d^Y_j(t+\Delta t))$ can be considered as the next possible location. Using the calculated next possible locations of both the source and destination devices, their possible distance can be calculated using the Euclidean distance equation [40]. Note that this is calculated according to the time that the data should be exchanged between these devices.

5 Problem Formulation

This article aims to achieve the following three objectives simultaneously:

$$\begin{aligned} Obj_1 &: \max \sum_{i=1}^{I} \max_{\boldsymbol{\zeta}_{i,k} \in \boldsymbol{T}_i} \zeta^{Lat}_{i,k} - \tau^{Proc}_i, \forall \boldsymbol{\zeta}_{i,k} \in \boldsymbol{T}_i \\ Obj_2 &: \max \sum_{i=1}^{I} \theta_{i,k,j} \forall \boldsymbol{\zeta}_{i,k} \in \boldsymbol{T}_i, \forall j \in [1, D+1] \\ Obj_3 &: \max \sum_{i=1}^{I} \Phi_i \end{aligned} \quad . \tag{13}$$

Obj_1 aims to minimize the total completion time for all composited tasks generated in the system. This can be achieved by maximizing the time saved by completing the composite tasks earlier than the maximum latency defined for their sub-tasks.

Obj_2 aims to maximize the utilization of the edge devices' resources, thereby reserving the MEC server for larger and more critical tasks. Obj_3 refers to maximizing the number of composite tasks completed successfully. This objective can conflict with Obj_1 and Obj_2 when tasks are assigned

to edge devices with limited computing resources, which can increase the execution time and the probability of task failure.

All aforementioned objectives can be consolidated into a single multi-objective problem, defined as follows:

$$\begin{aligned} &\max \sum_{i=1}^{I} (\max_{\zeta_{i,k} \in T_i} \zeta_{i,k}^{Lat} - \tau_i^{Proc}) + \theta_{i,k,j} + \Phi_i), \\ &\forall \boldsymbol{\zeta}_{i,k} \in \boldsymbol{T}_i, \forall j \in [1, D+1] \\ &s.t.: \\ &\quad C_1 : \delta_{i,k,j} = 1, \forall \zeta_{i,k} \in \boldsymbol{T}_i, \forall i \in [1, I], \forall j \in [1, D+1] \\ &\quad C_2 : \tau_i^{Proc} \leq \max_{\zeta_{i,k} \in T_i} \zeta_{i,k}^{Lat}, \forall \zeta_{i,k} \in \boldsymbol{T}_i \end{aligned} \tag{14}$$

where constraint C_1 guarantees that sub-tasks are assigned to devices with adequate resources to execute them before predefined latency. C_2 indicates that the total processing time for all sub-tasks must be less than the maximum predefined latency of the composite task. The problem in Equation (14) is a non-linear mixed integer problem that needs to be solved to enhance the success rate of composite tasks while utilizing heterogeneous edge devices. In our proposed model, we employ deep reinforcement learning to find an optimal policy to achieve all the above objectives when orchestrating tasks. The main notations used in this article are summarized in Table 1.

6 Mobility-aware Deep RL-based Dependent sub-tasks Orchestration Model

The mobility-aware Deep RL-based Dependent sub-Task Orchestration (DeTOrch) model introduces a central orchestrator that knows the current state of all the available resources, such as edge devices and a MEC server. This orchestrator is accessible to all edge devices and MEC server in an orchestration area. DeTOrch employs a state-of-the-art partitioner that breaks down a composite task into different partitions containing various numbers of sub-tasks. The DRL agent then detects the optimum destination for each partition to minimize the task completion time and waiting time of sub-tasks. Moreover, through trajectory prediction, the mobility-aware DeTOrch identifies and eliminates resources that might increase the risk of task failure due to their mobility, prior to determining the optimal destination for offloading a partition.

6.1 Partitioner

For partitioning of the dependent tasks, we adopt the method presented in Reference [21]. Specifically, Kaur et al. leverage **betweenness centrality (BC)**, a graph theory concept that measures node importance for partitioning a composite task graph. It hinges on finding the shortest paths between pairs of nodes, with the aim of minimizing edge weights in directed graphs. Each vertex's BC represents the sum of these shortest paths passing through it. Each edge's BC quantifies the number of shortest paths they participate in. The partitioning process involves determining the BC for each edge and removing the one with the highest BC index. To compute BC, Kaur et al. utilize Brandes' efficient algorithm [6] that calculates the BC for an edge (b, e) from vertex $b \in T$ to vertex $e \in T$, denoted as $\beta_{b,e}$, in the following two steps:

(1) Let all predecessors of vertex e be $L \subset T$. The path lengths for all predecessors of vertex b to vertex b and for all predecessors of vertex e to vertex e are first computed using the following equation, which is referred to as the centrality index, c_k:

$$c_k = \frac{l_{k,b}}{l_{k,e}}(1 + \sigma_e). \tag{15}$$

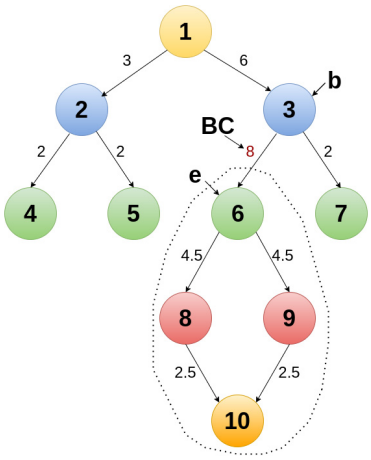
Table 1. Main Symbols Used in This Article

Attributes	Description
$\boldsymbol{T}_i$	Composite task i
τ_i^{Proc}	Total processing time of a composite task i
$\zeta_{i,k}^{Id}$	The ID of the sub-task k of task i
$\zeta_{i,k}^{Siz}$	The size of the sub-task k of task i
$\zeta_{i,k}^{Sd}$	The size of offloading request data
$\zeta_{i,k}^{Rd}$	The size of result data
$\zeta_{i,k}^{Clat}$	The computing latency for the sub-task
$\zeta_{i,k}^{Lat}$	The maximum overall latency for the sub-task
$\zeta_{i,k}^{Mo}$	Mobility state of the device that requested the task
d_j^{Id}	Device ID
d_j^{Comp}	The total MIPS available per core in the device
d_j^{Nc}	The number of CPU cores available in the device
d_j^{Cu}	Current CPU utilization of the device
d_j^{B}	Remaining battery (for battery-based devices)
$d_j^{D\lambda}$	The dynamic energy that device consumes each second for computing (W)
$d_j^{S\lambda}$	The static energy that device consumes to stay connected to the network and remain powered on (W)
d_j^{Mo}	The mobility state of the device
d_j^{Qu}	Number of tasks waiting for execution at the device
d_j^{Off}	A binary value that shows whether the device is in offloading coverage of the task generator device
d_j^{X}	The length value of the current location of the device in the area
d_j^{Y}	The width value of the current location of the device in the area
d_j^{V}	Speed of device
d_j^{α}	The angle that device is moving at timestamp t
$\delta_{i,n,j}$	Resource compatibility score
$p_{i,n}^{Id}$	The ID of the partition n of task i
$p_{i,n}^{Ds}$	Dependency score of the partition
$\mu_{i,n,j}^{Pre}$	Mobility failure possibility score for d_j and previous partition device
$\mu_{i,n,j}^{Nxt}$	Mobility failure possibility score for d_j and next partition device
$\tau_{i,k,j}^{Srv}$	Orchestrator service time
$\tau_{i,k,j}^{Net}$	Network time
$\tau_{i,k,j}^{Wait}$	Waiting time
$\tau_{i,k,j}^{Exec}$	Execution time
$\tau_{i,k,j}^{Proc}$	Total task processing time
$\tau_{i,k,j}^{D}$	Delay time
$\lambda_{i,k,j}^{Trn}$	Total energy is consumed for data transmission over the network (W)
$\lambda_{i,k,j}^{Comp}$	Total energy that device consumes for computing a task (W)
λ^{tb}	The energy is consumed for each bit of data (W) transmission over the network
$\lambda_{i,k,j}^{O}$	Total energy that orchestrator device consumes for selecting a device (W)
$\mathcal{P}_{mec}$	Offloading to a MEC server penalty
$\mathcal{P}_{wrs}$	Wrong resource assignment penalty
θ	Indicator to show a sub-task has been offloaded to an edge device or MEC server
Φ	Task success indicator
δ	Resource capability score
n_{alt}	Number of alternative devices to offload a task
r	Result reward

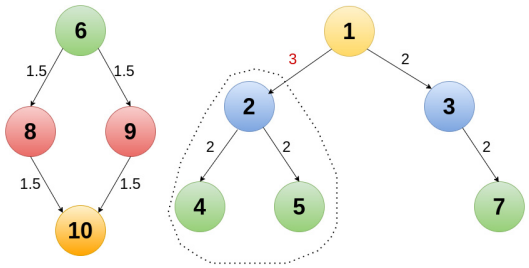
Here, $l_{k,b}$ represents the number of shortest paths from vertex k to vertex b. σ_e is the number of vertices that depend on vertex e.

