c2mec: Cooperative Multi-split and Multi-hop Edge Computing Based on Deep Reinforcement Learning

XIAOJIE ZHANG, Hunan First Normal University, Changsha, China
PENG WANG, School of Electronics and Information Engineering, Hunan First Normal University, Changsha, China
SHIMA YOUSEFI, City University of New York The Graduate Center, New York, United States
SAPTARSHI DEBROY, City University of New York The Graduate Center, New York, United States
KEQIN LI, Department of Computer Science, State University of New York, New Paltz, United States and Hunan University, Changsha, China

Recent research highlights a critical gap between the computing capabilities of modern IoT devices and the computational demands of Artificial Intelligent (AI) applications. The edge computing paradigm offers a promising solution by providing reliable and fast computing services close to the data source. Due to their inherent resource constraints, a single edge server often cannot handle the heavy computational load from nearby IoT devices, requiring multi-server collaboration. However, achieving efficient cooperation is challenging due to dynamic workload fluctuations and uneven data distribution. To address these issues, this article presents a novel solution that involves optimizing task execution paths and resource management to enhance the performance of edge servers, particularly in scenarios with unbalanced data or uneven distribution of IoT devices. Our approach not only deploys multiple edge servers, but also focuses on the intelligent allocation and management of computing tasks. Specifically, we propose *c2mec*, a cooperative multi-split and multi-hop edge computing framework. The proposed *c2mec* framework uses problem decomposition to efficiently decouple the variables that need to be optimized. In addition, *c2mec* employs a multi-agent Deep Reinforcement Learning (DRL) based algorithm to mitigate the negative impact of decoupling and provides a flexible data splitting strategy. Finally, *c2mec* designs an energy-aware training method for IoT devices to reduce their long-term training cost. Through our comprehensive experimental results, we demonstrate how *c2mec* achieves notable improvement in energy saving across different scenarios compared to existing solutions, such as full and partial task offloading.

CCS Concepts: • **Computing methodologies → Distributed computing methodologies**; • **Computer systems organization → Real-time systems;**

Additional Key Words and Phrases: Edge computing, task partitioning, task offloading, resource allocation, energy efficiency, Deep Reinforcement Learning.

This work was supported by the National Natural Science Foundation of China (Grant No. 62573187) and by grants from the Natural Science Foundation of Hunan Province, China (Grant No. 2024JJ7093).

Authors' Contact Information: Xiaojie Zhang, Hunan First Normal University, Changsha, Hunan, China; e-mail: xzhang6@gradcenter.cuny.edu; Peng Wang (corresponding author), School of Electronics and Information Engineering, Hunan First Normal University, Changsha, Hunan, China; e-mail: pwang@hnu.edu.cn; Shima Yousefi, City University of New York The Graduate Center, New York, NY, USA; e-mail: syousefi@gradcenter.cuny.edu; Saptarshi Debroy, City University of New York The Graduate Center, New York, NY, USA; e-mail: saptarshi.debroy@hunter.cuny.edu; Keqin Li, Department of Computer Science, State University of New York, New Paltz, NY, USA and Hunan University, Changsha, Hunan, China; e-mail: lik@newpaltz.edu.

ACM 1539-9087/2026/09-ART76
https://doi.org/10.1145/3772281

ACM Reference Format:
Xiaojie Zhang, Peng Wang, Shima Yousefi, Saptarshi Debroy, and Keqin Li. 2026. c2mec: Cooperative Multi-split and Multi-hop Edge Computing Based on Deep Reinforcement Learning. *ACM Trans. Embedd. Comput. Syst.* 25, 5, Article 76 (September 2026), 28 pages. https://doi.org/10.1145/3772281

1 Introduction

With the increasing popularity of **Internet of Things** (**IoTs**) and **Artificial Intelligence** (**AI**) technologies, the concepts of data and computing intelligence are increasingly gaining popularity across various use cases, such as smart cities [52] and Industry 4.0 [21]. What follows is the explosive amount of diverse IoT data brought about by various types of applications (such as image recognition, natural language processing, 3D reconstruction, etc) [11, 58]. At the same time, AI algorithms have made great progress, evolving from pioneering small-size models, such as *AlexNet* [18], *ResNet* [49], and *VGG* [10] to the now widely recognized **Large Language Models** (**LLMs**), such as *LLaMA* [41], *Bloom* [22], and *GPT-4* [1]. However, as a commonly known issue, the computing capabilities of modern IoT devices (e..g., drones, robots), either through in-built compute capabilities or externally attachable low-power compute units (e.g., NVIDIA Jetson nano, Jetson TX2, Raspberry Pi) are failing to match the computation requirements of emerging new applications, which tending to become more computationally-intensive every day [47]. Existing literature [56] shows that continuous execution of even small-size AI models on Nvidia Jetson TX2 (with an integrated 256-core Pascal GPU [37]) led to significant model inference time and considerable energy consumption. Therefore, in most smart use cases, the role of IoT devices is still limited to data collection, low-power communications, and sometimes only performing very small computation tasks.

As an alternative to in-device computation, use of traditional cloud computing platforms have been proposed for smart applications [15, 33]. However, with the rise in IoT data volume and growing concern for the protection of personal privacy data, cloud computing is gradually being replaced by edge computing [54]. Edge computing is a faster, more reliable, and more private computing paradigm where the computation servers often closer to the application data generation site [28, 48, 50]. Despite these advantages, such edge servers typically have significantly lower computation capacity compared to cloud data-centers. Therefore, a single edge server can quickly become overwhelmed by the computational demands of densely distributed IoT devices in smart city applications. To address this issue, collaboration among geographically distributed edge servers using multi-hop task offloading can be a game changer in meeting the continuously growing computing demands. A notable example is [19], that introduces an energy-efficient edge computing framework enabled by satisfaction games and approximate computing to intelligently distribute tasks between MEC servers and autonomous UAV-based edge nodes, thereby enhancing scalability and user satisfaction under diverse QoS constraints. Such inter-server collaboration aims to achieve load balancing for improved management of unbalanced data (or IoT device) distribution and service coverage in dynamic smart IoT-edge environments [14].

Although there are a large number of related works seeking to achieve collaborative edge computing involving multi-hop task offloading [4, 12, 16, 26, 35, 40], many works either simplify the system models or lack flexibility in the multi-hop assumption. On the other hand, the joint optimization problems inherent in edge computing are generally NP-hard due to the highly correlated variables (e.g., server selection, transmission power, relay node selection, and CPU frequency allocation, etc), which can have both discrete and continuous values. Works like [6, 8, 20, 27, 42, 57], they utilize decomposition (some of them use the term *bi-level*) methods to mitigate the computational complexity of solving intricate NP-hard problems. However, the decoupling process (e.g., **block coordinate descent** (**BCD**), game theory and matching) defined in these works can often result in

local optimal solutions. In contrast, [24, 25, 31, 53] employ **Deep Reinforcement Learning** (**DRL**) algorithms to tackle part of the challenging problems such as task offloading and channel allocation. By learning from the interactions between DRL agents and the edge environment, DRL-based algorithms yield more robust and efficient solutions that can adapt to dynamic edge computing environments. However, none of these DRL-based studies consider the training costs associated with IoT devices, or they assume the availability of offline training, which is often impractical in real-world scenarios. Furthermore, there have been limited efforts focused on splitting task data into multiple segments and transmitting each to a distant edge server in a multi-hop manner. To the best of our knowledge, this article is among the first few that aims to design an online algorithm capable of enabling such multi-split and multi-hop task offloading, alongside effective resource management.

In this article, we propose a cooperative multi-split and multi-hop stochastic task offloading framework for IoT networks, viz. *c2mec*, for smart applications. In *c2mec*, application data can be transmitted to a distant **hybrid access point** (**HAP**) that works as both computation and communication site through multi-hop communication. Additionally, *c2mec* enables IoT devices to partition their collected raw data into multiple segments, with execution path assigned to each such segment that spans from the HAP directly connected to the IoT device to the HAP responsible for processing that specific data segment. To tackle the data and computational fluctuations that are characteristic to edge environments, we formulate a long-term optimization problem aimed at minimizing the average energy consumption of all IoT devices while meeting their task deadlines. The proposed optimization problem results in a classic combinatorial problem that poses formidable computational challenge in finding high-quality solutions, both efficiently and promptly. To this end, *c2mec* adopts a divide-and-conquer approach to efficiently address the variables that need to be optimized. Particularly, in the first step, *c2mec* employs a distributed DRL-based algorithm to search solutions for IoT-to-HAP association and data splitting sub-problems. Then, *c2mec* employs a heuristic algorithm, designed to construct execution paths for every data segment. Thereafter, *c2mec* applies a sub-gradient method to derive the optimal resource allocation strategy of HAPs, and to configure the optimal local computation speed as well as the transmission power of IoT devices. The main contributions of this article are:

(1) We devise a long-term **Mixed Integer Non-linear Programming** (**MINLP**) problem, which allows *c2mec* to jointly optimize a variety of discrete and continuous variables, including transmission power, local computation speed and data splitting strategy of IoT devices, as well as IoT-to-HAP association, selection of task execution path, and resource allocation strategy of HAPs.
(2) To obtain high-quality solutions following variable decoupling, we propose a **distributed DRL-empowered Solution Searching** (***DSS***) algorithm. Specifically, we use the **Proximal Policy Optimization** (**PPO**) algorithm to implement the DRL agents for individual IoT devices.
(3) Additionally, *c2mec* strives to strike a balance between DRL's training costs and the attained solution quality, which is accomplished by gradually reducing the training steps in each time period.
(4) Finally, we implement *c2mec* framework in our hardware testbed where both the schedulability and scalability of *c2mec* are evaluated. The results demonstrate the high efficiency of *c2mec*'s data-splitting strategy in managing unbalanced task data distributions from IoT devices. Compared to other task offloading paradigms, *c2mec* achieves energy savings ranging from 16.1% to 35%.

The rest of the article is organized as follows. Section 2 presents the related works. Section 3 proposes the system model and problem formulation. Section 4 presents the problem decomposition. Sections 5 and 6 propose algorithms of *c2mec*. Section 7 discusses performance evaluation. Section 8 concludes the article.

2 Related Works

2.1 Multi-hop Task Offloading

The multi-hop task offloading problems have been studied in different research fields such as vehicular networks and UAV networks, involving a range of optimization factors including resource allocation, user association, and network flow scheduling, to name a few.

The work presented in [35] addresses the challenge of multi-hop, multi-task partial computation offloading in collaborative edge computing networks, with the objective of minimizing the average completion time of tasks. Their proposed algorithm, *JPOFH*, is developed based on task prioritization, integrating data partitioning and network flow scheduling. Other works, such as [26] and [4], investigate multi-hop task offloading problems in VEC networks. In [26], a task offloading scheme in VEC networks is designed with mobility and connectivity analysis. Utilizing a semi-definite relaxation-based approach for task allocation, the proposed offloading policy leverages one-hop and multi-hop service vehicles driving on the same road to minimize the cost of client vehicles. Similarly, in [4], a bat-based algorithm for k-hop task offloading in VEC networks is proposed, where candidate vehicle selection is determined by calculating the path connection time and path transmission time between vehicles. *Although these works considered network flow scheduling or task execution path selection, they ignore the impact of resource allocation on handling multiple simultaneous (or concurrent) computing tasks.*

Additionally, work, such as like [12, 16, 40] study multi-hop computation offloading problems in UAV networks, where UAV transmission is utilized to expand the coverage of computing service. In [12], UAVs are divided into aerial edge server nodes and relay nodes, where each task is transmitted to an assigned computing UAV in a multi-path and multi-hop manner with the help of other relay UAVs. In [16], a multi-UAV and multi-task computation offloading scheme is proposed, where UAVs are deployed in a fixed sequence between remote users and a terrestrial edge-server. It is assumed that the UAVs are working collaboratively, with each UAV executing a fraction of the received tasks and passing the remaining tasks to the subsequent UAV in the predefined sequence. A fairness based multi-hop collaborative algorithm is developed in [40], which aims to tackle the joint optimization problem of user-to-UAV association, UAV-to-UAV association, task offloading and resource allocation. Specifically, each UAV splits the offloaded data into various parts, which are then transmitted to a group of nearby UAVs for processing. Compared to the aforementioned papers, this article aims to address a scenario that closely aligns with the problem we have described. However, local computation and network flow scheduling were not considered in this work, as it specifically focuses on 2-hop task offloading. While the authors of the aforementioned works investigated multi-hop or cooperative task offloading in edge networks, they relied on certain strong assumptions for their individual systems and frameworks. *To the best of our knowledge, there is very limited research on the joint optimization of user-server association, resource allocation and task execution path selection, particularly for edge computing networks in multi-user and multi-server scenarios.*

2.2 Joint Optimization Problems

So far as we know, many existing problems involve joint optimization of multiple variables (as we have explored earlier). Especially in edge computing use cases, variables are often highly coupled to each other and the optimizations problems are usually formulated as MINLP problems. In the

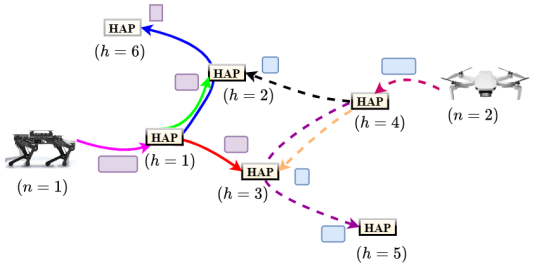

Fig. 1. An example of multi-split and multi-hop task offloading framework, where the boxes are the task data segments.

literature, researchers have employed either decomposition methods (sometimes referred to as bi-level methods) or deep learning algorithms to tackle the challenges posed by complex edge computing environments. The authors in [27] present a BCD method, which solves the server assignment and frame resolution selection sub-problems sequentially and iteratively by fixing one variable at a time. [42] decomposes the original complex MINLP problem into resource allocation and task offloading. The authors employ a heuristic algorithm that repeatedly performs one of two local operations–either the *remove* operation or the *exchange* operation–to derive the binary task offloading decisions. A Lyapunov optimization based online method is developed in [20], where the task offloading problem is formulated by a distributed non-cooperative game. Other works like [6, 8] also use bi-level optimization methods to decouple discrete variables from the original problems. [8] utilizes Hungarian method to find the optimal offloading strategy, while [6] formulates a two-sided matching game to implement sub-channel allocation. Indeed, bi-level optimization methods can significantly reduce the computational complexity of solving a complicated MINLP problem. However, they can also lead to the challenge of easily converging to local optimal solutions.

