Geometrized Task Scheduling and Adaptive Resource Allocation for Large-Scale Edge Computing in Smart Cities

Yang Chen, Yuemin Ding, Zhen-Zhong Hu, Zhengru Ren

***Abstract*—Edge computing is vital in developing smart cities by providing on-site computational resources to support the surging Internet of Things demands. However, the distributed nature of edge nodes and large scale of tasks distributed in expansive urban spaces challenge task scheduling and resource allocation. In this paper, a novel framework is developed to achieve efficient task scheduling (assignment and offloading) and resource allocation for large-scale edge computing in both wired and wireless smart-city applications. To overcome overparameterization in existing optimization-based heuristic algorithms, the geometrized task scheduling problem is addressed by transforming the assignment of clustered tasks into a regional partition problem in a two-dimensional graph and applying a Tetris-like task offloading strategy for edge-cloud cooperation. These approaches avoid combinatorial explosion and NP-hardness, and the regional partition problem is solved by multiplicative weighted Voronoi diagrams with polynomial computational complexity. Furthermore, an adaptive resource allocation algorithm is proposed to overcome the dynamic, uncertain, and highly concurrent task requests. An online learning algorithm is adopted to adjust the sliding window length according to the evolving conditions. Comparison results show that the proposed framework significantly reduces the average task deadline violation rate, i.e., up to 4.72% of (more than 20 times better than) those using the other schemes, especially when handling large-scale workloads.**

***Index Terms*—Smart cities; Edge computing; Task scheduling; Task assignment; Resource allocation**

I. Introduction

With the proliferation of Internet of Things (IoT) devices, a tremendous amount of data is supposed to be transmitted to data centers for analysis and processing. According to Statista [1], the number of global IoT devices is projected to reach 3.21 billion by 2030. Given the limited processing capabilities of IoT devices, it's necessary to offload tasks from these devices to satisfy the latency requirements for emerging applications in smart cities, such as intelligent transportation management, digital twin technology, intelligent manufacturing, etc. The tasks could be data preprocessing, artificial intelligence model training, target and failure recognition, or digital-twin model construction. They not only require massive computation resources to handle complicated data processing but also require low-latency communications. A typical solution is to offload these tasks to cloud platforms to compensate for the limited local computational capacity at terminal devices and arousing the issue of long access time. Nevertheless, the explosive growth of data volume generated by IoT applications imposes a heavy burden on cloud services and networks, leading to an inevitable decline in overall system performance [2], [3]. Cloud computing infrastructure can not be deployed widely, due to the high construction costs and relatively low flexibility. Hence, local terminal devices are generally located far from the cloud computing platforms, the resulting transmission delays make it difficult to support delay-sensitive tasks [4], [5].

Edge computing [6] is proposed as a complement and extension to cloud computing aiming to mitigate such limitations. By situating a certain number of edge nodes (ENs) at the network edge, edge computing provides the necessary computing, storage, and network resources for the resource-limited devices [7]. The ENs are typically base stations, but may also include other small servers equipped with computational resources. As these ENs are close to local terminal devices [8], the latency in network communications between servers and devices can be effectively reduced. Furthermore, it can effectively alleviate the communication and computation burden on cloud data centers, providing higher quality and efficiency services than traditional cloud computing [9]. Boasting the advantages of low latency and high bandwidth, edge computing effectively provides support for latency-sensitive compute-intensive tasks [10], [11].

However, the shifting from centralized to distributed computing in smart city scenarios introduces a host of challenges. Smart cities cover extensive geographic areas [12], meaning that ENs are often dispersed over a large geographical span, creating a complex and vast network. The multitude of IoT devices generates a massive volume of data and consequent computational tasks, necessitating the selection of the most suitable ENs or cloud platforms for task scheduling, namely large-scale task scheduling problems.

With the increase in both the number of tasks and the scale of nodes, task scheduling involves large-scale task assignment and offloading decisions [13]. Recent studies on edge-cloud collaboration have made significant progress in communication aspects like hybrid NOMA-FDMA transmission schemes [14] and energy efficiency optimization. However, these approaches mainly focus on communication-level optimization while the fundamental challenge of large-scale task scheduling remains unsolved. The high complexity of task scheduling makes traditional optimization methods impractical for large-scale scenarios involving hundreds of ENs and thousands of concurrent tasks. This calls for a new perspective to address

Institute for Ocean Engineering, Shenzhen International Graduate School, Tsinghua University, Tsinghua Campus, University Town, Shenzhen 518055, China (e-mail: zhengru.ren@sz.tsinghua.edu.cn).

the scheduling problem.