(2) The BC of the edge (b, e) is then obtained by summing up all centrality indices:

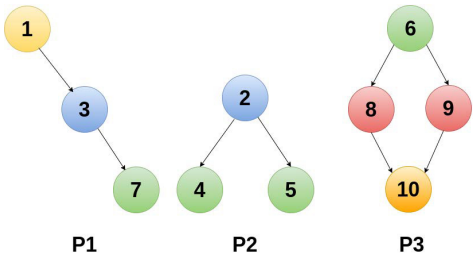
$$\beta_{b,e} = \sum_{k \in L} c_k. \tag{16}$$



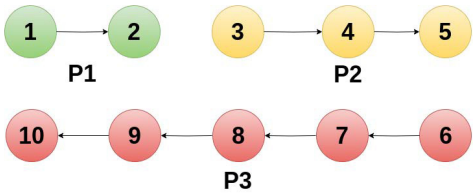
(a) Betweenness index for a task with general dependency.



(b) Two partitions after first edge removal iteration of a task with general dependency.



(c) Three partitions after second edge removal iteration of the composite task with general dependency.



(d) Three partitions after second edge removal iteration of a task with sequential dependency.

Fig. 3. Composite tasks partitioning steps.

Take the BC of edge (3, 6) of a composite task shown in Figure 3(a) as an example. The set of predecessors of vertex $e = 6$ is $L = \{1, 3\}$. For each predecessor, we can find its centrality index. Specifically, $c_1 = 4$ and $c_3 = 4$, with $l_{1,3} = 1$, $l_{3,3} = 1$, $l_{3,3} = 1$, $l_{3,6} = 1$, and $\sigma_6 = 3$. Note that vertices 8, 9, and 10 depend on vertex 6. Hence, we have $\beta_{3,6} = 8$. In Figure 3(b), the BC of each edge is marked on the edge. To partition a composite task, edges with the highest BC are removed following the steps below:

(1) Calculate the BC of all edges E in the composite task.
(2) Remove the edge that has the highest BC.
(3) Calculate the BC again for the two sub-graphs that are created as a result of the edge removal.
(4) Repeat steps 2 and 3 until the required edges are removed based on the predefined criteria.

For instance, after removing the edge between node 3 and 6, which has the highest BC, the composite task is split into two sub-composite tasks, as shown in Figure 3(b). After partitioning, the BC index is recalculated for the composite tasks graphs (i.e., Figure 3(b)), and the edge with the highest BC is removed to obtain the smaller partitions as shown in 3(c). Therefore, the composite tasks consist of 10 sub-tasks with general and sequential dependency shown in Figure 2; each is broken down into three partitions (after removing two edges), as illustrated in Figure 3(c) and 3(d), respectively.

ALGORITHM 1: Check mobility failure possibility

```
1: function CheckMobilityFailure(p_{i,n}, d_j, indicator)
       //The indicator determines the function is called to find the
2:         distance to the previous or the next partition.
3:     failurePossibility ← False // Possibility of failure due to
4:                                                          mobility
5:     offloadingRange ← 25 // Edge devices offloading range
6:     ρ ← 1 // noise tolerant constant
7:     if (indicator == "previous" & p_{i,n}.getType()! = 1) then
8:         givenPartition ← previousPartition
9:         givenDevice ←
10:          getOffloadingDestination(givenPartition)
11:    else
12:        if (indicator == "next" & p_{i,n}.getType() == 3) then
13:            givenPartition ← p_{i,n}
14:            givenDevice ← getDeviceSource(givenPartition)
15:    Δt ← maxLatencyOfPartition( givenPartition ) // Find the maximum deadline to execute all sub-tasks of the
   partition
16:    (X_{d1}, Y_{d1}) ← predictNextLocation(givenDevice, Δt)
17:    (X_{d2}, Y_{d2}) ← predictNextLocation(d_j, Δt)
18:    distance ←calculateDistance( (X_{d1}, Y_{d1}), (X_{d2}, Y_{d2}))
19:    if (distance >= offloadingRange + ρ) then
20:        failurePossibility ← True
21:    return failurePossibility
22:
23: function predictNextLocation(d_j, Δt)
24:    //Based on Equation (12)
25:    X ← d_j^X + (d_j^V × Δt × cos(d_j^α))
26:    Y ← d_j^Y + (d_j^V × Δt × sin(d_j^α))
27:    return (X, Y)
28:
29: CheckMobilityFailure(p_{i,n}, d_j, previous)
30: CheckMobilityFailure(p_{i,n}, d_j, next)
```

6.2 Mobility Manager

The mobility manager is responsible for mitigating the risk of task failure due to mobility in various situations. As mentioned above, in the EEC environment, both task generators and resource devices can be mobile, increasing the risk of task failure. This risk is exacerbated when dealing with composite tasks with varying dependencies.

For a composite task that is partitioned into multiple partitions, each partition is assigned to a device that can be either mobile or stationary. To avoid mobility-related failures, the orchestrator must consider two important factors: (1) the distance between the source and each resource device and (2) the distance between resource devices that need to exchange data between sub-tasks for execution. The mobility manager assesses the likelihood of failure caused by mobility for each device and upcoming partition based on these distances, as outlined in Algorithm 1.

The assessment distinguishes between three types of partitions:

— **Type 1**: For a Type 1 partition, there is no previous partition upon which it depends. It needs to receive data from the source device (the device that requested the offloading of the task) and sends output results to the host device of the next partition (the device to which the next partition will be offloaded). For example, in Figure 3(d), $P1$ is Type 1.

— **Type 2**: A Type 2 partition has at least one previous dependent partition and a next dependent partition. The device that hosts this type of partition needs to communicate with at least two devices that host previous and next partitions, other than the task source device at specific times to exchange required data. In Figure 3(d), $P2$ is considered a Type 2.
— **Type 3**: A Type 3 partition has at least one previous dependent partition and no next dependent partitions. For this type, the host needs to receive data from the previous partitions and return the output results to the task source device. In Figure 3(d), $P3$ is considered a Type 3.

To avoid task failure caused by mobility, the orchestrator must find resources for partitions of a composite task whose locations are within the connection range, $offloadingRange$, of each other whenever data transfer between the devices is required. Lines 7 to 14 aim to assign host devices according to the type of input partition. Then, on line 15, Δt, which is the maximum deadline to execute all sub-tasks of the partition, is calculated. Lines 16–18 use a trajectory predictor (see Section 6.2) to predict the distance, $distance$, between devices hosting each partition at a specific time Δt, when they need to exchange data, as shown in Equation (12).

Since the partitions arrive at the orchestrator in order (e.g., Type 1, Type 2, Type 3), predicting the distance from the next dependent partition's host device is impossible for partitions Type 1 and 2. For partition Type 3, since the results should be sent to the source device, the mobility manager predicts the distance to the source device at the time it is ready to send back results. For predicting the distance to the previous partition, since partition Type 1 has no preceding partition, the distance to the source at the offloading time is considered by the orchestrator. For a partition Type 2 or 3, the mobility manager predicts the distance to the previous partition at the time the results of the previous partition are ready and need to be sent as input to this device, as shown in lines 7 to 10. Finally, based on the predicted locations of devices, the distance between devices is calculated as shown in lines 16–18. If this distance is greater than the connection range, $offloadingRange$, plus noise tolerance constant, ρ, the $failurePossibility$ will be True; otherwise, it will be False.

6.3 Procedure of DeTOrch

The Mobility-aware DeTOrch model consists of seven steps (see Figure 4), encompassing all the necessary processes from when an offloading request is received until the result is returned.

Step 1: The edge device sends an offloading request for a composite task to the orchestrator.

Step 2: The *partitioner*, as explained in Section 6.1, decomposes the composite task into multiple partitions. Then, it sends an offloading request for each partition to the *Workload Inspector*.

Steps 3 & 4: The *Workload Inspector* retrieves the properties of all available resources and the prediction data about the next possible location of devices from the inventory. Then, it creates a state observation based on the attributes of the partition and resource devices, which is detailed in Section 6.5.1. The resulting state is then sent to the orchestrator's *Deep RL* component.

Step 4: The *Deep RL* component uses the **Deep Q Network (DQN)** algorithm based on the architecture introduced in Reference [45]. The algorithm is designed to minimize the task completion time and sub-tasks waiting time by utilizing a reward function, which will be explained in Section 6.5.3. The *Resource Selector* receives a list of scores for all actions (resource devices) generated by the DQN.

Step 5: The *Resource Selector* is responsible for deciding the final destination resource for a given partition. To do so, the sub-tasks QoS requirements of each partition, such as availability, reliability, and response time, are considered. To ensure availability and reliability, it reduces the action space by removing the potential host/destination resources that are outside the source device's connection coverage using the action masking technique, which will be described in Section 6.5.4. Then, the selected destination resource is sent back to the workload manager.