In contrast to the aforementioned papers, works such as [5] **Deep Q-Network** (**DQN**), [59] **Double Deep Q-Network** (**DDQN**), [7] **Deep Deterministic Policy Gradient** (**DDPG**), [30] **Advantage Actor-Critic** (**A2C**), [45] PPO employ DRL algorithms to tackle the joint task offloading and resource allocation problems, or other relevant problems. This deep learning approach is particularly effective for addressing discrete and combinatorial long-term optimization problems. Among the various types of DRL algorithms, PPO stands out as a type of Actor-Critic based method. This algorithm combines the strengths of both value-based (e.g., DQN and DDQN) and policy-based methods (e.g., DDPG), making it increasingly popular for its robustness and stability in solving complex and dynamic edge computing problems. *Nevertheless, none of the previously mentioned works have taken into account the task execution path selection problem and training cost of IoT devices in edge scenarios, where multi-hop task offloading and resource allocation are jointly optimized.*

3 System Model

In this article, we propose *c2mec* framework and explore a standalone edge architecture designed for dense IoT networks. *c2mec* consists of an undirected service graph $\mathcal{G} = \{\mathcal{H}, \mathcal{L}\}$ alongside a collection of IoT devices $\mathcal{N} = \{1, \dots, n, \dots, N\}$ (as depicted in Figure 1), serving as users. In $\mathcal{G}$, the vertices $\mathcal{H} = \{1, \dots, h, \dots, H\}$ are the interconnected HAPs. Each HAP $h \in \mathcal{H}$ is composed of a wireless **Access Point** (**AP**) linked to an edge server with heterogeneous two-dimensional resources denoted by $\{F_h^{HAP}, B_h^{HAP}\}$, where F_h^{HAP} and B_h^{HAP} refer to computation and radio resources (e.g., this article considers B_h^{HAP} as sub-channels), respectively. The link $l_{j \to j'} \in \mathcal{L}$ of $\mathcal{G}$, if it exists, it denotes the physical connection between HAP j and HAP j'. Conversely, the absence of such a link implies that the jth HAP and the j'th HAP are unable to communicate directly, typically due

Table 1. Table of Notations

Symbol	Definition	Symbol	Definition
$\mathcal{N}$	Set of IoT devices	$\mathcal{H}$	Set of HAPs
$l_{j\to j'}$	The physical connection between HAP j and HAP j'	F_h^{HAP}	The amount of computational resources of HAP h
B_h^{HAP}	The amount of network resources of HAP h	$< d_n(t), c_n(t) >$	The data size and computational complexity of tasks generated by nth IoT device
$o_{n,i}$	The association indicator between nth IoT device and ith HAP	$d^{n,h}(t)$	The amount of data generated by nth IoT device and processed by hth HAP
$r_{j\to j'}$	The link rate between HAP j and HAP j'	$b_{n,i}(t)$	The amount of allocated network resources from ith HAP to nth IoT device
$p_n(t)$	The transmission power of nth IoT device	$t_{n\to i}^{up}(t)$	The task offloading time of nth IoT device
$t_{n\to h}^{link}(t)$	The data transmission time over a path between nth IoT device and hth HAP	$\chi_{n\to h}(t)$	The set of links connects n-th IoT device and hth HAP
$e_n^D(t)$	The energy consumption of nth IoT device for task offloading	$e_n^C(t)$	The energy consumption of nth IoT device for local computation
$f_n^{IoT}(t)$	The computation speed of nth IoT device	$e_n^{DRL}(t)$	The energy consumption of nth IoT device for DRL model training

to being out of each other's transmission range. In this article, we use the expression $j \to j'$ to show the detection of a data flow. The major notations used in the system model are summarized in Table 1.

As illustrated in Figure 1, two devices (e.g., a ground robot and a drone) generate computational tasks, which are partitioned into smaller data segments. These segments are offloaded to different HAPs for processing via multi-hop communication paths. Each colored arrow represents a different transmission path, and the boxes indicate the segmented task data. This example highlights the flexibility of the offloading strategy supported by the proposed *c2mec* framework. A representative real-world use case that aligns with our system model is video surveillance. In such scenarios, IoT devices continuously capture high-resolution video streams to perform tasks such as abnormal behavior detection or crowd density estimation [32]. Due to the size of the video data and the latency constraints, processing the entire video on a single device or server is often infeasible. To address this issue, the video data is divided into multiple segments and each segment is offloaded to a different HAP. These segments are processed in parallel using identical models (e.g., object detection or activity recognition), thereby the system efficiency is significantly improved.

3.1 Two Timescales Optimization

While existing work primarily address the mobility patterns of IoT devices, *c2mec* uniquely targets a subset of IoT devices characterized by stationary deployment locations but encountering dynamic task arrivals. For instance, IoT devices that are deployed to monitor vehicle and pedestrian behavior alongside roads will adjust the video capture frequency, namely the *fps*. It will increase the *fps*

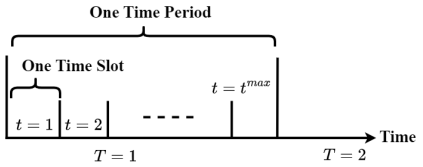

Fig. 2. *c2mec* uses a two timescales optimization system.

when traffic volume is high and decrease it when traffic volume becomes low. At times, it may also change the type of its AI models according to the needs of the scenario e.g., from car counting to anomaly detection.

With respect to such assumptions, *c2mec* uses an optimization system with two timescales (as illustrated in Figure 2), where the timeline is first divided into large time periods $\mathcal{T}_{large} = \{1, \dots, T, \dots, T^{max}\}$, usually lasting one or a few minutes. Then, each time period is further split into multiple smaller time slots $\mathcal{T}_{small} = \{1, \dots, t, \dots, t^{max}\}$, often lasting one to a few seconds. In each time slot t, we use a tuple $< d_n(t), c_n(t) >$ to represent the task features of IoT device n, where $d_n(t)$ indicates the amount of data generated by IoT device n (measured in bits) and $c_n(t)$ is the computational complexity of the task (measured in CPU cycles/bit).

In this article, we assume that $d_n(t)$ remains unchanged for all time slots $t \in [1, t^{max}]$ in the same time period. However, it may fluctuate between consecutive time periods i.e., T and $T + 1$. In realistic scenarios, the alterations in $c_n(t)$ may not occur too frequently, as it reflects the type of task being hosted. Therefore, we define a probability value $\rho \ll 1$ for all IoT devices, allowing each device to potentially switch its task type between consecutive time periods. Without loss of generality, we assume that tasks are released continuously at uniform intervals within each time period. The duration of each interval, denoted as the task deadline τ_n, is measured in seconds. By default, we consider τ_n to be equivalent to the length of a time slot.

3.2 Multi-split and Multi-hop Task Offloading

We assume that IoT devices can only be connected to at most one HAP during any given time slot within any time period (which is an universal assumption in the literature). Therefore, we define $o_{n,i}(t) \in \{0, 1\}$ as the association indicator, where $o_{n,i}(t) = 1$ indicates that the nth IoT device is connected to the ith HAP at time slot t; otherwise, $o_{n,i}(t) = 0$. In this context, i refers to the ID of the HAP to which an IoT device n is directly connected, and h refers to the relay node.

One of the challenges in edge computing is the limited computational capacity of edge servers when confronted with a large number of IoT devices. Unlike existing work, where tasks are processed either fully or partially by a single server, *c2mec* proposes a flexible multi-split and multi-hop task offloading strategy following *Definitions* 1 and 2.

Definition 1. Given that IoT device n is directly connected to HAP i, its task will be initially divided into two parts. The first part is processed by IoT device itself. The second part is offloaded to HAP i and further divided into several segments, where the hth segment is transmitted to hth HAP via intermediary relay HAPs. Therefore, the data splitting strategy is defined as follows:

$$\mathcal{D}_n(t) = \left\{ \overbrace{d^{n,0}(t)}^{\text{Local Segment}} \quad \overbrace{, d^{n,1}(t), \dots, d^{n,h}(t), \dots, d^{n,H}(t)}^{\text{Remote Segments}} \right\},$$

where $d^{n,0}(t)$ and $d^{n,h}(t)$ will be processed by the nth IoT device itself and the hth HAP, respectively. Obviously, the data splitting strategy $\mathcal{D}_n(t)$ needs to satisfy the following constraint:

$$d^{n,0}(t) + \sum_{h=1}^{H} d^{n,h}(t) = d_n(t), \forall n \in \mathcal{N}.$$

Definition 2. Given that IoT device n is directly connected to HAP i, each path $\chi_{n\to h}(t) \in \chi_n(t)$ is a set of connected links that connects HAP i to a target HAP h that processes one of the segments, whose size is $d^{n,h}(t)$. Thus, $\chi_{n\to h}(t)$ is defined as:

$$\chi_{n\to h}(t) = \left\{ l^{n,h}_{i\to j}(t), \dots, l^{n,h}_{j'\to h}(t) \right\},$$

where $l^{n,h}_{j\to j'}(t)$ indicates the virtual link (data flow) on route to the hth HAP from HAP i which connects HAP j and HAP j' (two relay nodes).

With to *Definitions* 1 and 2, we decompose the original multi-split and multi-hop task offloading strategy as constructing a collection of execution paths $\chi_n(t), \forall n \in \mathcal{N}$.

3.3 Data Multi-hop Transmission Time

We assume that HAPs are equipped with multiple antennas, and inter-HAP communication operates on proprietary high-speed connections, whether wireless or wired links. Thus, the data rate of all virtual links $l^{n,h}_{j\to j'}(t)$ between HAP j and HAP j' is presumed to be stable and fast, which is denoted by $r_{j\to j'}$ (measured by bits/s).

Given such assumption, the transmission time of sending a data segment $d^{n,h}(t)$ over a path $\chi_{n\to h}(t)$ is defined by:

$$t^{link}_{n\to h}(t) = \sum_{\forall l^{n,h}_{j\to j'}(t) \in \chi_{n\to h}(t)} \frac{\sum_{h'=1}^{H} d^{n,h'}_{j\to j'}(t)}{r_{j\to j'}}, \tag{1}$$

where $d^{n,h'}_{j\to j'}(t)$ is amount of data goes through virtual link $l^{n,h}_{j\to j'}(t)$, which is obtained from data splitting strategy $\mathcal{D}_n(t)$.

3.4 Task Offloading Time

To serve connected IoT devices $\{n | o_{n,i}(t) = 1,\ n \in \mathcal{N}\}$, HAPs implement an **Orthogonal Frequency-Division Multiple Access** (**OFDMA**) based spectrum sharing scheme, where HAP i is allocated a total of B_i^{HAP} sub-channels. We assume that a sub-channel can only be occupied by at most one IoT device $n \in \mathcal{N}$ for any given time slots $t \in \mathcal{T}_{small}$. Therefore, neither inter-HAP nor intra-HAP interference exists. Define σ^2 as the constant background noise variance, the data rate $r^{up}_{n\to i}(t)$ between the nth IoT device and its connected HAP i, is computed by:

$$r^{up}_{n\to i}(t) = b_{n,i}(t) w \log_2 \left(1 + \frac{p_n(t) g_{n,i}}{\sigma^2} \right), \tag{2}$$

where $b_{n,i}(t) = \frac{B_i^{HAP}}{\sum_{n=1}^{N} o_{n,i}(t)}$ is the number of allocated sub-channels via fair allocation schema [23] by HAP i. w is the bandwidth of a single sub-channel (measured in Hz). $p_n(t)$ is the transmission power of the nth IoT device allocated on a single sub-channel, and the total transmission power follows $b_{n,i}(t) p_n(t) \le p_{MAX}$. $g_{n,i}$ is the channel gain. Thus, the task offloading time is:

$$t^{up}_{n\to i}(t) = \frac{d_n(t) - d^{n,0}(t)}{r^{up}_{n\to i}(t)}, \tag{3}$$

and we rewrite the transmission power as:

$$p_n(t) = \mathbf{min}\left\{\frac{\mathbf{g}(r^{up}_{n\to i}(t))}{g_{n,i}}, \frac{P_{MAX}}{b_{n,i}(t)}\right\},$$

where $\mathbf{g}(r^{up}_{n\to i}(t)) = \sigma^2(2^{\frac{r^{up}_{n\to i}(t)}{b_{n,i}(t)w}} - 1)$ is a helper function.