The problem becomes even more complicated due to the need to minimize communication overhead and ensure quality of service (QoS). Existing heuristic algorithms lead to overparameterization when dealing with such large-scale task scheduling problem [15], while reinforcement learning approaches struggle with the high-dimensional and complex state and involved action spaces [16]–[19]. Although recent advancements in large-scale convex optimization techniques [13], [20] provide robust tools for computational resource allocation, they are not adequately equipped to directly address the challenges associated with non-convex computation offloading. Therefore, there is an urgent need for an algorithm with low overhead and complexity to effectively overcome the challenges of large-scale task scheduling.

Edge computing architecture is inherently distributed and heterogeneous, and it implies the necessity to manage a large number of ENs with significant differences in memory size, I/O throughput, and CPU capacities [21], [22]. Efficiently allocating and coordinating the substantial distributed computing resources at the edge sides is challenging. State-of-the-art cloud computing resource allocation strategies commonly adopt parallel processing models of virtual machines (VMs), where cloud platforms can instantiate numerous VMs to process many tasks in parallel due to their abundant computational resources. In contrast, ENs possess limited resources and thus can only support a restricted number of VMs. Therefore, it is commonly assumed that tasks follow a specific distribution (such as Poisson distribution) to manage the queuing process and to estimate the queuing time for tasks [23], [24]. However, task requests often do not conform to a specific distribution, and show a great degree of variability in their arrival process, making accurate predictions and planning challenging.

Docker [25] container technology offers millisecond-level boot times and a lightweight deployment paradigm, making it more apt for edge computing scenarios. However, the efficient resource allocation for containers faces challenges in two aspects, i.e., the container configuration approaches that augment the efficiency of resource utilization [26], [27] and the resource allocation schemes that adapt to random and highly concurrent task requests. Most existing studies overlook the stochastic nature and high concurrency of task requests, focusing primarily on reducing task latency without considering the assurance of task completion before deadlines.

To address the challenges mentioned above, the present research proposes a framework customized to applications of edge computing with expansive geographic areas and massive networks, such as smart cities, aimed at optimizing large-scale task scheduling while ensuring tasks are completed before their deadlines. The proposed framework innovatively transforms the task assignment problem into a two-dimensional (2D) geometric regional partitioning problem (RPP) using multiplicative weighted Voronoi diagrams, while simultaneously integrating density-based and prototype-based stream clustering to reduce the scale of dynamic task streams. The geometrized task scheduling is accomplished through a combination of tasks and a Tetris-like offloading evaluation mechanism to address the resource limitation of ENs. Furthermore, the dynamic resource allocation is formulated as a sliding window task segmentation problem with adaptive window length. The framework offers an effective and reliable solution for the scheduling of large-scale tasks in smart cities.

The main contributions of this paper are summarized as follows:

- A novel approach is proposed that transforms large-scale task scheduling into a geometric regional partition problem. The approach avoids the combinatorial explosion inherent in traditional approaches and reduces computational complexity from exponential to polynomial time. The innovative application of multiplicative weighted Voronoi diagrams provides a unique solution for geometric partitioning, enabling efficient geometrized task assignment.
- Compatible with the proposed geometrized task assignment strategy, an integrated clustering and Tetris-like task offloading evaluation method is introduced to address large-scale dynamic task processing and resource limitations of the ENs. The geometry-based offloading algorithm facilitates dynamic resource allocation and enables effective adaptation to massive, random, and concurrent task requests by leveraging cloud-edge collaboration advantages, while considering the cases where tasks have been partially processed.
- An adaptive resource allocation algorithm is proposed by transforming the complex edge problem into a task partitioning problem. The sliding time window length is adaptive to accommodate dynamic task requests and resource demands. The method remarkably reduces the computational overhead and fully leveraging the lightweight characteristics of containerization technology.

The rest of this paper is structured as follows. Section II provides detailed information on the system model and formulations. Section III briefly summarizes the frame design. Sections IV and V describe the proposed geometrized task scheduling and adaptive resource allocation algirithms in details, respectively. Section VI evaluates the performance of the proposed framework. Finally, Section VII concludes the paper.

II. System modeling and problem formulation

A. System description

Figure 1 presents a smart city system architecture composed of three layers: 1) cloud layer; 2) edge layer; and 3) device layer. The device layer consists of a series of IoT devices $\mathcal{N} = \{1, 2, \cdots, N\}$ with limited computational capacity, where N is the total number of IoT devices. The edge layer contains a cluster of ENs $\mathcal{J} = \{1, 2, \cdots, J\}$ with limited computing power situated in the vicinity of the IoT devices, where J is the total number of ENs. The cloud layer encompasses a cloud platform c, distanced from the IoT devices and ENs, providing computing, storage, and network resources to ENs that are resource-constrained. Tasks are processed either at an EN or in a cloud platform. Container images are employed to create virtualized environments. The cloud platform retains all necessary container images for computational tasks, while ENs contain only a selection of commonly used images. If an