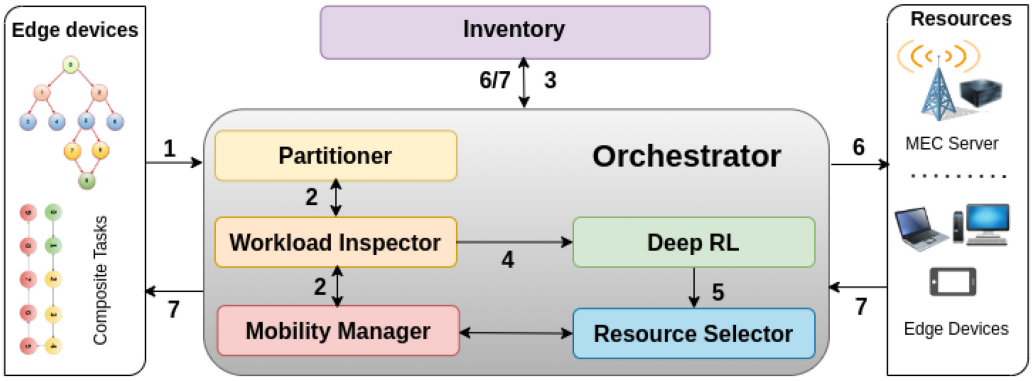


Fig. 4. Mobility-aware DeTOrch orchestrator components and composite task offloading flow.

Step 6: At this step, the partition is offloaded to the selected resource (device), and all sub-tasks are executed by the resource according to their dependency. Concurrently, the current status of the resource in the inventory is updated.

Step 7: After the completion of each sub-task execution, the result is sent back to the edge device that requested it. The status of device-available resources (i.e., memory, CPU utilization, remaining battery power, etc.) in the inventory is then updated, and the *Deep RL* component of the orchestrator receives a reward based on the outcome of the sub-task execution (i.e., success or failure).

6.4 Implementing DRL on the Edge

To solve the problem described in Equation (14), our orchestrator is designed as a DRL [45] agent. This requires modeling the problem in Equation 14 as a **Markov Decision Process (MDP)**, which can be expressed by a 5-tuple $< S, A, \Gamma, R, \gamma >$ defined as follows:

- S denotes the state space, encompassing the set of all valid states, i.e., $s \in S$, where s represents the state of the agent.
- A denotes the action space, encompassing the set of all valid actions, i.e., $a \in A$, where a represents the action.
- $\Gamma : S \times A \times S \rightarrow \Gamma(S)$ is the transition function. It describes the transition from the current state s_t to the next state s_{t+1}, given the current action a_t, and outputs the probability distribution over the next state.
- $R : S \times A \rightarrow R$ is the reward function that provides the immediate reward r for a taken action a at time t considering the state transition such that $r_t = R(s_t, a_t)$.
- $\gamma \in [0, 1]$ is the discount factor to indicate the importance of the immediate or long-term rewards. When γ is 0, only the immediate reward is considered.

The DRL agent seeks to determine the optimal deterministic policy $\pi : S \rightarrow A$ that maximizes the expected cumulative long-term rewards R_t, as defined by the following Equation (17):

$$R_t = \sum_{t=t}^{\infty} \gamma R(s_t, s_{(t)+1}). \tag{17}$$

Alternatively, the optimal policy can be found by maximizing the action-value function $Q_\pi(s_t, a_t)$ defined as follows:

$$Q_\pi(s_t, a_t) = \mathbb{E}[R_t | s_t = s, a_t = a]. \tag{18}$$

As a result, the optimal policy can be found by choosing the action with the highest Q value in each state, expressed as:

$$\pi(s_t) = \arg\max_{a'}(Q(s_t, a')). \tag{19}$$

To train the DRL agent, we apply the DQN [31].

It is worth noting that MDP relies on the Markov property, which requires that the next state depends only on the current state and not on the previous states.

In the dynamic edge computing environment, ensuring the Markov property is challenging. The state representation needs to be defined carefully to consider the changes in the environment from the current state to the next. For instance, when modeling the state in the task orchestration system, each incoming task must be represented cautiously. There is a risk that a new task with the same attributes may enter the environment simultaneously with a task that is already completed. In such a situation, no changes can be detected. To address this issue, it is essential for different tasks to be differentiable [45]. In this article, since we orchestrate a composite task in the partition level instead of a sub-task level, we add a unique partition ID, $p_{i,n}^{Id}$, for each incoming partition in the state. This ensures that the implementation of DRL satisfies the Markov property.

6.5 DeTOrch DRL Modeling

In this section, we describe the DRL modeling of the orchestrator, defining its state s, action a, and reward r as follows:

6.5.1 State. The state comprises the properties of the incoming partition that has arrived in the orchestrator and a subset of properties of all available devices, d'_j. The subset of device properties describes the current state of each available resource in the area at timestamp t, such as current CPU utilization, remaining battery of the device, device queue size, and offloadability. Additionally, to improve the algorithm's accuracy, we calculate complex items such as resource compatibility score, relative mobility score, and mobility failure possibility score—based on the primary attributes of the partition and device as described below:

- Relative mobility score (d'^{Mo}_j): It represents the relative mobility state of both task source and resource device. This value is 1 if both the task and the resource device are mobile. If one of them is mobile and the other one is stationary, then it is 0.5. If both devices are stationary, then it is 0.
- Resource compatibility score ($\delta_{i,n,j}$): A binary value that shows whether a device has enough CPU MIPS to execute the partition's largest sub-task. This criterion is assessed by the *CheckResourceMatching* function, represented in Algorithm 2. If the function output is $True$, then $\delta_{i,n,j}$ is set to 1; otherwise, this is 0. Section 6.5.3 explains in more detail how the *CheckResourceMatching* function works.
- Mobility failure possibility score (μ) is a binary value that is calculated for a device, d_j, and other relevant devices (when applicable). The device that needs to send data to d_j (i.e., task source or the previous partition host device) at a specific time is indicated as ($\mu^{Pre}_{i,n,j}$). And the device that d_j needs to send data to it (i.e., the next partition host device or task source) after finishing all sub-tasks execution is referred to as ($\mu^{Nxt}_{i,n,j}$). These values are calculated by the mobility manager, as explained in Section 6.2.

Therefore, d'_j is represented as: $d'_j = [d'^{Id}_j, d'^{Cu}_j, d'^{B}_j, d'^{Qu}_j, d'^{Off}_j, d'^{Mo}_j, \delta_{i,n,j}, \mu^{Pre}_{i,n,j}, \mu^{Nxt}_{i,n,j}]$.
State,s, represents as:

$$s = \left\{p_{i,n}, \cup_{j=0}^{D+1}\{d'_j\}\right\}. \tag{20}$$

ALGORITHM 2: Check resources matching

```
function CheckResourceMatching(p_{i,n}, d_j)
    τ^{Exec}_{i,k,j} ← 0 // Execution time
    δ_{i,k,j} ← 0 // Resource compatibility score for sub-task
    δ_{i,n,j} ← 1 //Resource compatibility score for partition
    for each ζ_{i,k} ∈ p_{i,n} do
        τ^{Exec}_{i,k,j} ← ζ^{Siz}_{i,k} / d^{Comp}_j
        if τ^{Exec}_{i,k,j} ≤ ζ^{Clat}_{i,k} then
            δ_{i,k,j} ← 1
        δ_{i,n,j} ← δ_{i,n,j} ∧ δ_{i,k,j}
    if δ_{i,n,j} == 1 then
        return True
    else
        return False
```

The transition function Γ is unknown due to the complexity of the environment.

6.5.2 Action. The orchestrator DRL actions are limited to the number of edge devices with computational resources in the environment plus 1 MEC server. Therefore, we define the action space at timestamp t to be $a_i \in \{a_1, a_2, ..., a_{D+1}\}$. The score for all actions is calculated by the neural network embedded in the orchestrator DRL process.

6.5.3 Reward. We aim to minimize the composite task completion time through decreasing the sub-tasks waiting time while increasing the utilization of edge devices at the EEC layer compared to the MEC. To accomplish this, we follow the below three principles to ensure optimum resource assignment in this environment.

- To take advantage of the computing resource of edge devices, it is imperative that partitions/sub-tasks that can be executed on the EEC layer are offloaded to the edge devices.
- To reduce the total sub-tasks waiting time, partitions/sub-tasks with higher dependency scores should be executed on more powerful devices to decrease their execution time and their dependent sub-tasks waiting time.
- To avoid failure due to latency, a partition must be offloaded to a device that allows all its sub-tasks to meet their designed execution latency.

We designed the reward function according to the above principles. Agents were rewarded for completing the sub-tasks earlier than the deadline and penalized for incorrect assignments to a small resource that was not capable to execute all sub-tasks of the partitions in predefined deadline. The orchestrator receives a reward, $r_{i,n,j}$, from the environment at time t after executing action $a_{t-\omega}$. The ω is the delayed time it takes for an action to affect the environment. This reward is based on the amount of time saved by completing a partition's sub-tasks before their predefined deadline.