3.5 Path Latency

In *c2mec*, the latency of path $\chi_{n\to h}(t)$ is the end-to-end time of processing data segment $d^{n,h}(t)$, which can be computed by:

$$t_{n,h}(t) = \overbrace{t^{up}_{n\to i}(t)}^{\text{Task Offloading}} + \overbrace{t^{link}_{n\to h}(t) + \frac{d^{n,h}(t)c_n(t)}{x_{n,h}(t)}}^{\text{Data Multi-hop and Computation}}. \tag{4}$$

In Equation (4), $x_{n,h}(t)$ shows the amount of computational resources allocated to nth IoT device from hth HAP (measured by CPU cycles/s). Clearly, the execution time for all offloaded segments is determined by the maximum path latency among all paths in $\chi_n(t)$, which can be stated as:

$$t_n(t) = \max\left\{t_{n,h}(t) | \forall \chi_{n\to h}(t) \in \chi_n(t)\right\}, \forall n \in \mathcal{N}. \tag{5}$$

We note that the physical resource allocation mechanism for concurrent tasks is indeed very important, however, it is beyond the scope of this article. The *c2mec* focuses on system-level optimization for resource management, especially the decision-making strategies that improve overall task processing efficiency in a distributed edge environment. For more information on physical resource allocation, readers can refer to the relevant literature in [29].

3.6 Energy Consumption of IoT Devices

In time slot t, the energy consumption of IoT devices $\forall n \in \mathcal{N}$ involves three components (given nth IoT device is connected to the ith HAP):

(1) *Task Offloading*: The energy consumption associated with transmitting all remote data segments to HAP i, which is stated as:

$$e^D_n(t) = t^{up}_{n\to i}(t) \times b_{n,i}(t) \times p_n(t). \tag{6}$$

(2) *Local Computation*: The energy spent to process local segment by IoT device itself, which is defined as:

$$e^C_n(t) = \kappa \times d^{n,0}(t) \times c_n(t) \times \left(f^{IoT}_n(t)\right)^2, \tag{7}$$

where $\kappa = 10^{-28}$ Joule per cycle [44] represents a constant associated with the computing chip architecture found in IoT devices (such as *NVIDIA Pascal™ GPU* or *VideoCore VII GPU*). The local computation speed of the nth IoT device during time slot t, denoted as $f^{IoT}_n(t)$, adheres to the constraint $f^{IoT}_n(t) \le F^{IoT}_n$, where F^{IoT}_n denotes its maximum speed. Therefore, the optimal local computation speed $f^{IoT}_n(t)$ that leads to minimum computation energy consumption is defined by:

$$f^{IoT}_n(t) = min\left\{\frac{d^{n,0}(t)c_n(t)}{\tau_n}, F^{IoT}_n\right\}. \tag{8}$$

(3) *DRL-model Training* (i.e., the *DSS* algorithm, which will be introduced in Section 5): In DRL training, the major training cost comes from the interaction phase, where the agent explores the environment to collect experience for learning. Therefore, the energy consumption

associated with executing the *DSS* algorithm is addressed in this article. The objective of the *DSS* algorithm is to find a high-quality solution for IoT-to-HAP association and data-splitting strategy by utilizing a DRL agent. We treat the training cost of the agent as a constant, denoted by $e_n^{DRL}(t)$ (per time unit). Furthermore, within each time period T (i.e., 1 to t^{max} time slots), the *DSS* algorithm allows the agent to be trained for $\delta(T)$ time slots to explore potential solutions, where $\delta(T)$ gradually decreases over time.

Based on the above points, the total energy consumption of IoT devices in a single time period is the cumulative sum of task cost and training cost, which can be expressed as follows:

$$e_n(T) = \overbrace{\sum_{t=1}^{t^{max}} \left(e_n^D(t) + e_n^C(t)\right)}^{\text{Task Offloading and Computation Cost}} + \overbrace{\sum_{t=1}^{\delta(T)} e_n^{DRL}(t)}^{\text{DRL-model Training Cost}}. \tag{9}$$

4 Problem Formulation and Decomposition

In this article, we propose a long-term optimization problem which jointly optimizes data splitting strategy of IoT devices and resource orchestration of HAPs. Our objective is to minimize the long-term average energy consumption of IoT devices, while satisfying the latency constraints. This design choice reflects real-world scenarios where IoT devices have strict energy budgets and their real-time computation requirements are explicitly modeled as latency constraints. The proposed optimization problem is stated as follows:

$$\begin{aligned}
&\min_{\Psi} \lim_{T^{max}\to+\infty} \frac{1}{T^{max}} \sum_{T=1}^{T^{max}} \left(\frac{1}{N}\sum_{n=1}^{N} e_n(T)\right) \\
\text{s.t. } &\mathbf{C1:}\ d^{n,0}(t) + \sum_{h=1}^{H} d^{n,h}(t) = d_n(t), \forall n \in \mathcal{N} \\
&\mathbf{C2:}\ \max\left\{t_{n,h}(t) | \forall \chi_{n\to h}(t) \in \chi_n(t)\right\} \le \tau_n, \forall n \in \mathcal{N} \\
&\mathbf{C3:}\ \sum_{n=1}^{N} x_{n,h}(t) \le F_h^{HAP}, \forall h \in \mathcal{H} \\
&\mathbf{C4:}\ \sum_{h=1}^{H} b_{n,h}(t) p_n(t) \le P_{MAX}, \forall n \in \mathcal{N} \\
&\mathbf{C5:}\ \frac{d^{n,0}(t) c_n(t)}{\tau_n} \le F_n^{IoT}, \forall n \in \mathcal{N} \\
&\mathbf{C6:}\ o_{n,i}(t) \in \{0,1\}, \forall n \in \mathcal{N}, \forall i \in \mathcal{H} \\
&\mathbf{C7:}\ \sum_{h=1}^{H} o_{n,i}(t) = 1, \forall n \in \mathcal{N},
\end{aligned} \tag{$\mathcal{P}_1$}$$

where Ψ is a set of variables that need to be optimized. Specifically, these variables are:

(1) The transmission power of IoT devices, denoted by $\mathbf{p}(t) = [p_1(t), \ldots, p_N(t)]$.
(2) The local computation speed of IoT devices, i.e.,$\mathbf{f}(t) = [f_1^{IoT}(t), \ldots, f_N^{IoT}(t)]$.
(3) The data splitting strategy of IoT tasks, denoted by $\mathcal{D}_n(t), \forall n \in \mathcal{N}$.
(4) The IoT-to-HAP association, denoted by $\mathbf{o}_n(t) = [o_{n,1}(t), \ldots, o_{n,H}(t)], \forall n \in \mathcal{N}$.
(5) The allocation of computational resources from HAPs, i.e., $\mathbf{x}_h(t) = [x_{1,h}(t), \ldots, x_{N,h}(t)], \forall h \in \mathcal{H}$.
(6) The task execution paths, denoted by $\chi_n(t), \forall n \in \mathcal{N}$.

In Problem ($\mathcal{P}_1$), **C1** and **C2** specify the constraints on the data splitting strategy and the corresponding path execution deadline; **C3** represents the constraints on the computing resource allocation from the HAP side; **C4** and **C5** limit the maximum transmission power and computing speed of IoT devices, respectively; **C6** and **C7** stipulate that IoT devices are only allowed to connect to one HAP in any given time period.

By building the execution paths alongside the links between HAPs, we can implement a highly flexible task offloading strategy. This strategy is assumed to significantly enhance the load balance and scalability of the service network in response to the dynamic features of IoT tasks. However, Problem ($\mathcal{P}_1$) is non-trivial to solve as it is a classic MINLP problem. The main difficulties in solving this problem come from two aspects: (1) the high coupling of optimization items and (2) the time-varying task parameters, which require that the problem be solved repeatedly, continuously, and quickly. Additionally, constraint **C2** involves sequences and discrete values, making Problem ($\mathcal{P}_1$) a type of combinatorial optimization problem, which is notoriously challenging.

4.1 Problem Decomposition: *c2mec* Framework

To solve Problem ($\mathcal{P}_1$), we design *c2mec*, an innovative edge framework that breaks down Problem ($\mathcal{P}_1$) into the following sub-problems or steps:

(1) At 1^{st} step, *c2mec* runs the *DSS* algorithm to solve the IoT-to-HAP association $\mathbf{o}_n(t)$ and the data splitting $\mathcal{D}_n(t)$ sub-problems, for every IoT device in $\mathcal{N}$.
(2) In step 2, *c2mec* proposes a heuristic algorithm to construct the execution paths $\chi_n(t)$ under given data splitting strategy $\mathcal{D}_n(t)$, $\forall n \in \mathcal{N}$.
(3) Then, in 3^{rd} step, *c2mec* finds the optimal local computation speed $\mathbf{f}(t)$ of IoT devices with known data splitting strategy $\mathcal{D}_n(t)$, for every IoT device in $\mathcal{N}$ (based on Equation (8)).
(4) At 4^{th} step, based on offloaded data segments, the computational resource allocation from the HAP-end $\mathbf{x}_h(t)$, $\forall h \in \mathcal{H}$ is conducted.
(5) In 5th step, the transmission power $p_n(t)$ is optimized to ensure that the total task completion time precisely matches the task deadline τ_n, for every IoT device in $\mathcal{N}$.
(6) Ultimately, a global reward is established and DRL models are updated.

As noted earlier, decomposing through such methods can often result in local optima. To mitigate this negative impact of decoupling, *c2mec* develops a distributed *DSS* algorithm aimed at discovering high-quality solutions for the sub-problems of IoT-to-HAP association $\mathbf{o}_n(t)$ and data splitting $\mathcal{D}_n(t)$. Utilizing the *DSS* paradigm, *c2mec* enables the edge system to learn through interactions between agents and the edge environment. Additionally, *c2mec* employs deep learning to address the challenges traditional RL faces when interacting with high-dimensional state-action spaces. Specifically, *c2mec* uses a **Deep Neural Network** (**DNN**) to approximate the mapping policy between states and actions. Given the high complexity and dynamics of the edge computing environment, we select PPO as the underlying implementation of our *DSS* algorithm. The algorithm flowchart is depicted in Figure 3.

5 DRL-empowered Solution Searching

The *c2mec* uses a DRL-empowered Solution Searching algorithm, namely *DSS*, to find high-quality solutions within each time period. Specifically, DRL models are trained on IoT devices and used to generate high-quality solutions through model reasoning.

5.1 MDP

We model the long-term optimization Problem ($\mathcal{P}_1$) as a **Markov Decision Process** (**MDP**), where the basic model can be defined as a tuple $< \mathcal{S}, \mathcal{A}, \mathcal{P}, R, \gamma >$. $\mathcal{S}$ and $\mathcal{A}$ denote the state and action spaces. In this article, we define $\mathbf{a}(t) = [a_1(t), \ldots, a_N(t)]$ as the joint action taken by IoT devices

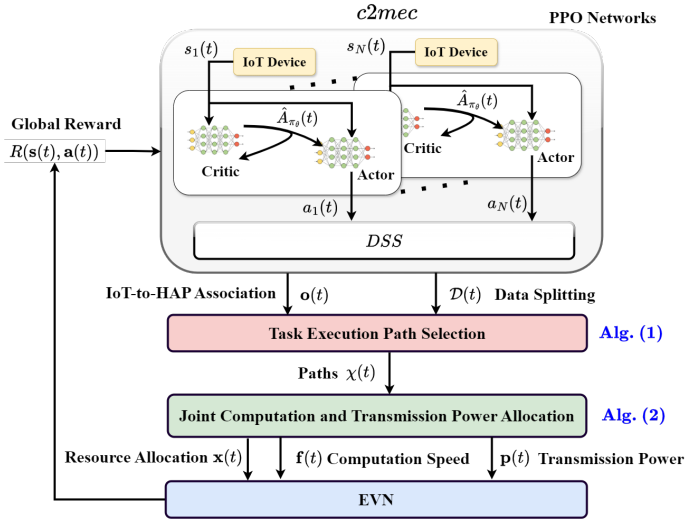

Fig. 3. *c2mec* framework.

and define $\mathbf{s}(t) = [s_1(t), \ldots, s_N(t)]$ as the joint system state. $\mathcal{P}(\mathbf{s}(t+1)|\mathbf{s}(t), \mathbf{a}(t))$ is the unknown transition probability from state $\mathbf{s}(t)$ to $\mathbf{s}(t+1)$ by taking action $\mathbf{a}(t)$. In *c2mec*, all IoT devices share a centralized reward value (i.e., $R_n(s_n(t), a_n(t)) = R(\mathbf{s}(t), \mathbf{a}(t)), \forall n \in \mathcal{N}$), aiming at collaboratively minimizing the average energy consumption of all IoT devices and ensuring that all tasks can be processed with their deadlines.