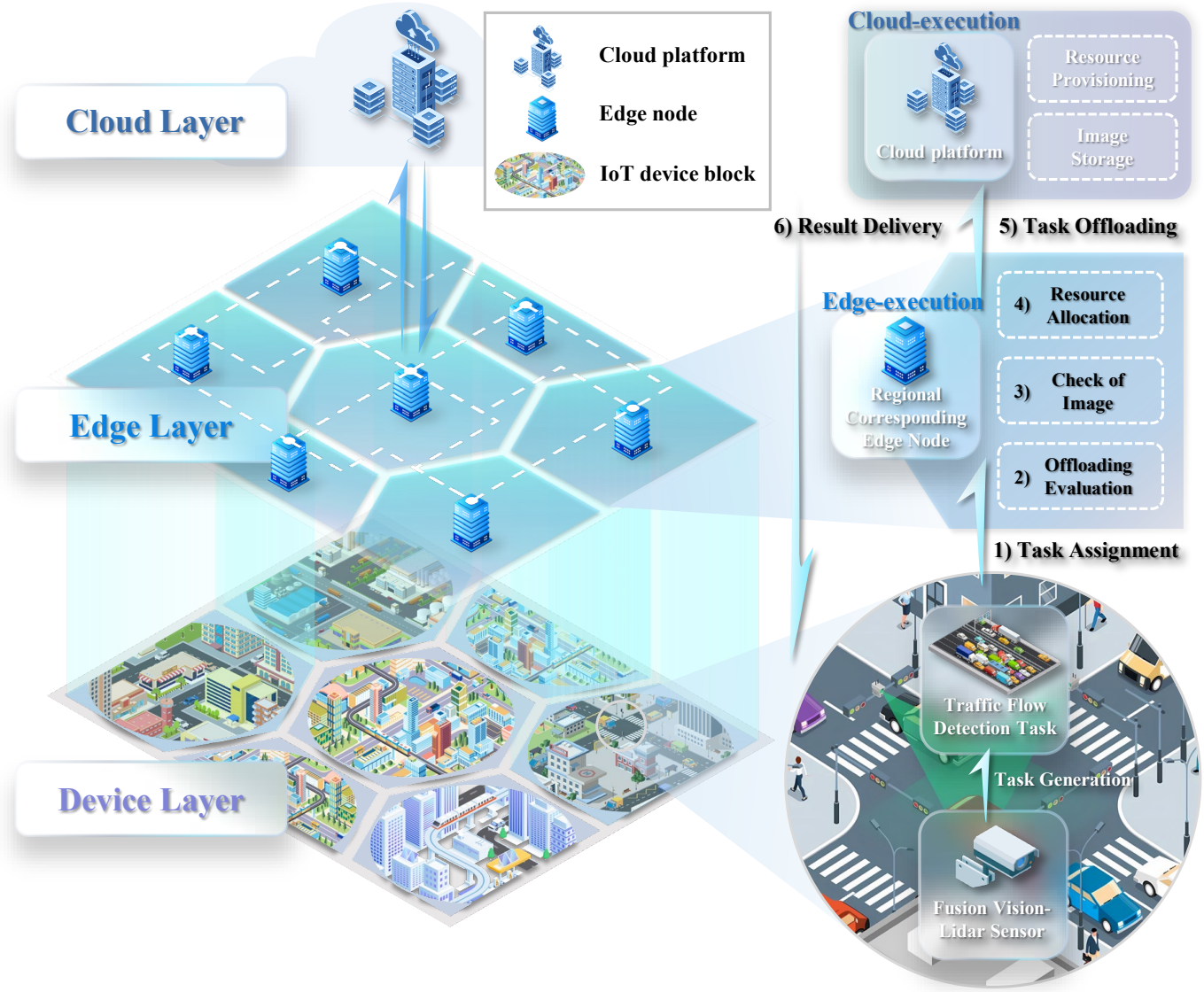


Fig. 1. An illustration of the smart city architecture.

EN lacks the required container image to process a specific task, it should download it from the cloud.