The reward related to completion of a sub-task $r_{i,k,j}$ is the time that is saved by completing a sub-task earlier than the predefined execution time is calculated by sub-task's latency $\zeta^{Lat}_{i,k}$ minus sub-task processing time $\tau^{Proc}_{i,k,j}$ (see Equation (21)). The completion reward r for each partition is the sum of all its sub-tasks completion reward $r_{i,k,j}$ (see Equation (22)). The orchestrator can get this reward for the partition $p_{i,n}$ if all sub-tasks of the partition are successfully completed. Otherwise, reward $r_{i,n,j}$ will be 0.

$$r_{i,k,j} = \zeta^{Lat}_{i,k} - \tau^{Proc}_{i,k,j} \tag{21}$$

$$r_{i,n,j} = \begin{cases} 0 & \text{if a } \zeta_{i,k} \in p_{i,n} \text{ is failed} \\ (scaler(\sum_{k=1}^{K} r_{i,k,j})), \forall \zeta_{i,k} \in p_{i,n} & \text{otherwise} \end{cases} \tag{22}$$

$$scaler(x) = \left(1 - \frac{1}{1+x}\right) \tag{23}$$

To normalize the reward or penalty values between 1 and 0, a *scaler* function is defined as represented in Equation (23) .

MEC Offloading penalty: The orchestrator receives a penalty $\mathcal{P}^{mec}_{i,n,j}$ when the partition, $p_{i,n}$, is sent to the MEC, when there are alternative edge devices within their offloading coverage, which are capable of executing the sub-tasks.

$$\mathcal{P}^{mec}_{i,n,j} = \begin{cases} 0 & \text{if } n_{alt} <= 1 \\ -1 + scaler(p^{Ds}_{i,n}) & \text{otherwise.} \end{cases} \tag{24}$$

Wrong Resource Selection penalty: The orchestrator receives a penalty $\mathcal{P}^{wrs}_{i,n,j}$ when a partition is offloaded to a device that one or all of its sub-tasks cannot meet their predefined execution latency. This criterion is assessed by *CheckResourceMatching*() function that is represented in algorithm 2. If the function output is *False*, then $\mathcal{P}^{wrs}_{i,n,j}$ is set to -2; otherwise, this is 0.

This function first calculates the execution time on device $\boldsymbol{d}_j$ for each sub-task in the partition based on Equation (1) (see Algorithm 2, line 6). Then, it checks if device $\boldsymbol{d}_j$ has the required resource and can execute all sub-tasks of the partition. Finally, if resource compatibility score of the partition $\delta_{i,n,j}$ is 1, then it returns *True*; otherwise, returns *False*.

By considering the explained penalties, the $R(s, a)$ is the reward value in range $[-2, 1]$ for state s and action a that is calculated as shown in Equation (25).

$$R(s, a) = r + \mathcal{P}_{mec} + \mathcal{P}_{wrs} \tag{25}$$

6.5.4 Resource Selection. The orchestrator agent needs to select a destination device to offload the partition. It does this by using a list of scores for all potential actions (resource devices) generated by the DQN. However, not all devices may be accessible at all times. To address this issue, it employs a **pruning** process to filter out or exclude the resource devices located outside the task source device connection range. In this article, the *Resource Selector* component is responsible for detecting and removing impossible actions before selecting the final action. In this process, two types of devices are detected and pruned:

- **Non-offloadable devices:** devices that are not within the connection range of the source device at the offloading time.
- **High-risk mobile devices:** mobile devices that are anticipated by the mobility manager using *CheckMobilityFailure*() function, demonstrated in Algorithm 1 to be located out of the connection range at the time of data exchange between partitions/devices or at result return time.

After the pruning process, the action with the highest score will be selected as the final action (i.e., resource device).

7 Performance Evaluation

To evaluate the proposed model, we simulated a developing agricultural region experiencing water shortages and issues related to hard water, as explained in Section 3. To address these challenges,

we have developed ML-based, data-driven applications, including smart water softeners and irrigation systems. To facilitate these solutions, IoT sensors have been deployed throughout the area. A MEC server is established, providing coverage for the entire region. Additionally, various edge devices, such as PCs, laptops, and smartphones, are equipped with computing resources and are interconnected using a **Device-to-Device (D2D)** network. The goal is to facilitate real-time processing of data collected from IoT sensors and other devices. This involves executing tasks using the computing resources available on edge devices, thereby decreasing reliance on the MEC server. At the same time, it is essential to ensure that applications operate smoothly and meet their QoS requirements.

This section outlines the simulation settings and evaluates the proposed model's performance in comparison to existing state-of-the-art models. In designing the simulation setting, we draw insights from how other researchers have defined their settings in similar works [9, 29, 45].

7.1 General Simulation Setting

All the characteristics of a real-world environment discussed throughout the article are included in our scenarios. These characteristics are as follows:

Simulation environment: To evaluate the performance of mobility-aware DeTOrch in an extreme edge environment, we simulated our designed scenarios using PureEdgeSim[1] [29]. This simulator enables evaluating resource management strategies in the EEC environments.

Resource-constrained heterogeneous devices: We consider an EEC environment that consists of an edge server and five types of devices, including smartphones, laptops, Raspberry Pis, and two types of IoT sensors (see Table 4). In the area, around 80% of devices are task generators and only 40% of devices have computational resources.

Number of devices: To assess the model's scalability with an increasing number of devices, the number of devices ranges from 100 to 200.

Heterogeneity of tasks: We have defined four applications. Two of these applications, namely, "App A" and "App B," consist of composite tasks with a general dependency model, as shown in Figure 2(a). The other two applications, App C and App D, consist of composite tasks with sequential dependency, as depicted in Figure 2(b). In each composite task, all sub-tasks have varying computing resource requirements andSLAs. Therefore, all devices may not be capable of executing all sub-tasks, and some sub-tasks must only be offloaded to specific devices for execution. For example, in applications "App A" and "App C," the sub-tasks with ID 1 can just be executed on laptops and the edge server. The sub-tasks with IDs 3 and 5 cannot be executed on Raspberry Pi. The detailed configurations of the sub-tasks in the general and sequential composite tasks are presented in Table 2 and Table 3, respectively. For all sub-tasks, we have set their request size (ζ^{Sd}) and result size (ζ^{Rd}) to 1,500 and 50 kilobytes, respectively.

Fluctuated workload: To evaluate the model's scalability with an increasing number of tasks, two levels of workload are defined in our scenarios: "Medium" and "High." At the "Medium" level, tasks are generated at a rate of one per minute. At the "High" level, tasks are generated at a rate of three per minute.

Mobility: Within the environment, 30% of all devices are mobile. Of these mobile devices, 18% are both task generators and resources, while 12% are only task generators. These devices can easily move within and join or exit the area.

For the **energy** and **network** models, we used the default settings and values in PureEdgeSim [29], as shown in Table 4.

[1] https://github.com/CharafeddineMechali,kh/PureEdgeSim

Table 2. The Properties of Sub-tasks for Applications A and B with General Dependencies

Application	No.(ζ^{Id})	required sub-task(ζ^{Rq})	Overall Latency(ζ^{Lat})	Computing Latency(ζ^{Lat})	Size (ζ^{Siz}) *(GIPS)*
App A (Resource-intensive)	1	-	0.3	0.3	30
	2	1	1	0.65	10
	3	1	1.1	0.8	20
	4	2	1.3	0.32	5
	5	2	3	2	50
	6	3	1.5	0.15	2
	7	3	2.1	1	15
	8	6	2.5	0.65	10
	9	6	3.5	0.8	12
	10	8,9	4	0.32	5
App B	1	-	0.7	0.65	10
	2	1	1.5	0.5	7
	3	1	1.8	0.8	12
	4	2	2	0.32	5
	5	2	2.8	0.8	10
	6	3	2	0.2	2
	7	3	3.5	1	15
	8	6	4	0.7	10
	9	6	4.8	0.8	12
	10	8,9	5.5	0.32	5

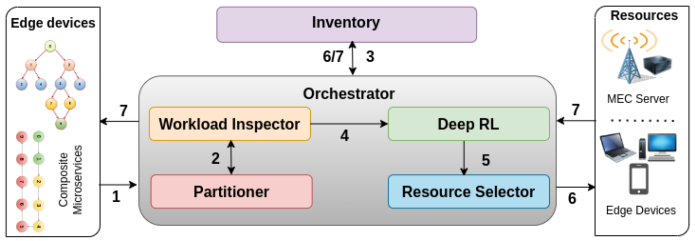


Fig. 5. DeTOrch orchestrator components and composite task offloading flow.

7.2 Benchmarks

We implemented the following algorithms as benchmarks to evaluate the effectiveness of our proposed methods, including partitioning, DRL, and mobility manager:

— **Mobility-aware DeTOrch:** This is our comprehensive model for composite-task orchestration in an environment with complex mobility behavior, as explained in Section 6.
— **DeTOrch:** It is a variant of the proposed model without the mobility manager, as illustrated in Figure 5. As a result, the *mobility failure possibility score (μ)* is not included in the DeTOrch state. Additionally, during the resource selection phase, only *non-offloadable devices* are pruned, and the *high-risk mobile devices* are not detected.
— **DeTOrch without partitioning:** It resembles an implementation of the DeTOrch model but without using task partitioning. It orchestrates each sub-task individually. It is the baseline for evaluating the partitioning method.