5.2 State $\mathcal{S}$

The state $s_n(t) \in \mathcal{S}$ of each IoT device $n \in \mathcal{N}$ specifies the partially observed environmental information at the start of time slot t. This information should provide valuable insights for decision-making, which should also being realistic and easily observable from an IoT device perspective. Therefore, the state $s_n(t), \forall n \in \mathcal{N}$ is defined as:

$$
\begin{aligned}
s_n(t) = \{ & \\
& \{\psi_{n,1}, \ldots, \psi_{n,H}\} \textit{// distance to HAP} \\
& \{F_1^{MAX}, \ldots, F_H^{MAX}\} \textit{// computation resource of HAP} \\
& \{B_1^{HAP}, \ldots, B_H^{HAP}\} \textit{// network resource of HAP} \\
& \{o_{n,1}(t), \ldots, o_{n,H}(t)\} \textit{// IoT-to-HAP association} \\
& \{d^{n,0}(t), \ldots, d^{n,H}(t)\} \textit{// data splitting} \\
& \{d_n(t), c_n(t)\} \textit{// task features} \\
& \{p_n(t), f_n^{IoT}(t)\} \textit{// transmission power and computation speed} \\
& \{x_{n,1}(t), \ldots, x_{n,H}(t)\} \textit{// computation resource allocation of HAP} \\
& \{b_{n,1}(t), \ldots, b_{n,H}(t)\} \textit{// network resource allocation of HAP} \\
& \{t_n(t), e_n(t)\} \textit{// latency and energy consumption of IoT device} \\
& \{done_n(t)\} \textit{// task completion indicator} \\
& \}
\end{aligned}
\tag{10}
$$

We set $done_n(t) = 1$ if the tasks of nth IoT device can be processed within its deadline τ_n in time slot t; otherwise $done_n(t) = 0$.

5.3 Action $\mathcal{A}$

When defining the action space, we adopt a scheme in which, once certain optimization variables are set, other lightweight algorithms can be employed to directly obtain optimal or high-quality solutions for the remaining variables. In this article, the action $a_n(t)$ of the nth IoT device is given by:

$$\begin{aligned} a_n(t) = \{ & \\ & \{\hat{o_{n,1}}(t), \ldots, \hat{o_{n,H}}(t)\} \text{// IoT-to-HAP association} \\ & \{d^{n,0}(t), \ldots, d^{n,K}(t)\} \text{// Top-K data splitting strategy} \\ & \} \end{aligned}, \tag{11}$$

where $\hat{o_{n,h}}(t) \in [0, 1]$ is a relaxed value of IoT-to-HAP association and $\hat{o_{n,h}}(t)$ is converted back to integer value by:

$$o_{n,h}(t) = \begin{cases} 1, & h = \underset{h \in \mathcal{H}}{\operatorname{argmax}}\, \hat{o_{n,h}}(t) \\ 0, & otherwise \end{cases}. \tag{12}$$

In Equation (11), *c2mec* employs a constant $K \in [1, H]$ to limit the maximum number of segments into which data can be divided, i.e., one local segment plus K remote segments. This feature distinguishes *c2mec* from other approaches. We assign priorities to HAPs through relaxed IoT-to-HAP association $\{\hat{o_{n,1}}(t), \ldots, \hat{o_{n,H}}(t)\}$. Subsequently, the top K HAPs are selected based on their assigned priorities and *c2mec* allocates the kth data segment in $\{d^{n,1}(t), \ldots, d^{n,K}(t)\}$ to the HAP ranked kth priority in $\{\hat{o_{n,1}}(t), \ldots, \hat{o_{n,H}}(t)\}$.

5.4 Global Reward

In general, the reward function should be constructed directly from the objective of their proposed optimization problem. We must confess that this approach works when there is a significant number of valid solutions (showing that all constraints can be easily satisfied). However, in scenarios with intense competition for resources (e.g., $N \gg H$), even finding effective solutions becomes challenging. In this case, relying solely on the objective function is not enough to establish an effective learning model.

Therefore, *c2mec* takes both objective function and latency constraint of Problem ($\mathcal{P}_1$) into consideration, and creates a global reward which is shared by all IoT devices. Thus, all $R_n(s_n(t), a_n(t)),\ \forall n \in \mathcal{N}$ are defined as follows:

$$R(\mathbf{s}(t), \mathbf{a}(t)) = \frac{\sum_{n=1}^{N} done_n(t)}{N \times \sum_{n=1}^{N} (e_n^D(t) + e_n^C(t))}. \tag{13}$$

5.5 PPO Network

The core mission of our *DSS* algorithm is to find a good policy $\pi_\theta : s(t) \rightarrow a(t), \forall n \in \mathcal{N}$, which is a mapping from states $\mathcal{S}$ to actions $\mathcal{A}$ (in this subsection, subscript n is ignored as the equations work for all IoT devices). More specifically, the policy aims to maximize the expected cumulative discounted reward ($\gamma \in (0, 1]$ is the discount factor), which is:

$$R_{\pi_\theta}(s(1)) = \mathbb{E}_{a(t)\sim\pi_\theta(a(t),s(t))}\left[\sum_{t=1}^{\infty}\gamma^t R(s(t),a(t))|s(1)\right].$$

Moreover, the **policy gradient** (**PG**) methods are widely used to directly optimize policy parameters θ [36], where the policy parameters are updated by $\theta = \theta + \alpha\nabla_\theta J(\theta)$ with α be defined as the learning rate and $\nabla_\theta J(\theta)$ be the gradient. In *DSS* algorithm, we choose a variant of PG methods, called A2C. In A2C, the actor network is responsible for learning the policy while the critic network evaluates the value function, which is denoted by $V_{\pi_\theta}(s(t))$. A general expression for A2C PG is:

$$\nabla_\theta J(\theta) = \mathbb{E}\left[\sum_{t=1}^{\infty}\nabla_\theta \log \pi_\theta(s(t),a(t))\hat{A}_{\pi_\theta}(t)\right],$$

where $\hat{A}_{\pi_\theta}(t)$ estimates the advantage function and is expressed by:

$$\hat{A}_{\pi_\theta}(t) = R(s(t),a(t)) + \gamma V_{\pi_\theta}(s(t+1)) - V_{\pi_\theta}(s(t)).$$

In episodic environments (like our time slotted system), $\hat{A}_{\pi_\theta}(t)$ can be simply estimated by $\hat{A}_{\pi_\theta} = G(t) - V_{\pi_\theta}(s(t))$. $G(t)$ is the empirical returns in one training episode (or one time period).

To improve the stability and exploration-exploitation balance [43] of A2C, we adopt to the PPO algorithm by introducing a clipped surrogate objective function into A2C and it is defined as:

$$J^{CLIP}(\theta) = \mathbb{E}\left[\min\left(r_t(\theta)\hat{A}_{\pi_\theta}(t), CLIP(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_{\pi_\theta}(t)\right)\right] \quad . \tag{14}$$

In Equation (14), $r_t(\theta)$ denotes the probability ratio between current policy θ and old policy θ_{old}, which is clipped between $1-\epsilon$ and $1+\epsilon$. ϵ is a clipping parameter (often set to 0.2). The motivation behind such clipped surrogate objective function is to prevent large policy updates that could have catastrophic consequences [34].

5.6 *c2mec*'s Reasoning Strategy

However, given that task features $< d_n(t), c_n(t) >$ remain unchanged in a single time period, it is counterproductive to perform reasoning across all its time slots (i.e., $t = 1$ to $t = t^{max}$). This is mainly because, even though DRL-empowered algorithms such as DDPG and PPO typically use models with relatively few parameters, continuous training on IoT devices can still result in high energy consumption. Moreover, constant training during a stable time period provides minimal improvement to the model's learning ability. Therefore, we argue that, in *c2mec*'s two timescales optimization system, the *DSS* algorithm does not necessary to obtain the optimal solution in every time slot. Instead, it can generate a series of high-quality solutions through continuous interaction with the environment and identify the best one. In *c2mec*, the lower the energy consumption brought by a solution, the higher its quality.

Specifically, as shown in Figure 4, *c2mec* divides each time period into two phases:

(1) Reasoning phase: During the reasoning phase, *c2mec* allocates the first $\delta(T)$ time slots to train the models, and let them explore the environment and generate a set of candidate solutions using multi-agent reinforcement learning methodology. This set of candidate solutions is denoted as $\mathcal{A}^{\text{cand}}(T) = \{\mathbf{a}(t) \mid t \in [1, \delta(T)]\}$.

(2) Acting phase: *c2mec* selects the best solution from $\mathcal{A}^{cand}(T)$ based on the highest quality. This optimal solution is denoted as $\mathbf{a}(t^*)$. Subsequently, from time slot $t = \delta(T) + 1$ to $t = t^{\max}$, *c2mec* consistently applies $\mathbf{a}(t^*)$ until the beginning of the next time period.

Clearly, a larger $\delta(T)$ increases the probability of obtaining a higher-quality solution. However, due to the inherent randomness involved in exploration, this benefit comes at the expense of higher energy consumption for the training process and extended periods of performance fluctuations.

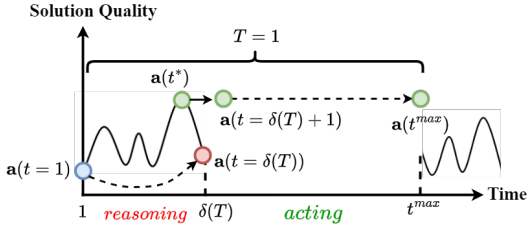

Fig. 4. Reasoning and acting strategy of *c2mec* framework.

Conversely, a small $\delta(T)$ struggles to leverage prior solutions effectively but can quickly stabilize the performance of a time period using a satisfactory solution with minimal energy expenditure. In *c2mec*, to manage this trade-off, we initially employ a large $\delta(T)$, and gradually reduces it by a constant δ_{step} in every T^C time periods until $\delta(T) = \delta_{min}$. Therefore, with *DSS* strategy, $\delta(T)$ is defined as follows:

$$\delta(T) = \max\left(\delta_{min}, \delta_{max} - \left\lfloor \frac{T}{T^C} \right\rfloor \times \delta_{step}\right). \tag{15}$$

This *DSS* strategy of *c2mec* helps to quickly initially train the base model and slowly reduce the training intensity as the model gains reasoning capabilities.

While *c2mec* assumes a static topology for evaluation clarity, the distributed nature of our decision-making framework and the stochastic nature of our task model enable it to adapt to dynamic changes in network topology naturally. In *c2mec*, each IoT device independently observes its local state $s_n(t)$ and makes offloading decisions $a_n(t)$ using its local agent. When the network topology changes (such as the addition or removal of HAP or IoT devices), these changes essentially reflect the variations in the degree of resource competition among IoT devices, which has the same effect as caused by the stochastic task model we proposed. Over the training time, the agent adapts to these changes based on the rewards it receives from the environment. Moreover, some well-studied training methodologies such as federated learning and transfer learning, may also be applied to address dynamic network topology. However, they are beyond the scope of this article. For further reference, readers can find related works in [2, 39, 46, 51].

6 Path Selection and Resource Allocation

This section addresses path selection and joint optimization of computation and transmission power allocation.

6.1 Path Selection

In *c2mec*, our objective in path selection is to minimize average data transmission time over the paths *s.t.*, for any HAP h, if $d^{n,h}(t) > 0$, then path $\chi_{n\to h}(t)$ must be in $\chi_n(t)$ and it is a complete sequence from directly connected HAP i to target HAP h. In Equation (1), an intuitive way to reduce the data transmission time is to let different segments of data in $\mathcal{D}_n(t)$ flow from different physical links $l_{j\to j'}$ to the corresponding computing nodes h, thereby achieving load balance.

To make path selection work in conjunction with the upper-layer DRL algorithm, we propose a heuristic algorithm powered by **Breadth First Search** (**BFS**), which is fast and efficient. The steps for each IoT device $n \in \mathcal{N}$ to establish their task execution paths $\chi_n(t)$ are described in Algorithm (1).