The system's operation extends indefinitely, structured into equal time slots indexed by $t = 0, 1, 2, \ldots$. The length of each time slot is represented by a small constant τ, during which the positions of the IoT devices are assumed to be static. If the heights are neglected and we only consider a 2D problem, let the vector $\mathrm{p}_{n,t} \in \mathbb{R}^2$ represent the location of an IoT device in time slot t. Similarly, the location of EN $j \in \mathcal{J}$ is indicated by the vector $\mathrm{q}_j \in \mathbb{R}^2$. An IoT devices $n \in \mathcal{N}$ generate a dynamic task stream $\mathcal{I} = \{1, 2, \cdots, I\}$ in a given period, where I is the total number of tasks. The creation of these tasks is often highly concurrent and the total number could be exceedingly large. The time required for sending computational task requests and receiving the computed results at the IoT devices is defined as the task completion time, the process during this period is referred to as the computational task completion process. The computational tasks generated by IoT devices could be represented by the metadata

$$\mathcal{M}_i \triangleq < \gamma_i, \varepsilon_i, \iota_i, t_i, \zeta_i, l_i >, \tag{1}$$

where γ_i denotes the volume of data to be uploaded and processed for the task (in bits), and ε_i indicates the number of instructions required per bit of data for task i in MIPS. It can be determined through offline methods or estimated based on historical operational data [28]. Thus, the total number of instructions required to complete the task i can be represented as $\gamma_i \varepsilon_i$. ι_i represents the size of the container image required to process task i. t_i records the time slot when the task is generated and the request will be sent to the EN. ζ_i records the deadline for each task computational results received after this deadline are considered invalid for the IoT devices. l_i represents the location of the IoT device at time slot t_i, i.e., $l_i = l_{n,t_i}$.

Assuming full-duplex operation of ENs, where task uploading and result downloading proceed concurrently, the transmis-

sion technology differs for wired and wireless connections. For wired connections, the uplink and downlink transmissions utilize Ethernet technology, which allows for high-speed, reliable, and full-duplex communication. For wireless connections, the uplink transmission employs advanced multiple-access technologies to optimize channel use and boost upload speeds. Given that the data size of the computational results returned is significantly smaller than that of the uploaded tasks, the downlink in wireless connections typically uses orthogonal frequency division multiple access technology or similar schemes to send the computational results back to IoT devices [29]. Figure 1 illustrates the entire computational task completion process, comprising six stages, i.e,

1) **Task Assignment.** Tasks are assigned to ENs corresponding to the subregions delineated for the device layer and the locations of task generation. IoT devices send task metadata to an EN.
2) **Offloading Evaluation.** The EN initially assesses the tasks to determine its own ability to accept the tasks based on available computational resources.
3) **Check of Image.** If the EN can accept the tasks, it checks for the requisite container image to process the tasks; if absent, the image is downloaded from the cloud platform.
4) **Task Offloading.** If the EN is unable to process the tasks, the tasks are offloaded to the cloud platform.
5) **Resource Allocation.** The EN or cloud platform utilizes the image to create a containerized virtual environment for the tasks and processes them.
6) **Result Delivery.** The computed results are returned from the EN or cloud platform to the IoT devices.

B. System modeling

1) Edge execution: Connections between IoT devices and ENs could take various forms, including wireless and wired connections. A binary variable $a_{n,j}$ is introduced to differentiate between connection types, representing the connection type between IoT device n and EN j. When device n connects to node j via a wired connection, $a_{n,j} = 1$. When device n connects to node j via a wireless connection, $a_{n,j} = 0$.

For wireless connections, practical factors such as path loss and randomness are considered. The path loss is modeled as $d_{n,j}^{-v}$, where v denotes the path loss exponent [30], and $d_{n,j}$ represents the distance between IoT device n and EN j during the task request transmission, expressed as

$$d_{n,j} = \|l_{n,t_i} - l_j\|_2. \tag{2}$$

For wired connections, the transmission rate is assumed to be a constant $w_{n,j}^{(wired)}$, with its value depending on the network settings and hardware specifications.

Considering both wireless and wired connections, the channel transmission rate $w_{n,j}$ between IoT device n and EN j is represented as

$$w_{n,j} = a_{n,j} w_{n,j}^{(wired)} + (1 - a_{n,j}) B \log_2 \left(1 + \frac{P_n |h|^2 d_{n,j}^{-v}}{\sigma_0^2}\right), \tag{3}$$

where B denotes the bandwidth of the wireless connection, P_n represents the transmission power of the IoT device, h is the channel fading coefficient of the uplink, and σ_0^2 represents the power of Gaussian white noise.

Given that computational results are typically small data packets, such as control signals [4], the result delivery time could be negligible [31]. Hence, only the upload time of the data γ_i is considered. Consequently, the transmission time of task i at the EN j could be calculated by

$$T_{pos}(i,j) = \frac{\gamma_i}{w_{n,j}}. \tag{4}$$

When processing a specific task, if the EN j does not contain the corresponding container image for the task i, it should be downloaded from the cloud platform. The image download time is given by

$$T_{img}(i,j) = \begin{cases} \dfrac{\iota_i}{w_{c,j}}, & \text{if image not stored in EN } j \\ 0, & \text{otherwise} \end{cases}, \tag{5}$$

where $w_{c,j}$ represents the download throughput of the downlink between the EN j and the cloud platform.