Table 3. The Properties of Sub-tasks for Applications C and D Sequential Dependencies

Application	No.(ζ^{Id})	required sub-task(ζ^{Rq})	Overall Latency(ζ^{Lat})	Computing Latency(ζ^{Lat})	Size (ζ^{Siz}) *(GIPS)*
App C (Resource-intensive)	1	-	0.3	0.5	30
	2	1	1.2	0.65	10
	3	2	2.5	1.25	20
	4	3	2.85	0.32	5
	5	4	4.9	2	50
	6	5	5	0.13	2
	7	6	6	0.95	15
	8	7	6.6	0.65	10
	9	8	7.5	0.8	12
	10	9	7.8	0.32	5
App D	1	-	0.7	0.65	10
	2	1	1.3	0.5	7
	3	2	2.1	0.8	12
	4	3	2.5	0.32	5
	5	4	3.2	0.65	10
	6	5	3.5	0.2	2
	7	6	4.5	1	15
	8	7	5.2	0.7	10
	9	8	6	0.8	12
	10	9	7.8	0.32	5

Table 4. Edge Devices Properties

Types	Edge server	Laptops	Smart phones	Raspberry Pis	Sensor Type1	Sensors Type2
Generate tasks	No	No	Yes	No	Yes	Yes
Total ratio(%)	1	11	18	11	48	12
Mobility	No	No	Yes	No	No	Yes
speed	0	0	1.4	0	0	12
CPU (GIPS)	400	110	25	16	-	-
CPU (cores)	10	8	8	4	-	-
Battery powered	No	Yes	Yes	No	No	No
Max computing energy (W)	0.07	0.0014	0.0066	0.0015	-	-
Static energy consumption (W)	0.027	0.00005	0.00047	0.001	-	-

— **DeTOrch with baseline DQN:** It is an implementation of the DeTOrch model with baseline DQN, which uses the same state as input but defines the reward as 0 and 1 for failure and success, respectively. It is the baseline for evaluating the proposed reward function.

— **Bacterial Foraging Optimization Algorithm (BFOA):** It is a meta heuristic model introduced in Reference [21]. It uses the same partitioning method, and we implemented it based on the objectives formulated in Section 14.

All aforementioned benchmark models based on DQN are configured using hyperparameters listed in Table 5.

Table 5. Hyperparameters and Their Values

Hyperparameter	Value
Learning Rate	0.0001
Discount Factor	0.8
Initial Exploration	0.99
Exploration Factor	0.9988
Final Exploration	0.1
Replay Memory Size	200000
Minibatch Size	50
Target Network Update Frequency	100

7.3 Evaluation Metrics

We define the following evaluation metrics, and in each scenario, we use a couple of them based on the scenario objectives and characteristics.

- **Composite task success rate:** It is calculated by dividing the total number of composite tasks that are successfully executed by the total number of generated tasks.
- **Sub-tasks success rate:** It is calculated by dividing the total number of sub-tasks that are successfully executed by the total number of generated sub-tasks.
- **Orchestrator service time:** It is the duration from when a sub-task/partition arrives at the orchestrator until the orchestrator assigns it to a device.
- **Orchestrator device energy usage:** It is calculated by the consumed energy per second multiplied by the orchestrator service time.
- **Composite task failure due to mobility:** It is calculated by the number of failed composite tasks divided by the total number of generated tasks.
- **Composite task completion time:** It is the sum of the waiting time and execution time of all sub-tasks. Each sub-task's waiting time includes the time it must wait for the sub-task it depends on to finish, as well as the time it needs to stay in a device queue for execution. The execution time of a sub-task depends on both the size of the task and the computing resources of the device.
- **Edge devices resource usage:** It is calculated by the number of sub-tasks that are sent to edge devices for execution divided by the number of generated sub-tasks.
- **Total network transmission energy usage:** It is the total energy used for transferring data over the network in the offloading process, measured by Equation (8).

7.4 Simulation Scenarios

To assess various aspects of our work, we consider three groups of scenarios:

7.4.1 Scenarios Group I. In the first group of scenarios, we aim to evaluate the effectiveness of the partitioning method in the proposed model. In these scenarios, we considered the *DeTOrch* as the proposed model and *DeTOrch without partitioning* as the baseline model.

Evaluation metrics: The evaluation metrics used in these scenarios are *Composite task success rate*, *Sub-tasks success rate*, *Orchestrator service time*, and *Orchestrator device energy usage*.

7.4.2 Scenarios Group II. In the second group of scenarios, we aim to evaluate the proposed DRL model for task orchestration. In these scenarios, the *DeTOrch* is considered as the proposed model, which is compared to (i) *DeTOrch with baseline DQN* and (ii) *BFOA*. To evaluate this model, all the characteristics of a real-world environment that are considered in general settings mentioned in Section 7.1 are also considered

Table 6. Scenarios Groups I and II Profile

Label	Dependency type	Workload	No. devices
Gen-M-100	General	Medium	100
Gen-H-100	General	High	100
Gen-H-250	General	High	200
Seq-M-100	Sequential	Medium	100
Seq-H-100	Sequential	High	100
Seq-H-250	Sequential	High	200

Table 7. Scenarios Group III Profile

Label	Low speed(%)	Medium speed(%)
Seq-H-200-L30	30	0
Seq-H-200-L30-M20	30	20
Seq-H-200-L30-M30	30	30

Evaluation metrics: The evaluation metrics used in these scenarios include the *Composite task success rate*, *Composite task failure due to mobility*, *Composite task completion time*, *Edge devices resource usage*, and *Total network transmission energy usage*.

For scenarios groups I and II, an environment with all characteristics explained in general settings (Section 7.1) are considered. These aspects are reflected in the six scenarios listed in Table 6. The simulation ran for 10 sets of 60 minutes each.

7.4.3 Scenarios Group III. The third group of scenarios aims to evaluate the mobility management component in the proposed model. In addition to the general settings described in Section 7.1, the following configurations are applied:

Resource-constrained heterogeneous devices: Five types of devices, as explained in general settings (Section 7.1) are considered, with only Raspberry Pis being replaced with mobile Tablets.

Heterogeneity of tasks and resource-intensive tasks: Sequential tasks as outlined in general settings (Section 7.1) are considered.

Number of devices and workload: The number of devices is set to 200, which all have a "High" workload.

Mobility: Three mobility behaviors are defined: (i) *Low mobility*: 30% of all devices in the environment are mobile at a low speed of 1.4 m/s (i.e., pedestrian) to the scenarios group I. (ii) *Medium mobility:* 50% of devices are mobile, 20% are moving at a medium speed of 4.3 m/s (i.e., bicycle riding), and 30% are moving at a low speed. (iii) *High mobility:* 60% of devices are mobile, with 30% of them moving at medium speed and 30% at low speed.

These aspects are reflected in the three scenarios listed in Table 7. The simulation ran for 10 sets of 60 minutes each.

Baselines and evaluation metrics: The DeTOrch model is used as the baseline in comparison with the proposed Mobility-aware DeTOrch model. The evaluation metrics used in these scenarios include the *Composite task success rate* and *Composite task failure due to mobility*.

7.5 Results and Discussions

7.5.1 Scenarios Group I - Partitioning Effectiveness Assessment.

Composite task success rate: As shown in Figure 6(a), DeTOrch, when using partitioning in all scenarios, outperforms the model that does not use partitioning, improving the composite task success rate by 10%. Furthermore, upon comparing the success rate of sub-tasks as shown