6.2 Resource Allocation and Transmission Power

Now, Problem ($\mathcal{P}_1$) is reduced into the following identical and one-shot resource allocation problems for each time slot t:

ALGORITHM 1: Multi-hop Path Selection

Input: IoT device n, connected HAP i, data splitting strategy $\mathcal{D}_n(t)$, a copy of links $\mathcal{L}_n$.
Set $target = i$.
Set $visited = \{target\}, next = \{target\}$.
Set $\chi_{n\to target}(t) = \emptyset$, $\chi_n(t) = \{\chi_{n\to target}(t)\}$.
while $next \neq \emptyset$ **do**
  $target = next.pop()$.
  for $l_{j\to j'} \in \mathcal{L}_n$ **do**
    if $target == j$ ***and*** $j' \notin (next \cup visited)$ **then**
      $next.add(j')$ and $visited.add(j')$.
      $\chi_{n\to j'}(t) = \chi_{n\to target}(t) \cup \{link_{j\to j'}\}$
      $\mathcal{L}_n.del(l_{j\to j'})$
      if $d^{n,j'}(t) > 0$ **then**
        $\chi_n(t).add(\chi_{n\to j'}(t))$
return $\chi_n(t)$

$$\begin{aligned}
&\min_{\mathbf{p},\mathbf{x}} \frac{1}{N}\sum_{n=1}^{N} b_{n,i}(t) \times \mathbf{min}\left\{\frac{\mathbf{g}(r^{up}_{n\to i}(t))}{g_{n,i}}, \frac{P_{MAX}}{b_{n,i}(t)}\right\} \times t^{up}_{n\to i}(t) \\
&\text{s.t. } \mathbf{C1:}\ t_{n,h}(t) \le \tau_n, \forall \chi_{n\to h}(t) \in \chi_n(t), \forall n \in \mathcal{N} \\
&\quad \mathbf{C2:}\ \sum_{n=1}^{N} x_{n,h}(t) \le F_h^{HAP}, \forall h \in \mathcal{H}.
\end{aligned} \tag{$\mathcal{P}_2(t)$}$$

Considering that the second derivative $\mathbf{g}''(r^{up}_{n\to i}(t)) > 0$, the Hessian matrix $\mathbf{H}$ of Problem ($\mathcal{P}_2(t)$) are positive semi-definite in terms of variables $x_{n,h}(t)$ and $t^{up}_{n\to i}(t)$, making Problem ($\mathcal{P}_2(t)$) strictly convex [44]. We then define the Lagrangian function of Problem ($\mathcal{P}_2(t)$) as follows:

$$\begin{aligned}
\mathrm{L}(\mathcal{P}_2(t)) = &\frac{1}{N}\sum_{n=1}^{N}\left(t^{up}_{n\to i}(t) \times \mathbf{min}\left\{\frac{\mathbf{g}(r^{up}_{n\to i}(t))}{g_{n,i}}, \frac{P_{MAX}}{b_{n,i}(t)}\right\} \times b_{n,i}(t)\right) + v_h\left(\sum_{n=1}^{N} x_{n,h}(t) - F_h^{HAP}\right) \\
&+ \sum_{n=1}^{N}\sum_{h=1}^{H} \lambda_{n,h}\left(t^{up}_{n\to i}(t) + t^{link}_{n\to h}(t) + \frac{d^{n,h}(t)c_n(t)}{x_{n,h}(t)} - \tau_n\right)
\end{aligned}, \tag{16}$$

where Lagrangian multipliers v_h and $\lambda_{n,h}$ are associated with constraints **C2** and **C1**, respectively. Based on Equation (16), we can apply the KKT conditions to yield the optimal solution for $x_{n,h}(t)$, which is given by:

$$x_{n,h}(t) = \frac{[\lambda_{n,h} \times d^{n,h}(t) \times c_n(t)]^{\frac{1}{2}}}{\sum_{n'=1}^{N}[\lambda_{n',h} \times d^{n',h}(t) \times c_{n'}(t)]^{\frac{1}{2}}} \times F_h^{HAP}. \tag{17}$$

Now, we expect all remote segments can be finished just before the task deadline by adjusting the transmission power. In above discussions, we have fixed all other items, which help us to compute the maximum remaining time for task offloading, which is denoted by:

$$t^{up}_{n\to i}(t) = \tau_n - \max\left\{t^{link}_{n\to h}(t) + \frac{d^{n,h}(t)c_n(t)}{x_{n,h}(t)} | \forall \chi_{n\to h} \in \chi_n(t)\right\}.$$

ALGORITHM 2: Joint Optimization of Resource Allocation and Transmission Power

1 Input: current time slot t, IoT-to-HAP association $\mathbf{o}_n(t)$, data multi-split strategy $\mathcal{D}_n(t)$, task execution paths $\chi_n(t)$ for every IoT device in $\mathbf{N}$.
2 Set $\lambda = [1, \dots, 1, \dots, 1, \dots, 1]$ and $step = 1$.
3 Compute path transmission time $t^{link}_{n\to h}(t)$ for paths $\forall \chi_{n\to h}(t) \in \chi_n(t)$ based on Equation (1) and data splitting strategy $\mathcal{D}_n(t)$.
4 Compute $\mathbf{x}_h(t), \forall h \in \mathcal{H}$ and $\mathbf{p}(t)$ based Equations (17) and (18).
5 Compute path execution time $t_{n,h}(t)$ for all paths in $\chi_n(t)$.
6 **while** $\exists t_{n,h}(t) > \tau_n, \forall \chi_{n\to h}(t) \in \chi_n(t)$ **do**
7 Update λ based on Equation (19).
8 Repeat line 4 and line 5 to re-compute path execution time.
9 $step += 1$.
10 **if** $step \geq max\ step$ **then**
11 **break**
12 **return** $\mathbf{x}(t), \forall h \in \mathcal{H}$ and $\mathbf{p}(t)$

Thus, the optimal transmission power is:

$$p^*_n(t) = \begin{cases} \mathbf{min}\left\{ \dfrac{\mathbf{g}\left(\frac{d_n(t) - d^{n,0}(t)}{t^{up}_{n\to i}(t)}\right)}{g_{n,i}}, \dfrac{P_{MAX}}{b_{n,i}(t)} \right\}, & t^{up}_{n\to i}(t) > 0 \\ \frac{P_{MAX}}{b_{n,i}(t)}, & t^{up}_{n\to i}(t) \leq 0 \end{cases}. \tag{18}$$

Finally, we recompute the task offloading time with the optimal transmission power $p^*_n(t)$ and obtain the path execution time for all paths, i.e., $t_{n,h}(t)$. With $t_{n,h}(t)$, we update the Lagrangian multipliers $\lambda_{n,h}$ via following expression:

$$\lambda_{n,h} = \lambda_{n,h} + \Delta_\lambda(t_{n,h}(t) - \tau_n), \tag{19}$$

where Δ_λ is the step size (e.g., 10^{-2}). The algorithm to find the optimal resource allocation and transmission power is described in Algorithm (2).

6.3 Overall Algorithm Implementation

This subsection delves into the detailed implementation of the *c2mec* framework, where the agents are implemented with PPO models. During each training episode, the agent's policy is trained by a set of experience replay generated by its current policy. We define $\mathcal{M}_n$ as the experience replay of the nth IoT device, it can be stated as:

$$\mathcal{M}_n = \{< s_n(t), a_n(t), R(\mathbf{s}(t), \mathbf{a}(t)), s_n(t+1) > | \forall t \in [1, \delta(T)]\}.$$

The complete algorithm of *c2mec* is depicted in **Algorithm** (3). At very beginning, each IoT device initializes a PPO model with policy parameter θ_n and set its experience replay as $\mathcal{M}_n = \emptyset$, $\forall n \in \mathcal{N}$. Then, in each time period T, *c2mec* will perform following phases:

(1) In reasoning phase (Line 4 to Line 16), the *DSS* algorithm (Section 5) is deployed to explore and find a high-quality solution. First, the set of candidate solutions is set to $\mathcal{A}^{cand}(T) = \emptyset$. In each time slot t, a joint action $\mathbf{a}(t) = \{\mathbf{a}_1(t), \dots, \mathbf{a}_N(t)\}$ is created by allowing IoT devices using their current policies θ_n, $\forall n \in \mathcal{N}$. Then, other variables that need to be optimized, such as local computation speed, task execution path, transmission power and resource allocation,

ALGORITHM 3: Online Training For DRL-empowered Solution Searching

1 Initialize PPO policy θ_n, $\forall n \in \mathcal{N}$.
2 Set experience replay $\mathcal{M}_n = \emptyset$, $\forall n \in \mathcal{N}$.
3 **for** ***each*** *time period* $T : 1 \rightarrow +\infty$ **do**
4 // reasoning phase starts
5 Set $\mathcal{A}^{cand}(T) = \emptyset$.
6 **for** *each time slot* $t \in \{1, 2, \dots, \delta(T)\}$ **do**
7 Generate joint action $\mathbf{a}(t) = \{\mathbf{a}_1(t), \dots, \mathbf{a}_N(t)\}$ by multi-agent reasoning.
8 According to action $a_n(t)$, all IoT devices do:
9 (1) Establish execution paths $\chi_n(t)$ by **Algorithm** (1) and data splitting strategy $\mathcal{D}_n(t)$ by Equation (11).
10 (2) Set local computation speed $f_n^{IoT}(t)$ based on Equation (8).
11 Run **Algorithm** (2) to obtain HAP resource allocation $\mathbf{x}_h(t), \forall h \in \mathcal{H}$ and transmission power $\mathbf{p}(t)$.
12 Set $done_n(t) = 0$ if $\exists t_{n,h}(t) > \tau_n, \forall \chi_{n \rightarrow h}(t) \in \chi_n(t)$; otherwise, set $done_n(t) = 1$, $\forall n \in \mathcal{N}$.
13 Compute $R(\mathbf{s}(t), \mathbf{a}(t))$ based on Equation (13).
14 All IoT devices store transition $(s_n(t), a_n(t), R(\mathbf{s}(t), \mathbf{a}(t)), s_n(t+1))$ into their local experience replay $\mathcal{M}_n$.
15 Save $\mathbf{a}(t)$ into $\mathcal{A}^{cand}(T)$.
16 // reasoning phase ends
17 // acting phase starts
18 Obtain best time slot $t^* = \underset{t \in [1, \delta(T)]}{\text{argmax}}\, R(\mathbf{s}(t), \mathbf{a}(t))$ from the set of candidate solutions $\mathcal{A}^{cand}(T)$.
19 **for** *each time slot* $t \in \{\delta(T) + 1, \dots, t^{max}\}$ **do**
20 Set $\mathbf{a}(t) = \mathbf{a}(t^*)$.
21 All IoT devices $n \in \mathcal{N}$ and HAPs $h \in \mathcal{H}$ behave based on $\mathbf{a}(t)$.
22 Update $\delta(T)$ based on Equation (15).
23 // acting phase ends
24 // updating phase starts
25 **for** *each IoT* $n \in \mathcal{N}$ **do**
26 **if** $|\mathcal{M}_n| \geq step_{th}$ **then**
27 IoT device n uses its experience replay $\mathcal{M}_n$ to update its policy network θ_n.
28 Set $\mathcal{M}_n = \emptyset$.
29 // updating phase ends

are configured (by **Algorithm** (2)). By the end of time slot t, a global reward $R(\mathbf{s}(t), \mathbf{a}(t))$ is observed from the environment, and all IoT devices store the corresponding transition $(s_n(t), a_n(t), R(\mathbf{s}(t), \mathbf{a}(t)), s_n(t+1))$ into their own experience replay $\mathcal{M}_n$. Finally, the action $\mathbf{a}(t)$ is saved into $\mathcal{A}^{cand}(T)$ as a candidate solution. After $\delta(T)$ time slots, the reasoning phase ends.

(2) In acting phase (Line 17 to Line 23), the best joint action $\mathbf{a}(t^*)$ is selected from $\mathcal{A}^{cand}(T)$, where $t^* \in [1, \delta(T)]$. For the remainder of time period T (i.e., $t \in [\delta(t) + 1$ to $t^{max}]$), we automatically set $\mathbf{a}(t) = \mathbf{a}(t^*)$, without any training. By the end of time period T, $\delta(T)$ is updated based on Equation (15).

(3) In policy update phase (Lines 24 to 29), the PPO model is updated when the size of experience replay exceeds $step_{th}$. In *c2mec*, we clear $\mathcal{M}_n$ once the current policy is updated.

The above phases will be carried out in sequence and repeated at different time periods. Gradually, $\delta(T)$ becomes smaller and the agent's strategy begins to converge.