For container-based virtualization, the computation time for task i is determined by the number of instructions required to complete the task and the computational resources that the EN can allocate to the task from the arrival time slot t_i when the task is sent to the deadline time slot ζ_i. The computation time is given by

$$T_{comp}(i,j) = \frac{\gamma_i \varepsilon_i}{\sum_{t=t_i}^{\zeta_i} r_{i,j}(t)}, \tag{6}$$

where $r_{i,j}(t)$ denotes the computational resources allocated by the EN j to the task i at time slot t.

Moreover, the computational task completion process also includes the waiting time for tasks at ENs, denoted as $T_{wait}(i,j)$. When tasks are in a non-active state, a waiting time is required. For instance, when virtual containers are prepared but not yet initiated, or when they are pending the availability of processing resources. Early studies often assume that the task requests follow a specific distribution, such as the Poisson distribution, allowing this component of time to be simplified into the task's queuing time. However, task requests are typically random and highly concurrent in practice, leading to a situation where the waiting time may constitute a significant portion of the total task completion time. Oversimplification as a specific distribution may result in very large errors. Therefore, for random and highly concurrent task requests, it is necessary to consider the waiting time $T_{wait}(i,j)$ with greater precision.

To summarize, the total execution time for the task i within the EN j consists of the data transfer time, image download time, computation time, and the waiting time, i.e.,

$$T(i,j) = T_{pos}(i,j) + T_{img}(i,j) + T_{comp}(i,j) + T_{wait}(i,j). \tag{7}$$

2) Cloud execution: Similarly, ignoring the time taken for the return of computed results, the transmission time is considered solely as the upload time from the EN j to the cloud platform for the pending data in the task, i.e.,

$$T_{pos}(j,c) = \frac{\gamma_i}{w_{j,c}}, \tag{8}$$

where $w_{j,c}$ represents the uplink transmission rate between the EN and the cloud platform. The cloud platform stores all necessary container images for task processing, thus it can directly invoke the relevant images to create a containerized virtual environment, which takes negligible time and can be discounted.

Assuming that the computational resource of the cloud platform is sufficient, the processing time for a task i on the cloud is given by

$$T_{comp}(i,c) = \frac{\varepsilon_i \gamma_i}{f_c}, \tag{9}$$

where f_c represents the cloud platform's data processing capability, i.e., the number of instructions per second.

It is important to note that even if a task needs to be offloaded for processing to the cloud platform, the data to be processed should first be uploaded to the EN and then transferred to the cloud platform according to the architecture used in the present paper. Hence, the total time taken for offloading a task to the cloud platform can be expressed as

$$T(i,c) = T_{pos}(i,j) + T_{pos}(j,c) + T_{comp}(i,c) \tag{10}$$

3) Total processing time: Based on the summarized processing times from edge and cloud models, the total processing time for a task can be represented as

$$T(i) = x_{i,j} T(i,j) + x_{i,c} T(i,c), \tag{11}$$

where the binary variables $x_{i,j}$ and $x_{i,c}$ indicate whether the task is executed at the EN j or on the cloud platform c, respectively. The variable is assigned a value of 1 if the task is executed at the specified location, and 0 otherwise. Hereafter, a task is assumed to be executed at only one location, either an EN or a cloud platform, it follows

$$\sum_{j \in \mathcal{J}} x_{i,j} + x_{i,c} = 1, \quad \forall i \in \mathcal{I} \tag{12}$$

C. Problem statement

Due to the widespread geographical span and dispersed distribution of ENs and IoT devices in smart cities, the challenges of large-scale task scheduling and resource allocation are encountered. The primary objective of large-scale task scheduling and resource allocation is to utilize the minimum possible resources of the EN cluster to fulfill a greater number of computational task requests, thereby maximizing the QoS. Given the strong correlation between QoS metrics and deadline violations, it becomes necessary to define the average task deadline violation rate as

$$\bar{\Theta} = \frac{1}{I} \sum_{i=1}^{I} \mathcal{V}(i), \tag{13}$$

where $\mathcal{V}$ denotes the deadline violation metric for tasks. For a given task request, if the completion time of the computational task T_i exceeds its deadline ζ_i, the task is considered to violate its deadline, and $\mathcal{V}(i)$ is computed as

$$\mathcal{V}(i) = \begin{cases} 0, & T(i) \le \zeta_i - t_i \\ 1, & T(i) > \zeta_i - t_i \end{cases}. \tag{14}$$