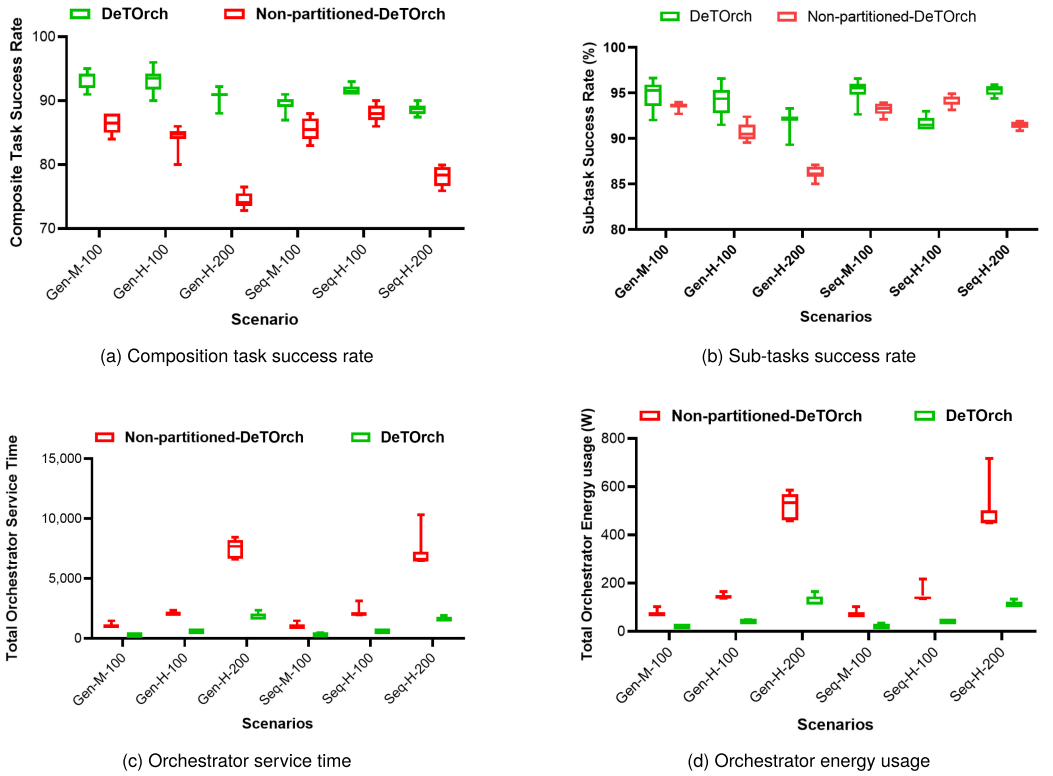


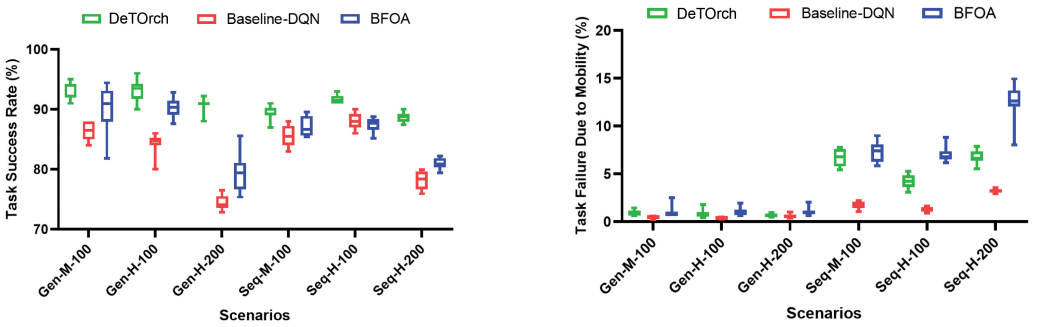
Fig. 6. The simulation results for comparing DeTOrch and DeTOrch without using task partitioning.

in Figure 6(b), it is evident that the improvement in success rate for sub-tasks is approximately 3%, whereas, for composite tasks, it is around 10%. This occurrence is due to partitioning, which reduces the impact of an incorrect resource assignment made by the orchestrator. For instance, let us consider a scenario where the orchestrator operates with 90% accuracy. In a non-partitioned system, to handle a composite task comprising 10 sub-tasks, each sub-task needs to be assigned to a resource individually. Even if one of the assignments out of 10 is incorrect, the entire composite task will fail. However, with partitioning, a composite task is partitioned into three, and the orchestrator finds a resource for all sub-tasks in a partition at once. Consequently, in three rounds, the orchestrator can handle all partitions of a composite task, and in 10 rounds, it can manage three composite tasks. Therefore, if 1 out of 10 assignments is incorrect, only one composite task will fail, and the other two will run successfully.

Orchestrator service time and energy usage: Figure 6(c) shows that the total service time of the DeTOrch orchestrator is significantly reduced when partitioning is used by around 70%. This is because partitioning allows the orchestrator to handle multiple sub-tasks with each action, resulting in fewer actions needed to orchestrate tasks. The orchestrator device's processor consumes a specific amount of energy for each process needed to carry out an action. Therefore, by reducing the number of required actions, the processor's time and energy consumption decreases, as shown in Figure 6(d).

7.5.2 Scenarios Group II - DRL Model Assessment.

Composite task success rate: The figure shown in Figure 7(a) indicates that DeTOrch outperforms the baselines in all scenarios of composite tasks. The proposed reward function enhances



(a) Composition task success rate

(b) Composition tasks failure due to mobility

Fig. 7. The simulation results for comparing DeTOrch task success and failure rate to the baselines.

the composite task success rate while increasing resource utilization of edge devices and reducing their completion time. When it comes to scenarios with general dependencies, DeTOrch improved the success rate by more than 8% compared to the DeTOrch with baseline DQN and more than 3% compared to the BFOA algorithm. In scenarios related to sequential tasks, DeTOrch increased the success rate by more than 4% compared to both baselines.

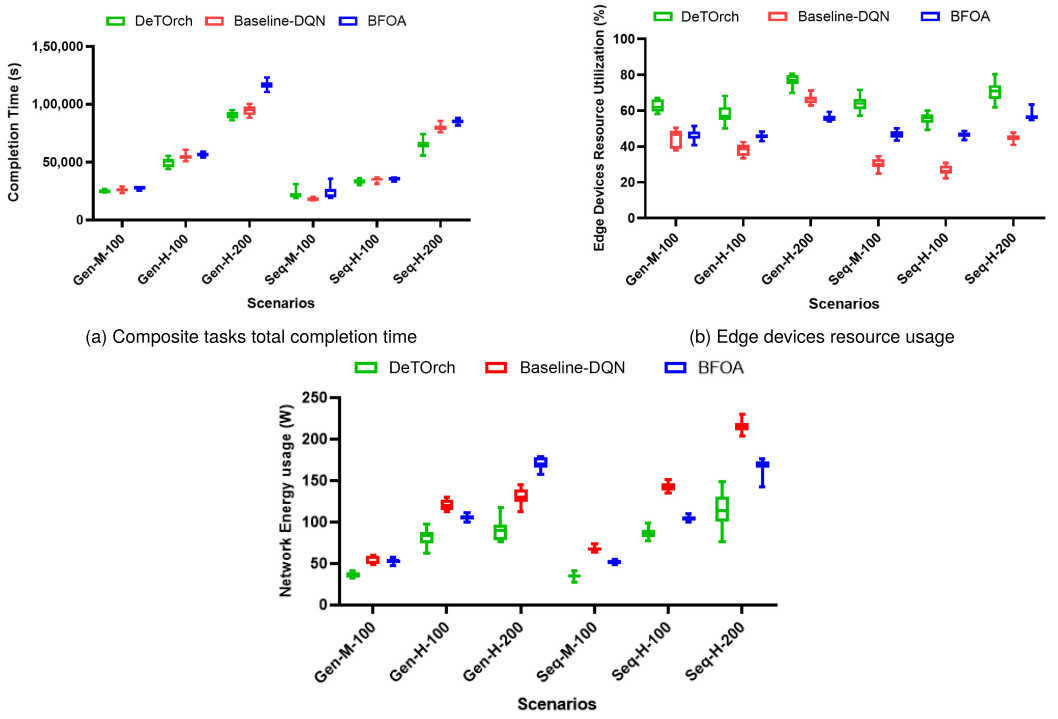
It is worth noting that as can be seen in Figure 7(a) in all scenarios with a high workload and a large number of devices, "Gen/Seq-H-200," the success rate improves significantly compared to other scenarios. This shows that DeTOrch is more scalable and reliable in extreme environments.

Edge devices resource utilization: The graph shown in Figure 8(b) illustrates that DeTOrch utilizes more resources on edge devices compared to the baselines in all scenarios. In most scenarios, DeTOrch has assigned over 60% of sub-tasks to edge devices, whereas the baseline DQN has assigned around 30% of sub-tasks to edge devices. On average, the BFOA has assigned about 50% of sub-tasks for execution on edge devices. The reason for this is the penalty that we have defined for offloading tasks to the edge server in cases when there is an alternative edge device available for partition offloading.

Composite task completion time: The results presented in Figure 8(a) demonstrate that DeTOrch significantly reduces the completion time in most scenarios compared to the baselines. However, in the "Seq-M-100" and "Seq-H-100" scenarios, all models perform equally well. This is because, in these two scenarios, the baseline models utilize the edge devices' resources less compared to other scenarios. As a result, most of the tasks are sent to the edge server, which is more powerful and can execute tasks faster.

Total network transmission energy usage: The results depicted in Figure 8(c) show that DeTOrch significantly decreases the network energy usage. On average, it reduces energy consumption by more than 30% in most scenarios. The reduction is attributed to DeTOrch sending fewer sub-tasks to the edge server, resulting in reduced data transmission between the edge server and edge devices. Consequently, the overall energy used for data transmission over the network is reduced.