Table 2. Simulation Parameters

Name	Value
H	8
N	$[26, 32, 38]$
P_{MAX}	1 $Watt$
F_n^{IoT}	2 GHz
$d_n(t)$	$[0.5, 1.5]$ MB
$c_n(t)$	$[1, 2.5] \times 10^9$ $CPU\ cycles\ per\ MB$
F_h^{HAP}	$[8, 15]$ GHz
B_h^{HAP}	$[8, 15]$ sub-channels
w	1 MHz
$r_{j \to j'}$	10 MB/s
δ_{max} / δ_{min} / δ_{step}	200 / 25/ 10
$step_{th}$	300
T^C	50

7 Performance Evaluation

Next, we show simulation results to validate the schedulability and scalability of *c2mec*. The *c2mec* framework is built using Pytorch and is tested on a Dell workstation with *13th Gen Intel® Core™ i9-13900K* and *Nvidia® RTX A6000*. All simulations last for 1,000 time periods.

7.1 Simulation Settings

To begin with, we consider a standalone edge computing network in square area of $1000m \times 1000m$, where multiple HAPs and IoT devices are randomly distributed in this area. The channel background noise is set to $\sigma^2 = -174\ dbm/hz$ [17]. We assume that the path-loss model from IoT device n to the ith HAP is estimated by $128.1 + 10\eta \log_{10}(\frac{\psi_{n,i}}{\psi_0})$ [13]. We set $\eta = 3.75$ in response to the urban area. $\psi_{n,i}$ is the distance between IoT device n and the selected HAP i. ψ_0 is the reference distance (1 km). For PPO networks, both the actor and critic networks are configured with 3 fully connected layers [9]. The network policy is updated for 10 epochs each time, the clip parameter is set to 0.2, and the discount factor is set to 0.9. Other parameters are summarized in Table 2.

7.2 Competing Algorithms

In this evaluation, we aim to compare the performance of algorithms designed based on different task offloading paradigms and DRL methods, i.e., **full offloading** (**FO**) and **partial offloading** (**PO**). Specifically, we perform and compare the following algorithms.

(1) *baseline*: IoT devices are always connected to nearest HAP with FO mode.
(2) *full* [55]: all IoT devices select the optimal HAP through a PPO model (simplified version of *c2mec*) and perform FO.
(3) *one-hop* [3]: in *one-hop*, IoT devices and their connected HAPs collaboratively process the task either partially or fully.
(4) *a2c* [38]: the agent is implemented based on the A2C model (on-policy method).
(5) *ddpg* [7]: the agent is implemented based on the DDPG model (off-policy method).
(6) *c2mec*: our proposed solution, where the maximum number of segments that data can be split into is $K = 3$.

Table 3. Competing Algorithms

	FO	PO	Multi-Hop	Multi-Split	*DSS*	*DRL*
baseline	√					
full	√				√	PPO
one-hop		√			√	PPO
a2c		√	√	√	√	A2C
ddpg		√	√	√	√	DDPG
c2mec		√	√	√	√	PPO

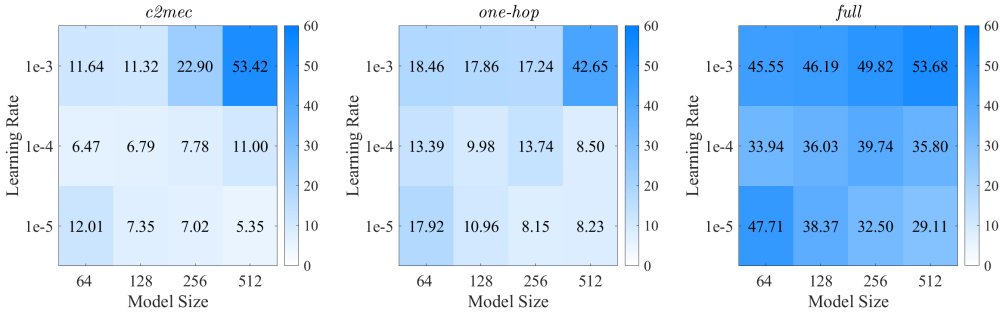

Fig. 5. Impact of learning rate and model size on energy consumption (J).

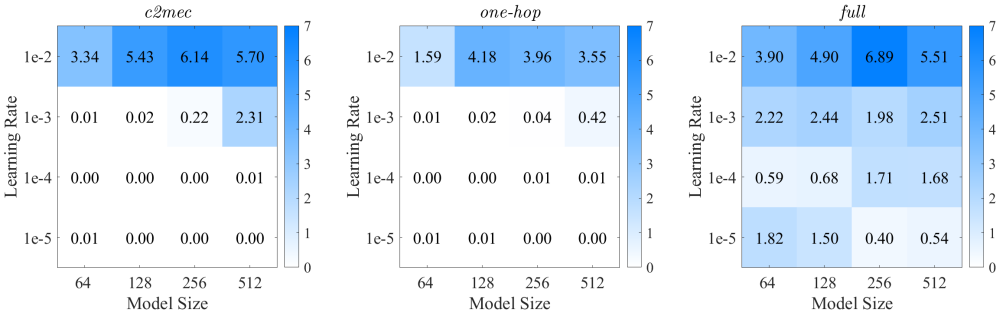

Fig. 6. Impact of learning rate and model size on number of failed tasks.

All algorithms, except *baseline*, are powered by *DSS*. Please also note that the underlying implementation of data splitting strategy in *one-hop* algorithm is also based on *c2mec* ($K = 1$), except that *one-hop* uses a standard PO paradigm. The detailed functioning comparisons of the competing algorithms are summarized in Table 3.

7.3 Configuring Learning Rate and Model Size of PPO

Our initial evaluation focuses on analyzing the impact of learning rate and model size of the PPO model on *c2mec*'s training efficiency. The results of long-term average energy consumption and the number of IoT devices that failed to meet their task deadlines are summarized in Figures 5 and 6, where the total number of IoT devices is set to $N = 32$. Since the algorithms, namely *c2mec*, *one-hop*, and *full*, utilize the same *DSS* strategy, their energy consumption during the solution search phase are identical. Therefore, these figures only present the task-related energy consumption. It has been observed that there is no clear monotonic relationship between learning rate and model size (in

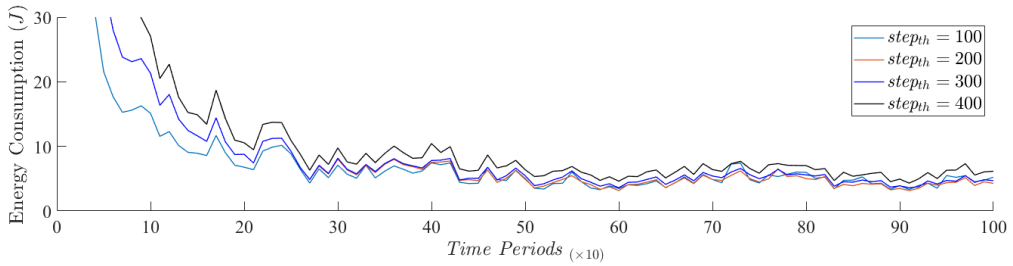

Fig. 7. Impact of $step_{th}$ on energy consumption (J).

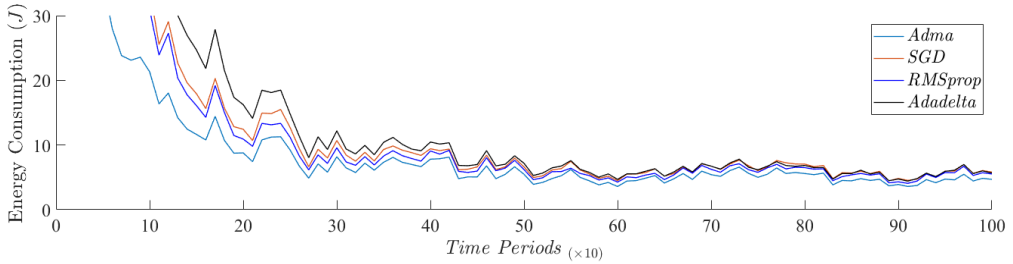

Fig. 8. Impact of optimizer on energy consumption (J).

terms of their impact on performance metrics). However, by combining both performance metrics, PPO models start to work efficiently when learning rate is less or equal than 10^{-4}.

In Figures 5 and 6, it can be clearly observed that the performance of the *full* algorithm is poor due to the limitations of the full task offloading strategy. In contrast, *one-hop* employs a PO strategy, resulting in nearly zero failed tasks and significant energy savings when $lr \leq 10^{-4}$. Compared to *one-hop*, our solution *c2mec*, which takes advantage of multi-hop data transmission and top-K data splitting, achieves further improvements. Across different model sizes, the best performance of *c2mec* in terms of minimal energy consumption shows an average improvement of 33.1% compared to the *one-hop* algorithm. For *c2mec*, the optimal configuration is $lr = 10^{-5}$ with a model size of 512, yielding a minimal energy consumption of 5.35 J per time period.

We also acknowledge that other hyperparameters, such as batch size and the choice of optimizer, can also influence a model's robustness and generalization performance. In off-policy methods (such as DQN and DDPG) that rely on random sampling from a replay buffer, the batch size used for sampling plays a crucial role. However, *c2mec* adopts PPO, an on-policy algorithm in which training data is generated directly from the current policy rather than sampled from past experience. Therefore, batch size is less critical in this context. Instead, *c2mec* focuses more on the policy update frequency, which is controlled by parameter $step_{th}$ (as shown in Algorithm 3, Lines 24–29). The impact of $step_{th}$ is depicted in Figure 7. Except for large update interval ($step_{th} = 400$), other values for $step_{th}$ behave similarly after a few training periods (the difference is within 6%). As for the choice of optimizer, we find *Adma* to be the best fit for our model as shown in Figure 8. Compared to other optimizers, such as *SGD*, *RMSprop*, and *Adadelta*, *Adma* can reduce energy consumption by 13% to 21%. Overall, *c2mec* has good convergence and robustness performance when appropriate hyperparameters are chosen.

7.4 Convergence Analysis

In this subsection, we evaluate the convergence behavior of *c2mec* in comparison with alternative task offloading paradigms (i.e., *full* and *one-hop*) and two state-of-the-art DRL methods (i.e., *ddpg*

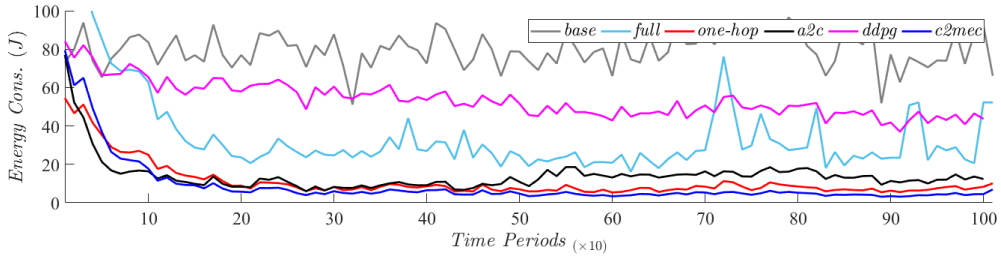

Fig. 9. Average energy consumption per IoT device in each time period against competing algorithms with $N = 32$ number IoT devices.

and *a2c*). The results are presented in Figure 9. The *base* algorithm demonstrates the highest level of energy consumption and the lowest degree of stability. While the *full* algorithm represents an improvement, exhibiting reduced energy consumption and diminished fluctuations, it still experiences noticeable instability, particularly within the time interval from 700 to 1,000 periods. Compared to *base* and *full* algorithms, the *one-hop* algorithm effectively adapts to dynamic computing environments by continuously adjusting the task offloading ratio, thereby demonstrating robust convergence characteristics. On the other hand, the other two DRL methods perform poorly. In particular, *ddpg*, an off-policy method, exhibits low learning efficiency and slow convergence. Although *a2c* performs well initially, it later suffers from overfitting, leading to energy consumption even higher than that of the *one-hop* algorithm. In contrast, the *c2mec* algorithm (which uses PPO) achieves the best performance, with the lowest energy consumption, highest stability, and fastest convergence.

7.5 Impact of Inter-HAP Communication

Since *c2mec* relies on multi-hop transmissions among HAPs, we acknowledge that the inter-HAP data rate $r_{j \to j'}$ can significantly influence its performance. To assess this impact, we conduct simulations under varying inter-HAP data rates and different values of the maximum number of segments into which a task can be split (i.e., the parameter K). As shown in Figure 10, the energy consumption of *c2mec* with HAP cooperation (i.e., $K = 2$ and $K = 3$) decreases greatly as the inter-HAP data rate increases. This observation confirms that *c2mec* benefits from higher inter-HAP communication bandwidth, particularly when multiple HAPs are involved in cooperative decision-making and task execution. In contrast, when $K = 1$ (equivalent to the *one-hop* scheme where no inter-HAP communication occurs), the energy consumption remains consistently high. These results suggest that *c2mec* can adaptively select an appropriate value of K based on the expected inter-HAP communication conditions. When the inter-HAP data rate is limited, a smaller K is preferable to minimize communication overhead. Conversely, when inter-HAP communication is fast and reliable, a larger K (e.g., $K = 3$) enables better cooperation and leads to higher energy efficiency.