The average resource utilization rate of the ENs is introduced to evaluate the efficient resource utilization and computational capability of the ENs. It is defined as the ratio of total resources allocated to tasks to the total resources of the EN cluster, expressed as

$$\bar{\mathcal{U}} = \frac{\sum_{t=1}^{\mathcal{T}} \sum_{j=1}^{J} \sum_{i=1}^{I} r_{i,j}(t)}{\sum_{j=1}^{J} \mathcal{T} R_j^{(t_u)}}, \tag{15}$$

where $R_j^{(t_u)}$ represents the total computing resources of EN j, $\mathcal{T}$ is the total number of time slots. The computing resources occupied at EN j are subject to strict constraints. To ensure a timely and efficient reaction, at any time slot t, the sum of resources utilized by all tasks running on an EN should not exceed the total available resources to the EN, i.e.,

$$\sum_{i=1}^{I} x_{i,j} r_{i,j}(t) \le R_j^{(t_u)}, \quad \forall t, j \in \mathcal{J}. \tag{16}$$

Furthermore, the concept of average task offloading rate is introduced to leverage the abundant computing resources of the cloud platform and alleviate the computational pressure on the ENs. The average task offloading rate represents the proportion of tasks offloaded to the cloud platform for execution,i.e.,

$$\bar{\mathcal{O}} = \frac{\sum_{i=1}^{I} x_{i,c}}{\sum_{i=1}^{I} \left(\sum_{j=1}^{J} x_{i,j} + x_{i,c} \right)}. \tag{17}$$

An optimization problem is formulated to satisfy multiple objectives, aiming to strike a balance between ensuring service quality, efficient resource utilization, and reasonable offloading. These objectives require carefully designed constraints to achieve optimal system performance. The primary goal is to minimize the average task deadline violation rate $\bar{\Theta}$ to maintain the overall QoS. Additionally, to prevent edge resource underutilization while maintaining system stability, the aim is to fully utilize the computational capability of the EN cluster by keeping the average resource utilization rate $\bar{\mathcal{U}}$ above a certain threshold $\overline{\mathcal{U}}^*$. The average offloading rate constraint $|\bar{\mathcal{O}} - \overline{\mathcal{O}}^*| \le \epsilon$ plays a crucial role in system optimization, excessive offloading would cause network congestion and high communication costs, while insufficient offloading would overwhelm edge nodes with compute-intensive tasks, both leading to increased deadline violations. Moreover, the resource capacity constraint on each EN ensures operational stability by preventing processing overload at individual nodes.

Therefore, the main objective is to minimize the average task deadline violation rate while satisfying the constraints on the average resource utilization rate and average offloading

rate, as well as the resource capacity constraint. It could be formulated as

$$\begin{aligned} \min_{\mathbf{x}\in\mathcal{X}} \quad & \bar{\Theta} \\ \text{s.t.} \quad & \bar{\mathcal{U}} \geq \overline{\mathcal{U}}^* \\ & |\bar{\mathcal{O}} - \overline{\mathcal{O}}^*| \leq \epsilon \\ & \sum_{i=1}^{I} x_{i,j} r_{i,j}(t) \leq R_j^{(t_u)}, \quad \forall t, j \in \mathcal{J}, \end{aligned} \tag{18}$$

where $\mathcal{X} = \{x_{i,j}, x_{i,c} \mid i \in \mathcal{I}, j \in \mathcal{J}\}$ denotes the scheduling strategy for all tasks within the task stream $\mathcal{I}$, with the minimization of the average task deadline violation rate being the objective. $\overline{\mathcal{U}}^*$ is the expected minimum average resource utilization rate, $\overline{\mathcal{O}}^*$ is the expected average offloading rate, and ϵ is the allowed deviation range for the offloading rate.

Eq.(18) is an integer optimization problem that aims to find an optimal scheduling strategy from a set of possible combinations $\mathcal{X}$. The computational complexity of this problem scales exponentially with the number of tasks I and the number of ENs J, i.e., $O((J+1)^I)$, leading to a significant decrease in computational speed.

The highlights lie in employing algorithms with low computational complexity to address large-scale task scheduling problems and utilizing dynamic resource allocation strategies to efficiently handle unpredictable and concurrent task requests. A practical framework and several effective algorithms are introduced to tackle large-scale task scheduling and resource allocation issues in smart city contexts.

III. Framework overview

The framework design for optimizing task scheduling and dynamic resource allocation across extensive edge computing applications, organized through four integrated parts is shown in Figure 2. The first part is (a) integrated incremental streaming clustering, which addresses large-scale task scheduling by clustering tasks.

The second part, (b) a weighted Voronoi diagram for task assignment, and the third part, (c) Tetris-like task offloading evaluation, collectively complete the task scheduling by assigning tasks to an appropriate EN and assessing the feasibility of task offloading based on resource constraints, deadlines, and partial task processing. The fourth part dynamically manages tasks by merging the concepts of (d) deadline-aware task segmentation and (e) adaptive sliding window optimization for dynamic adaptive resource allocation, accommodating the unpredictability of tasks arrival while ensuring the completion of tasks.