7.5.3 Scenarios Group III - Mobility and Task Structures. Results show that the composite task with sequential dependency is more affected by the mobility of devices, as indicated in Figure 7(b). In scenarios with general dependencies, failure caused by mobility is around 1% on average, while in scenarios with sequential dependencies, it increases to more than 6% on average. The reason for this is that in sequential dependent sub-tasks, parallel execution is not possible. This

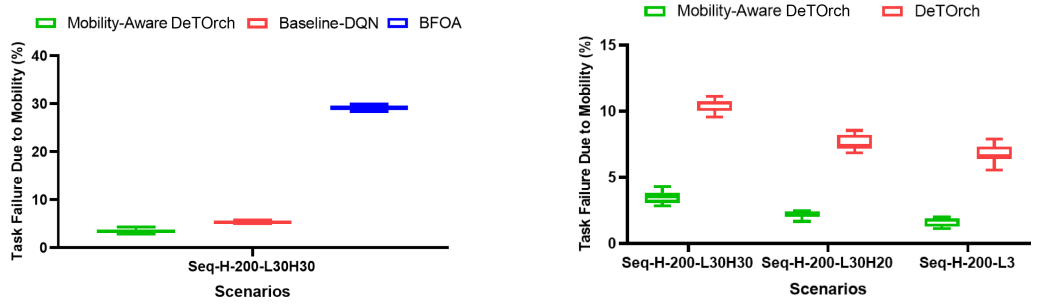


(a) Composite tasks total completion time

(b) Edge devices resource usage

(c) Total energy usage for data transmission over the network

Fig. 8. The simulation results for evaluating DeTOrch DRL algorithm effectiveness comparing the baselines.



(a) Composition task success rate

(b) Task failure due to mobility in various mobility behaviors

Fig. 9. The simulation results for comparing mobility-aware DeTOrch failure caused by mobility rate to the baselines.

can cause a significant delay between offloading the partition containing the last sub-task to a mobile device and the time it can be executed. This gap can be quite high, which increases the possibility of the resource device getting further away from the source device. Therefore, to evaluate the mobility manager, we defined all scenarios based on sequential tasks and high workload.

Composite task failure due to mobility: According to Figure 9(a), mobility-aware DeTOrch reduces failure caused by device mobility in the most complex scenario with 60% mobility,

outperforming DeTOrch with DQN baseline by around 4% and BFOA by 48%. Moreover, in comparison to DeTOrch, as shown in Figure 9(b), mobility-aware DeTOrch reduces failures due to mobility around 70% in all scenarios with varying mobility behaviors which causes to improve total task success rate by 5%. All of these are because of the mobility manager component, which can predict failure using the trajectory prediction algorithm. Therefore, the orchestrator can assign dependent partitions to devices that can transfer data safely when required.

In summary, to evaluate the effectiveness of partitioning, mobility manager, and the proposed DRL model for tasks with general and sequential dependencies, we defined three categories of scenarios. Our analysis showed that DeTOrch performs better in scenarios with general dependencies in terms of task success rate and network energy usage than in scenarios with sequential dependencies. However, for other evaluation metrics, tasks with both types of dependencies perform almost the same. The reason for this is that sub-tasks can be executed in parallel in tasks with general dependencies, whereas in sequential tasks, it is not possible. Therefore, the time between the last sub-tasks being offloaded and their execution time is high, which increases the risk of moving mobile devices and the distance between the devices that need to transfer data during the task offloading process. The mobility-aware DeTOrch, which uses the mobility manager component, improved the task success rate in sequential task scenarios by 5% compared to DeTOrch.

It is also worth noting that the DeTOrch model performs much better in extreme scenarios, "Seq/Gen-H-200," in terms of task success rate and completion time, demonstrating the scalability and reliability of our model.

8 Conclusion and Future Work

In this article, we extend a state-of-the-art task partitioning algorithm by proposing a deep reinforcement learning-based model for **Dependent sub-Task Orchestration (DeTOrch)** in an EEC environment. The model aims to improve the composite task success rate while increasing the usage of edge devices' resources and decreasing the composite task completion time. These objectives are achieved through partitioned-based DRL algorithms that we designed. In addition, we introduce a novel mobility manager model that employs trajectory prediction and can operate in an EEC environment consisting of devices with complex mobility behavior. To evaluate the effectiveness of partitioning, mobility manager, and the proposed DRL model, we defined three categories of scenarios for tasks with general and sequential dependencies. The results show that the mobility-aware DeTOrch outperforms the baselines in all scenarios and meets all defined objectives. Importantly, our approach promotes sustainability in computing systems by efficiently utilizing edge resources. It reduces the energy consumption associated with excessive reliance on high-capacity servers. It improves resource utilization across heterogeneous edge devices, thereby promoting sustainability in computing systems.

The DeTOrch model runs on a central orchestrator, which is the risk of overloading the orchestrator device or making it a single point of failure. To manage such situations, a network of orchestrators can be developed to facilitate federated learning, resulting in faster adaptation and enhanced model scalability, robustness, and fault tolerance. Additionally, implementing a load balancer can help distribute jobs among orchestrators, preventing overload on a single orchestrator.

Moreover, the partitioning algorithm partitions all composite tasks into a fixed number of partitions, regardless of their varying dependency types and environment characteristics. However, considering the dynamic nature of the EEC environment, future work could explore a dynamic partitioning model. Such a model can incorporate learning techniques that take into account task characteristics, environment characteristics, and resource availability as input parameters.

Acknowledgments

For the purpose of Open Access, the author has applied a CC BY public copyright license to any Author Accepted Manuscript version arising from this submission.

References

[1] Bilge Acun, Benjamin Lee, Fiodar Kazhamiaka, Kiwan Maeng, Udit Gupta, Manoj Chakkaravarthy, David Brooks, and Carole-Jean Wu. 2023. Carbon explorer: A holistic framework for designing carbon aware datacenters. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems* 2 (2023), 118–132.

[2] Ahmed A. Al-Habob, Octavia A. Dobre, Ana García Armada, and Sami Muhaidat. 2020. Task scheduling for mobile edge computing using genetic algorithm and conflict graphs. *IEEE Trans. Vehic. Technol.* 69, 8 (2020), 8805–8819.

[3] Laha Ale, Scott A. King, Ning Zhang, Abdul Rahman Sattar, and Janahan Skandaraniyam. 2022. D3PG: Dirichlet DDPG for task partitioning and offloading with constrained hybrid action space in mobile-edge computing. *IEEE Internet Things J.* 9, 19 (2022), 19260–19272.

[4] Mhd Saria Allahham, Amr Mohamed, Aiman Erbad, and Hossam Hassanein. 2022. On the modeling of reliability in extreme edge computing systems. In *5th International Conference on Communications, Signal Processing, and Their Applications (ICCSPA'22)*. IEEE, 1–6.

[5] Suman Banerjee, Neil Klingensmith, Peng Liu, and Anantharaghavan Sridhar. 2017. Edge computing in the extreme for sustainability. In *Communication Systems and Networks: 9th International Conference, COMSNETS 2017, Bengaluru, India, January 4–8, 2017, Revised Selected Papers and Invited Papers 9*. Springer, 93–109.

[6] Ulrik Brandes. 2001. A faster algorithm for betweenness centrality. *J. Math. Sociol.* 25, 2 (2001), 163–177.

[7] Ruitao Chen and Xianbin Wang. 2022. Situation-aware orchestration of resource allocation and task scheduling for collaborative rendering in IoT visualization. *IEEE Trans. Sustain. Comput.* 7, 4 (2022), 935–949.

[8] Santiago Correa, Gustavo Perez, Paulina Jaramillo, and Jay Taneja. 2023. Taking the long view: Enhancing learning on multi-temporal, high-resolution, and disparate remote sensing data. In *10th ACM International Conference on Systems for Energy-efficient Buildings, Cities, and Transportation*. 11–20.

[9] Chrysoula Dikonimaki. 2023. Edge compute offloading strategies using heuristic and reinforcement learning techniques. Ph. D. Dissertation. KTH Royal Institute of Technology, Stockholm, Sweden.

[10] Ankan Dutta, Surbhi Pal, Aishwarya Banerjee, Pratap Karmakar, Arpita Mukherjee, Debaprasad Mukherjee, and Prabal Kumar Sahu. 2023. Survey on irrigation scheduling with machine learning. In *International Conference on Smart Trends for Information Technology and Computer Communications*. Springer, 797–806.

[11] Mariam Elgamal, Doug Carmean, Elnaz Ansari, Okay Zed, Ramesh Peri, Srilatha Manne, Udit Gupta, Gu-Yeon Wei, David Brooks, Gage Hills et al. 2023. Carbon-efficient design optimization for computing systems. In *2nd Workshop on Sustainable Computer Systems*. 1–7.

[12] Mingjie Feng, Marwan Krunz, and Wenhan Zhang. 2021. Joint task partitioning and user association for latency minimization in mobile edge computing networks. *IEEE Trans. Vehic. Technol.* 70, 8 (2021), 8108–8121.

[13] Mingjin Gao, Rujing Shen, Long Shi, Wen Qi, Jun Li, and Yonghui Li. 2021. Task partitioning and offloading in DNN-task enabled mobile edge computing networks. *IEEE Transactions on Mobile Computing* 22, 4 (2021), 2435–2445.