7.6 Impact of Number of IoT Devices

This subsection focuses on the average energy consumption of IoT devices, where the number of devices gradually increases from $N = 26$ to $N = 32$ and $N = 38$, representing low, medium, and high distribution densities of IoT devices, respectively. The comparisons are summarized in Figure 11. For low-density ($N = 26$), *c2mec* demonstrates an approximate 25.9% reduction in energy consumption compared to the *one-hop* algorithm. This improvement becomes even more significant at medium density ($N = 32$), where *c2mec* achieves around a 35% reduction in energy consumption. Even at high density ($N = 38$), *c2mec* maintains better energy efficiency, with an approximate 16.1%

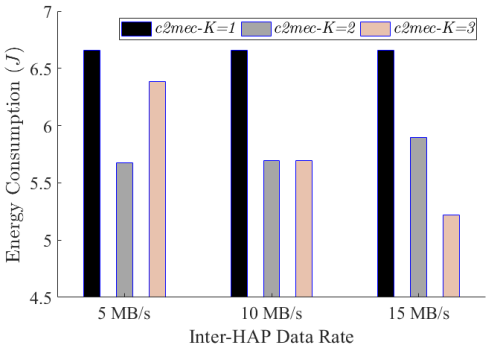

Fig. 10. Impact of Inter-HAP communication.

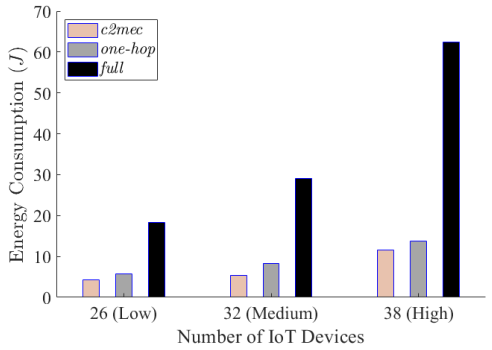

Fig. 11. Impact of number of IoT devices.

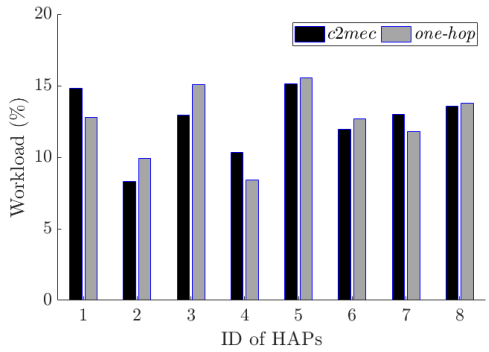

Fig. 12. An example of comparison between *c2mec* and *one-hop* algorithm in terms of workload distribution with 32 IoT devices.

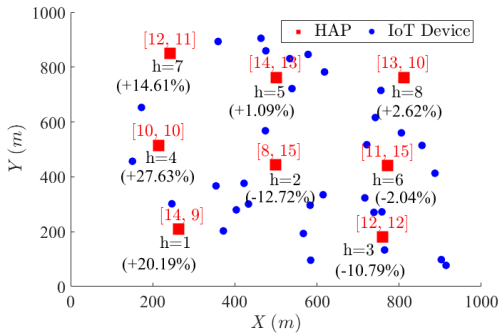

Fig. 13. An example of comparison between *c2mec* and *one-hop* algorithm in terms of load balance with 32 IoT devices. The coordinates $[v1, v2]$ indicate the normalized computation and network resource availability at each HAP. The percentage values, both positive and negative, represent the adjustments in workload made by *c2mec* compared to the *one-hop* algorithm.

reduction compared to *one-hop* (due to resource limitations, energy saving reaches a saturation point where further optimization yields marginal savings). Overall, *c2mec* consistently outperforms *one-hop* in terms of energy efficiency across all tested device densities, with the most notable improvements observed at lower and medium densities.

7.7 Load Balance

We also visualize an example of comparison between *c2mec* algorithm and *one-hop* algorithm in terms of load balance. The results are depicted in Figures 12 and 13. First of all, in Figure 12, we can clearly see that the workloads taken by an HAP is mostly proportional to its resource availability (e.g., HAP $h = 2$ and $h = 5$ have the lowest and highest workload, respectively), which signifies the effectiveness of the PPO model in maintaining load balance. In Figure 13, the number of IoT devices around HAP $h = 1$, $h = 4$ and $h = 7$ is obviously smaller than that in other HAPs. Therefore, the distribution of IoT devices (or we say the computation requirement) is unbalanced. While *one-hop* only aims at creating a good IoT-to-HAP association, *c2mec* additionally shifts the computational workload in places where IoT devices are densely distributed to places where IoT devices

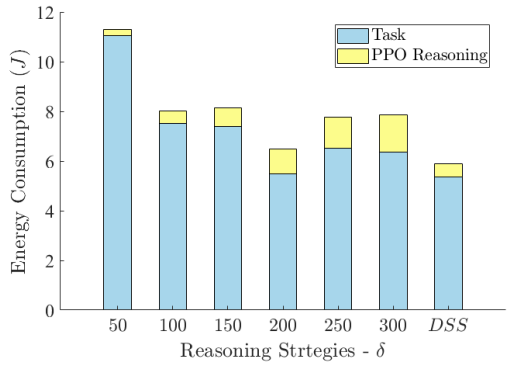

Fig. 14. *DSS vs* constant reasoning - δ. The reasoning energy consumption is set to $e^{DRL}(t) = 5 \times 10^{-2}$ per time slot based on experiments with *Nvidia Jetson TX2*.

are sparsely distributed through data splitting and multi-hop transmission. Thereupon, when applying the *c2mec* algorithm, some of HAPs have redistributed a certain amount of workload to other HAPs.

Specifically, the work load of HAP $h = 2$ and $h = 3$ (with a large number of neighboring IoT devices) is reduced by 12.72% and 10.79% with the implementation of the *c2mec* algorithm. Meanwhile, the workloads of $h = 1$, $h = 4$ and $h = 7$ are increased by 20.19%, 27.63%, and 14.61%, respectively. These adjustments highlight *c2mec*'s capability to effectively redistribute workloads, leading to more balanced network resource utilization compared to the *one-hop* approach.

7.8 *DSS vs* Constant Reasoning

Next, we analyze the impact of the length of reasoning phase at the beginning of each time period on the overall energy consumption. Specifically, we compare *c2mec* with non-adaptive algorithm which always perform fixed times of reasoning during the solution search phase, denoted by constant reasoning - δ (within $[0, t^{max}]$). The comparisons are summarized in Figure 14, where the average energy consumption for training is estimated by running PPO agent on *Nvidia Jetson TX2* (from our prior study [56]).

In Figure 14, low δ values (like $\delta \leq 150$) may not provide enough opportunities to find good solution, leading to higher task related energy consumption. Conversely, very high δ values (like $\delta \geq 250$) may consume excessive energy during the long training phase, offsetting the benefits of the optimized solution. Among constant reasoning strategies, $\delta = 200$ strikes a good balance between exploration (finding the best solution) and exploitation (applying the best solution), achieving a lower task-related energy consumption of 5.48 (*J*). In contrast, *c2mec* with *DSS* achieves the lowest task-related energy consumption of 5.35 (*J*) with a reasoning energy consumption of 0.53 (*J*), which is notably low compared to constant reasoning strategies with higher δ values, indicating its high efficiency. This is because deep learning models based on probability distributions tend to stabilize in their statistics (such as the mean, median, maximum, and minimum values) when the sample space expands sufficiently. In our scenario, each reasoning step can be viewed as exploring a portion of the sample space. Over a single time period, as the policy of the PPO model remains unchanged, the probability distribution of actions taken during that period also remains constant. Therefore, extending the duration of the reasoning phase makes it increasingly difficult to find a better solution and instead leads to higher average energy consumption due to the inherent randomness of the reasoning process. As shown in Figure 14, compared to the best constant reasoning strategy with $\delta = 200$, *c2mec* can still save 9.3% in total energy consumption.

8 Conclusions

In this article, we proposed *c2mec*, a cooperative multi-split and multi-hop edge computing framework for IoT networks in scenarios like smart cities. We showed how the proposed *c2mec* can allow IoT devices to split their data into multiple segments, and transmit each segment to a designated HAP. To minimize the energy consumption of IoT devices, we formulated a long-term MINLP problem to jointly optimize a series of discrete and continuous variables. We also showed how *c2mec* decomposed this problem by employing a PPO-based online algorithm for solving the most difficulty parts of the problem, which are IoT-to-HAP association and data splitting. Subsequently we proposed an efficient algorithm to establish all the execution paths. Hereafter, the proposed *c2mec* conducted a subgradient method to find the optimal resource allocation strategy as well as the optimal local computation speed and transmission power of IoT devices. With comprehensive simulation results, we showed how *c2mec* exhibited notable performance when compared to baseline solutions, such as FO or *one-hop* offloading.

References

[1] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. arXiv:2303.08774. Retrieved from https://arxiv.org/abs/2303.08774

[2] Aqeel Anwar and Arijit Raychowdhury. 2020. Autonomous navigation via deep reinforcement learning for resource constraint edge nodes using transfer learning. *IEEE Access* 8 (2020), 26549–26560.

[3] Furong Chai, Qi Zhang, Haipeng Yao, Xiangjun Xin, Ran Gao, and Mohsen Guizani. 2023. Joint multi-task offloading and resource allocation for mobile edge computing systems in satellite IoT. *IEEE Transactions on Vehicular Technology* 72, 6 (2023), 7783–7795.

[4] Chen Chen, Yini Zeng, Huan Li, Yangyang Liu, and Shaohua Wan. 2023. A multihop task offloading decision model in MEC-enabled internet of vehicles. *IEEE Internet of Things Journal* 10, 4 (2023), 3215–3230. DOI: https://doi.org/10.1109/JIOT.2022.3143529

[5] Xianfu Chen, Honggang Zhang, Celimuge Wu, Shiwen Mao, Yusheng Ji, and Mehdi Bennis. 2018. Performance optimization in mobile-edge computing via deep reinforcement learning. In *Proceedings of the 2018 IEEE 88th Vehicular Technology Conference (VTC-Fall)*. IEEE, 1–6.

[6] Yali Chen, Bo Ai, Yong Niu, Zhangdui Zhong, and Zhu Han. 2020. Energy efficient resource allocation and computation offloading in millimeter-wave based fog radio access networks. In *Proceedings of the ICC 2020-2020 IEEE International Conference on Communications (ICC)*. IEEE, 1–7.

[7] Zhao Chen and Xiaodong Wang. 2020. Decentralized computation offloading for multi-user mobile edge computing: A deep reinforcement learning approach. *EURASIP Journal on Wireless Communications and Networking* 2020, 1 (2020), 1–21.

[8] Kang Cheng, Yinglei Teng, Weiqi Sun, An Liu, and Xianbin Wang. 2018. Energy-efficient joint offloading and wireless resource allocation strategy in multi-MEC server systems. In *Proceedings of the 2018 IEEE international conference on communications (ICC)*. IEEE, 1–6.

[9] Yuliang Cong, Maiou Liu, Cong Wang, Shuxian Sun, Fengye Hu, Zhan Liu, and Chaoying Wang. 2024. Task scheduling and power allocation in multi-user multi-server vehicular networks by NOMA and deep reinforcement learning. *IEEE Internet of Things Journal* 11, 13 (2024), 23532–23543.

[10] Xiaohan Ding, Xiangyu Zhang, Ningning Ma, Jungong Han, Guiguang Ding, and Jian Sun. 2021. Repvgg: Making vgg-style convnets great again. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 13733–13742.

[11] Shouki A. Ebad. 2023. Quantifying IoT security parameters: An assessment framework. *IEEE Access* 11 (2023), 101087–101097. DOI: https://doi.org/10.1109/ACCESS.2023.3313975

[12] Guangsheng Feng, Xin Li, Zihan Gao, Chengbo Wang, Hongwu Lv, and Qian Zhao. 2021. Multi-path and Multi-hop task offloading in mobile Ad Hoc networks. *IEEE Transactions on Vehicular Technology* 70, 6 (2021), 5347–5361. DOI: https://doi.org/10.1109/TVT.2021.3077691

[13] Honghao Gao, Wanqiu Huang, Tong Liu, Yuyu Yin, and Youhuizi Li. 2022. PPO2: Location privacy-oriented task offloading to edge computing using reinforcement learning for intelligent autonomous transport systems. *IEEE Transactions on Intelligent Transportation Systems* 24, 7 (2022), 7599–7612.

[14] Hongzhi Guo, Weifeng Huang, Jiajia Liu, and Yutao Wang. 2021. Inter-server collaborative federated learning for ultra-dense edge computing. *IEEE Transactions on Wireless Communications* 21, 7 (2021), 5191–5203.