IV. Geometrized task scheduling

A. Geometrization of the problem

IoT devices typically generate a number of compute-intensive tasks that require efficient allocation to heterogeneous ENs for execution in smart city scenarios. Specifically, given a task stream $\mathcal{I}$ and an EN cluster $\mathcal{J}$, heterogeneity among ENs means the execution time $T(i,j)$ for each task $i \in \mathcal{I}$ on different ENs $j \in \mathcal{J}$ varies. Moreover, tasks may also be allocated to the cloud platform c for execution. The objective is to identify an allocation scheme $\pi : \mathcal{I} \to \mathcal{J} \cup c$ that minimizes the average task deadline violation rate $\bar{\Theta}$, i.e.,

$$\min_{\mathbf{x}\in\mathcal{X}} \ \bar{\Theta} = \frac{1}{I} \sum_{i=1}^{I} \mathcal{V}(i). \tag{19}$$

The surge is due to the necessity to explore all possible scheduling strategies $\mathcal{X}$, whose combination count exponentially grows as the number of tasks and nodes increases. Furthermore, the heterogeneity of ENs exacerbates the severity of the combinatorial explosion.

To handle the NP-hard problem, an innovative approach that transforms the original issue into an RPP is proposed. The method involves dividing the entire area into J non-overlapping subregions, each corresponding to an EN $j \in \mathcal{J}$. Additionally, tasks can also be assigned to cloud platform c after evaluation. A task $i \in \mathcal{I}$ is allocated to the EN j that minimizes the comprehensive cost function $f(i,j)$, rather than directly determining the target EN for each task as in the original problem. The comprehensive cost function $f(i,j)$ considers three factors, i.e., task execution time $T(i)$, task transmission distance cost $d(i,j)$, and an abstract auxiliary function $g(i,j)$ representing optimization strategies such as task clustering and node resource availability, expressed as

$$f(i,j) = w_1 T(i) + w_2 d(i,j) + w_3 g(i,j), \tag{20}$$

where w_1, w_2, w_3 are the weighting coefficients for each component, respectively. The objective function of the RPP is to minimize the sum of the comprehensive costs for all tasks as

$$\min \sum_{i=1}^{I} \min_{j=1}^{J+1} f(i,j). \tag{21}$$

The advantage of the geometrization approach lies in avoiding the combinatorial explosion inherent in the original optimization problem, thereby reducing the problem's complexity to a polynomial level. For instance, in a scenario involving 100 tasks and 10 ENs, the optimal task assignment problem requires evaluating 10^{100} potential allocation schemes. In contrast, the RPP merely necessitates determining the geographic area scopes corresponding to the 10 ENs, significantly lowering the computational complexity. RPP can obtain a reliable approximate solution within polynomial time, with the worst-case performance upper bound being α times the optimal solution of the original problem, where α is a constant factor related to the optimization strategy.

The effectiveness of the RPP depends on determining reasonable regional partitioning schemes by tuning w_1 and w_2, as well as appropriately designing the auxiliary function $g(i,j)$ related to optimization strategies and tuning its weight w_3, further reducing the constant factor α and enhancing the approximation quality.

B. Integrated incremental streaming clustering

To design the optimization strategy's auxiliary function $g(i,j)$ and its weight w_3, the key lies in leveraging the specific