[14] Fatemeh Golpayegani, Nanxi Chen, Nima Afraz, Eric Gyamfi, Abdollah Malekjafarian, Dominik Schäfer, and Christian Krupitzer. 2024. Adaptation in edge computing: A review on design principles and research challenges. *ACM Transactions on Autonomous and Adaptive Systems* 19, 3 (2024), 1–43.

[15] Jiaying Guo, Saeedeh Ghanadbashi, Shen Wang, and Fatemeh Golpayegani. 2023. Urban traffic signal control at the edge: An ontology-enhanced deep reinforcement learning approach. In *IEEE 26th International Conference on Intelligent Transportation Systems (ITSC'23)*. IEEE, 6027–6033.

[16] Leo Han, Jash Kakadia, Benjamin C. Lee, and Udit Gupta. 2024. Towards game-theoretic approaches to attributing carbon in cloud data centers. Retrieved from hotcarbon.org

[17] Amelia Holcomb, Linzhe Tong, and Srinivasan Keshav. 2023. Robust single-image tree diameter estimation with mobile phones. *Rem. Sens.* 15, 3 (2023), 772.

[18] Fatemeh Jalali, Kerry Hinton, Robert Ayre, Tansu Alpcan, and Rodney S. Tucker. 2016. Fog computing may help to save energy in cloud computing. *IEEE J. Select. Areas Commun.* 34, 5 (2016), 1728–1739.

[19] Youbin Jeon, Hosung Baek, and Sangheon Pack. 2021. Mobility-aware optimal task offloading in distributed edge computing. In *International Conference on Information Networking (ICOIN'21)*. IEEE, 65–68.

[20] Hidehiro Kanemitsu, Masaki Hanada, and Hidenori Nakazato. 2019. Multiple workflow scheduling with offloading tasks to edge cloud. In *12th International Conference on Cloud Computing (CLOUD'19), Held as Part of the Services Conference Federation (SCF'19)*. Springer, 38–52.

[21] Mandeep Kaur, Sanjay Kadam, and Naeem Hannoon. 2022. Multi-level parallel scheduling of dependent-tasks using graph-partitioning and hybrid approaches over edge-cloud. *Soft Comput.* 26, 11 (2022), 5347–5362.

[22] Noah Klugman, Joshua Adkins, Emily Paszkiewicz, Molly G. Hickman, Matthew Podolsky, Jay Taneja, and Prabal Dutta. 2021. Watching the grid: Utility-independent measurements of electricity reliability in Accra, Ghana. In *20th International Conference on Information Processing in Sensor Networks (Co-located with CPS-IoT Week'21)*. 341–356.

[23] Yu-Jen Ku, Sabur Baidya, and Sujit Dey. 2021. Adaptive computation partitioning and offloading in real-time sustainable vehicular edge computing. *IEEE Trans. Vehic. Technol.* 70, 12 (2021), 13221–13237.

[24] Dapeng Lan, Amir Taherkordi, Frank Eliassen, Lei Liu, Stéphane Delbruel, Schahram Dustdar, and Yang Yang. 2022. Task partitioning and orchestration on heterogeneous edge platforms: The case of vision applications. *IEEE Internet Things J.* 9, 10 (2022), 7418–7432.

[25] Hui Liu, Zhaocheng Niu, Junzhao Du, and Xiaomin Lin. 2023. Genetic algorithm for delay efficient computation offloading in dispersed computing. *Ad Hoc Netw.* 142 (2023), 103109.

[26] Shumei Liu, Yao Yu, Xiao Lian, Yuze Feng, Changyang She, Phee Lep Yeoh, Lei Guo, Branka Vucetic, and Yonghui Li. 2023. Dependent task scheduling and offloading for minimizing deadline violation ratio in mobile edge computing networks. *IEEE J. Select. Areas Commun.* 41, 2 (2023), 538–554.

[27] Lefu Maqelepo, Fhazhil Wamalwa, Nathan Williams, and Jay Taneja. 2025. Two sides of a coin: Assessing trade-offs between reliability and profit in mini grids and the policy implications for subsidies. *Appl. Energy* 378 (2025), 124726.

[28] Eric Masanet, Arman Shehabi, Nuoa Lei, Sarah Smith, and Jonathan Koomey. 2020. Recalibrating global data center energy-use estimates. *Science* 367, 6481 (2020), 984–986.

[29] Charafeddine Mechalikh, Hajer Taktak, and Faouzi Moussa. 2021. PureEdgeSim: A simulation framework for performance evaluation of cloud, edge and mist computing environments. *Comput. Sci. Inf. Syst.* 18, 1 (2021), 43–66.

[30] Mahshid Mehrabi, Shiwei Shen, Yilun Hai, Vincent Latzko, George P. Koudouridis, Xavier Gelabert, Martin Reisslein, and Frank H. P. Fitzek. 2021. Mobility-and energy-aware cooperative edge offloading for dependent computation tasks. *Network* 1, 2 (2021), 191–214.

[31] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski et al. 2015. Human-level control through deep reinforcement learning. *Nature* 518, 7540 (2015), 529–533.

[32] Aggrey Muhebwa, Colin J. Gleason, Dongmei Feng, and Jay Taneja. 2024. Improving discharge predictions in ungauged basins: Harnessing the power of disaggregated data modeling and machine learning. *Water Resour. Res.* 60, 9 (2024), e2024WR037122.

[33] Shishir G. Patil, Paras Jain, Prabal Dutta, Ion Stoica, and Joseph Gonzalez. 2022. POET: Training neural networks on tiny devices with integrated rematerialization and paging. In *International Conference on Machine Learning*. PMLR, 17573–17583.

[34] Jorge Portilla, Gabriel Mujica, Jin-Shyan Lee, and Teresa Riesgo. 2019. The extreme edge at the bottom of the internet of things: A review. *IEEE Sensors J.* 19, 9 (2019), 3179–3190. DOI: https://doi.org/10.1109/JSEN.2019.2891911

[35] Zahra Safavifar, Saeedeh Ghanadbashi, and Fatemeh Golpayegani. 2021. Adaptive workload orchestration in pure edge computing: A reinforcement-learning model. In *IEEE 33rd International Conference on Tools with Artificial Intelligence (ICTAI'21)*. IEEE, 856–860.

[36] Zahra Safavifar, Eric Gyamfi, Eleni Mangina, and Fatemeh Golpayegani. 2024. Multi-objective deep reinforcement learning for efficient workload orchestration in extreme edge computing. *IEEE Access* 12 (2024), 74558–74571.

[37] Zahra Safavifar, Charafeddine Mechalikh, and Fatemeh Golpayegani. 2023. Fault tolerant robust adaptive workload orchestration in pure edge computing. In *International Conference on Agents and Artificial Intelligence*. Springer, 370–386.

[38] Zahra Safavifar, Charafeddine Mechalikh, Junfei Xie, and Fatemeh Golpayegani. 2023. Enhancing VRUs safety through mobility-aware workload orchestration with trajectory prediction using reinforcement learning. In *2023 IEEE 26th International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, 6132–6137.

[39] Fuhong Song, Huanlai Xing, Xinhan Wang, Shouxi Luo, Penglin Dai, and Ke Li. 2022. Offloading dependent tasks in multi-access edge computing: A multi-objective reinforcement learning approach. *Fut. Gen. Comput. Syst.* 128 (2022), 333–348.

[40] Gilbert Strang. 1991. *Calculus*. Vol. 1. Wellesley-Cambridge, Cambridge, MA.

[41] Ju Wang, Liqiong Chang, Shourya Aggarwal, Omid Abari, and Srinivasan Keshav. 2025. Sustainable and low-cost greenhouse soil moisture monitoring using battery-free RFID sensors. *ACM Transactions on Sensor Networks* 21, 2 (2025), 1–28.

[42] Grant Wilkins, Srinivasan Keshav, and Richard Mortier. 2024. Hybrid heterogeneous clusters can lower the energy consumption of LLM inference workloads. In *15th ACM International Conference on Future and Sustainable Energy Systems*. 506–513.

[43] Carole-Jean Wu, Ramya Raghavendra, Udit Gupta, Bilge Acun, Newsha Ardalani, Kiwan Maeng, Gloria Chang, Fiona Aga, Jinshi Huang, Charles Bai et al. 2022. Sustainable AI: Environmental implications, challenges and opportunities. *Proc. Mach. Learn. Syst.* 4 (2022), 795–813.

[44] Huaming Wu, William J. Knottenbelt, and Katinka Wolter. 2019. An efficient application partitioning algorithm in mobile environments. *IEEE Trans Parallel Distrib. Syst.* 30, 7 (2019), 1464–1480.

[45] Baris Yamansavascilar, Ahmet Cihat Baktir, Cagatay Sonmez, Atay Ozgovde, and Cem Ersoy. 2021. DeepEdge: A deep reinforcement learning based task orchestrator for edge computing. *arXiv preprint arXiv:2110.01863* (2021).

Received 21 June 2024; revised 22 November 2024; accepted 2 February 2025