[15] Bulbul Gupta, Pooja Mittal, and Tabish Mufti. 2021. A review on amazon web service (aws), microsoft azure & google cloud platform (gcp) services. In *Proceedings of the 2nd International Conference on ICT for Digital, Smart, and Sustainable Development, ICIDSSD 2020, 27-28 February 2020, Jamia Hamdard, New Delhi, India.*

[16] Xiaofan He, Richeng Jin, and Huaiyu Dai. 2022. Multi-hop task offloading with on-the-fly computation for multi-uav remote edge computing. *IEEE Transactions on Communications* 70, 2 (2022), 1332–1344. DOI: https://doi.org/10.1109/TCOMM.2021.3129902

[17] Wei Hua, Peng Liu, and Linyu Huang. 2023. Energy-efficient resource allocation for heterogeneous edge-cloud computing. *IEEE Internet of Things Journal* 11 (2023).

[18] Forrest N. Iandola, Song Han, Matthew W. Moskewicz, Khalid Ashraf, William J. Dally, and Kurt Keutzer. 2016. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and< 0.5 MB model size. arXiv:1602.07360. Retrieved from https://arxiv.org/abs/1602.07360. (2016).

[19] Nafis Irtija, Iraklis Anagnostopoulos, Georgios Zervakis, Eirini Eleni Tsiropoulou, Hussam Amrouch, and Jörg Henkel. 2021. Energy efficient edge computing enabled by satisfaction games and approximate computing. *IEEE Transactions on Green Communications and Networking* 6, 1 (2021), 281–294.

[20] Hongbo Jiang, Xingxia Dai, Zhu Xiao, and Arun Iyengar. 2023. Joint task offloading and resource allocation for energy-constrained mobile edge computing. *IEEE Transactions on Mobile Computing* 22, 7 (2023), 4000–4015. DOI: https://doi.org/10.1109/TMC.2022.3150432

[21] Mohamed Khadmaoui-Bichouna, Gelayol Golcarenarenji, Ignacio Martinez-Alpiste, and Jose M. Alcaraz Calero. 2022. Edge computational offloading for corrosion inspection in industry 4.0. *IEEE Internet of Things Magazine* 5, 4 (2022), 116–120. DOI: https://doi.org/10.1109/IOTM.001.2200203

[22] Teven Le Scao, Angela Fan, Christopher Akiki, Ellie Pavlick, Suzana Ilić, Daniel Hesslow, Roman Castagné, Alexandra Sasha Luccioni, François Yvon, Matthias Gallé, et al. 2022. Bloom: A 176b-parameter open-access multilingual language model. (2022).

[23] Linqian Li, Yong Niu, Shiwen Mao, Bo Ai, Zhangdui Zhong, Ning Wang, and Yali Chen. 2022. Resource allocation and computation offloading in a millimeter-wave train-ground network. *IEEE Transactions on Vehicular Technology* 71, 10 (2022), 10615–10630.

[24] Peisong Li, Ziren Xiao, Xinheng Wang, Kaizhu Huang, Yi Huang, and Honghao Gao. 2023. EPtask: Deep reinforcement learning based energy-efficient and priority-aware task scheduling for dynamic vehicular edge computing. *IEEE Transactions on Intelligent Vehicles* 9, 1 (2023), 1830–1846.

[25] Lei Liu, Jie Feng, Xuanyu Mu, Qingqi Pei, Dapeng Lan, and Ming Xiao. 2023. Asynchronous deep reinforcement learning for collaborative task computing and on-demand resource allocation in vehicular edge computing. *IEEE Transactions on Intelligent Transportation Systems* 24, 12 (2023), 15513–15526.

[26] Lei Liu, Ming Zhao, Miao Yu, Mian Ahmad Jan, Dapeng Lan, and Amirhosein Taherkordi. 2023. Mobility-aware multi-hop task offloading for autonomous driving in vehicular edge computing and networks. *IEEE Transactions on Intelligent Transportation Systems* 24, 2 (2023), 2169–2182. DOI: https://doi.org/10.1109/TITS.2022.3142566

[27] Qiang Liu, Siqi Huang, Johnson Opadere, and Tao Han. 2018. An edge network orchestrator for mobile augmented reality. In *Proceedings of the IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*. 756–764. DOI: https://doi.org/10.1109/INFOCOM.2018.8486241

[28] Youshui Lu, Xiaojun Tang, Lei Liu, F. Richard Yu, and Schahram Dustdar. 2023. Speeding at the edge: An efficient and secure redactable blockchain for IoT-based smart grid systems. *IEEE Internet of Things Journal* 10, 14 (2023), 12886-12897.

[29] Yaser Mansouri and M. Ali Babar. 2021. A review of edge computing: Features and resource virtualization. *Journal of Parallel and Distributed Computing* 150 (2021), 155–183.

[30] Motahare Mounesan, Xiaojie Zhang, and Saptarshi Debroy. 2024. Edgerl: Reinforcement learning-driven deep learning model inference optimization at edge. In *Proceedings of the 2024 20th International Conference on Network and Service Management (CNSM)*. IEEE, 1–5.

[31] Motahare Mounesan, Xiaojie Zhang, and Saptarshi Debroy. 2025. Infer-EDGE: Dynamic DNN inference optimization in'just-in-time'Edge-AI implementations. arXiv:2501.18842. Retrieved from https://arxiv.org/abs/2501.18842. (2025).

[32] Devashree R. Patrikar and Mayur Rajaram Parate. 2022. Anomaly detection using edge computing in video surveillance system. *International Journal of Multimedia Information Retrieval* 11, 2 (2022), 85–110.

[33] Sita Rani, Pankaj Bhambri, Aman Kataria, Alex Khang, and Arun Kumar Sivaraman. 2023. *Big Data, Cloud Computing and IoT: Tools and Applications*. CRC Press.

[34] Abdeladim Sadiki, Jamal Bentahar, Rachida Dssouli, Abdeslam En-Nouaary, and Hadi Otrok. 2023. Deep reinforcement learning for the computation offloading in MIMO-based Edge Computing. *Ad Hoc Networks* 141 (2023), 103080.

[35] Yuvraj Sahni, Jiannong Cao, Lei Yang, and Yusheng Ji. 2021. Multi-hop multi-task partial computation offloading in collaborative edge computing. *IEEE Transactions on Parallel and Distributed Systems* 32, 5 (2021), 1133–1145. DOI: https://doi.org/10.1109/TPDS.2020.3042224

[36] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. (2017). arXiv:1707.06347. Retrieved from http://arxiv.org/abs/1707.06347

[37] Piyush Subedi, Jianwei Hao, In Kee Kim, and Lakshmish Ramaswamy. 2021. AI multi-tenancy on edge: Concurrent deep learning model executions and dynamic model placements on edge devices. In *Proceedings of the 2021 IEEE 14th International Conference on Cloud Computing (CLOUD)*. IEEE, 31–42.

[38] Yukun Sun and Xing Zhang. 2022. A2C learning for tasks segmentation with cooperative computing in edge computing networks. In *Proceedings of the GLOBECOM 2022-2022 IEEE Global Communications Conference*. IEEE, 2236–2241.

[39] Yuechuan Tao, Jing Qiu, and Shuying Lai. 2021. A hybrid cloud and edge control strategy for demand responses using deep reinforcement learning and transfer learning. *IEEE Transactions on Cloud Computing* 10, 1 (2021), 56–71.

[40] Shiyuan Tong, Yun Liu, Jelena Mišić, Xiaolin Chang, Zhenjiang Zhang, and Chunyan Wang. 2023. Joint task offloading and resource allocation for fog-based intelligent transportation systems: A UAV-enabled multi-hop collaboration paradigm. *IEEE Transactions on Intelligent Transportation Systems* 24, 11 (2023), 12933–12948. DOI: https://doi.org/10.1109/TITS.2022.3163804

[41] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023. Llama: Open and efficient foundation language models. arXiv:2302.13971. Retrieved from https://arxiv.org/abs/2302.13971. (2023).

[42] Tuyen X. Tran and Dario Pompili. 2019. Joint task offloading and resource allocation for multi-server mobile-edge computing networks. *IEEE Transactions on Vehicular Technology* 68, 1 (2019), 856–868. DOI: https://doi.org/10.1109/TVT.2018.2881191

[43] Anthony Triche, Anthony S. Maida, and Ashok Kumar. 2022. Exploration in neo-Hebbian reinforcement learning: Computational approaches to the exploration–exploitation balance with bio-inspired neural networks. *Neural Networks* 151 (2022), 16–33.

[44] Feng Wang, Jie Xu, Xin Wang, and Shuguang Cui. 2017. Joint offloading and computing optimization in wireless powered mobile-edge computing systems. *IEEE Transactions on Wireless Communications* 17, 3 (2017), 1784–1797.

[45] Jin Wang, Jia Hu, Geyong Min, Wenhan Zhan, Qiang Ni, and Nektarios Georgalas. 2019. Computation offloading in multi-access edge computing using a deep sequential model based on reinforcement learning. *IEEE Communications Magazine* 57, 5 (2019), 64–69.

[46] Xiaofei Wang, Yiwen Han, Chenyang Wang, Qiyang Zhao, Xu Chen, and Min Chen. 2019. In-edge ai: Intelligentizing mobile edge computing, caching and communication by federated learning. *Ieee Network* 33, 5 (2019), 156–165.

[47] Yun-Cheng Wang, Jintang Xue, Chengwei Wei, and C.-C. Jay Kuo. 2023. An overview on generative ai at scale with edge-cloud computing. *IEEE Open Journal of the Communications Society* 4 (2023), 2952–2971.

[48] P William, D Rajani, Manish Gupta, Resham Taluja, Ahmed Hussien Radie Alawadi, and Dinesh Kumar Yadav. 2023. Edge computing based traffic control management for distributed environment. In *Proceedings of the 2023 World Conference on Communication & Computing (WCONF)*. IEEE, 1–6.

[49] Zifeng Wu, Chunhua Shen, and Anton Van Den Hengel. 2019. Wider or deeper: Revisiting the resnet model for visual recognition. *Pattern Recognition* 90 (2019), 119–133.

[50] Jianjie Yang, Yingyang Chen, Zhijian Lin, Daxin Tian, and Pingping Chen. 2023. Distributed computation offloading in autonomous driving vehicular networks: A stochastic geometry approach. *IEEE Transactions on Intelligent Vehicles* 9, 1 (2023), 2701–2713.

[51] Shuai Yu, Xu Chen, Zhi Zhou, Xiaowen Gong, and Di Wu. 2020. When deep reinforcement learning meets federated learning: Intelligent multitimescale resource management for multiaccess edge computing in 5G ultradense network. *IEEE Internet of Things Journal* 8, 4 (2020), 2238–2251.

[52] Pengjie Zeng, Anfeng Liu, Chunsheng Zhu, Tian Wang, and Shaobo Zhang. 2022. Trust-based multi-agent imitation learning for green edge computing in smart cities. *IEEE Transactions on Green Communications and Networking* 6, 3 (2022), 1635–1648. DOI: https://doi.org/10.1109/TGCN.2022.3172367

[53] Hangyu Zhang, Rongke Liu, Aryan Kaushik, and Xiangqiang Gao. 2023. Satellite edge computing with collaborative computation offloading: An intelligent deep deterministic policy gradient approach. *IEEE Internet of Things Journal* 10, 10 (2023), 9092–9107.

[54] Xiaojie Zhang and Saptarshi Debroy. 2023. Resource management in mobile edge computing: A comprehensive survey. *ACM Computing Surveys* 55, 13s, Article 291 (jul 2023), 37 pages. DOI: https://doi.org/10.1145/3589639

[55] Xiaojie Zhang, Saptarshi Debroy, Peng Wang, and Keqin Li. 2025. DeepRB: Deep resource broker based on clustered federated learning for edge video analytics. *IEEE Transactions on Network and Service Management* 22, 4 (2025), 3325–3340.

[56] Xiaojie Zhang, Motahare Mounesan, and Saptarshi Debroy. 2023. EFFECT-DNN: Energy-efficient edge framework for real-time DNN inference. In *Proceedings of the 2023 IEEE 24th International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM)*. 10–20. DOI: https://doi.org/10.1109/WoWMoM57956.2023.00015

[57] Xiaojie Zhang, Amitangshu Pal, and Saptarshi Debroy. 2024. Edgeurb: Edge-driven unified resource broker for real-time video analytics. In *Proceedings of the NOMS 2024-2024 IEEE Network Operations and Management Symposium*. IEEE, 1–8.

[58] Yu Zhang, Yanmin Gong, and Yuanxiong Guo. 2024. Energy-efficient resource management for multi-uav-enabled mobile edge computing. *IEEE Transactions on Vehicular Technology* 73, 8 (2024), 12026–12037. DOI: https://doi.org/10.1109/TVT.2024.3379298

[59] Nan Zhao, Ying-Chang Liang, Dusit Niyato, Yiyang Pei, Minghu Wu, and Yunhao Jiang. 2019. Deep reinforcement learning for user association and resource allocation in heterogeneous cellular networks. *IEEE Transactions on Wireless Communications* 18, 11 (2019), 5141–5152.

Received 2 November 2024; revised 3 May 2025; accepted 7 October 2025