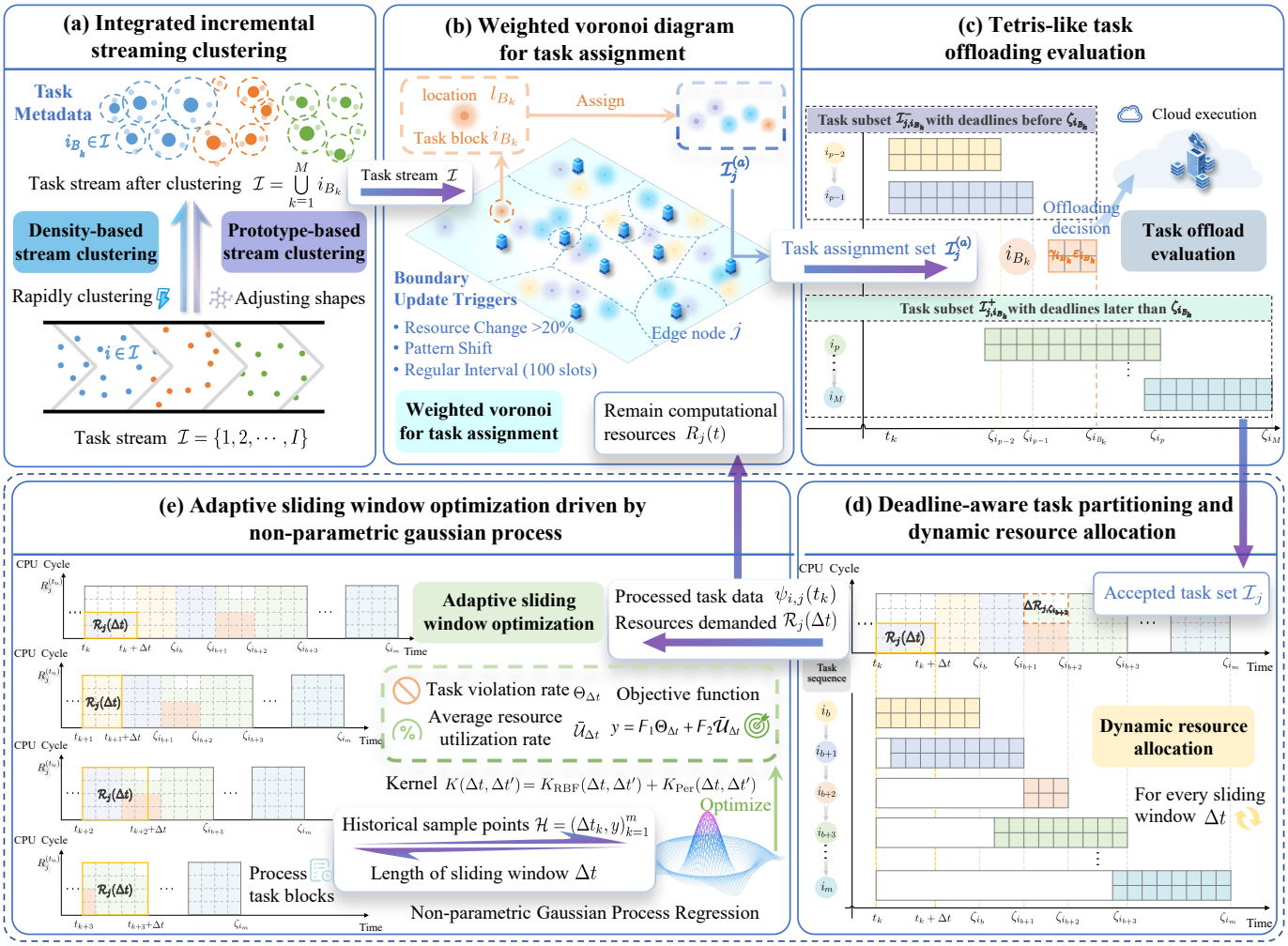


Fig. 2. Overview of the proposed framework.

characteristics of tasks to implement effective clustering strategies. Particularly for large-scale tasks, concentrating multiple smaller tasks into one larger task significantly reduces the overall number of variables. For a task streaming clustering problem, an incremental streaming clustering algorithm is adopted to decrease the number of variables by combining the density-based and prototype-based perspectives.

For a large-scale dynamic task stream $\mathcal{I}$, the feature vector of a task $i \in \mathcal{I}$ denoted by $(\zeta_i, \gamma_i \varepsilon_i, l_i)$ is defined through three core attributes, its deadline ζ_i, computational demand $\gamma_i \varepsilon_i$, and generation location l_i. It necessitates defining a similarity measure between tasks. For any two tasks i and i', their comprehensive distance $\mathcal{D}(i, i')$ is defined as the weighted Euclidean distance based on their three core attributes, expressed as

$$\mathcal{D}(i,i') = \sqrt{\omega_\zeta(\zeta_i-\zeta_{i'})^2 + \omega_\gamma(\gamma_i\varepsilon_i-\gamma_{i'}\varepsilon_{i'})^2 + \omega_l||l_i-l_{i'}||_2^2} \tag{22}$$

where ω_ζ, ω_γ, and ω_l are the weight coefficients for the three attributes deadline, computation workload, and location, respectively, and they satisfy $\omega_\zeta + \omega_\gamma + \omega_l = 1$.

1) Density-based stream clustering: The density-based stream clustering algorithm facilitates rapid identification of and response to core dense areas within task streams by dynamically processing data streams through the maintenance and updating of CF-Tree and CF-Kernel data structures, forming micro-clusters promptly. Each micro-cluster $\mathcal{P}_k$ maintains its centroid c_k, entry radius $r_r(\mathcal{P}_k)$, number of members $n_m(\mathcal{P}_k)$, and density $\phi(\mathcal{P}_k)$.

For a newly arrived task i, the algorithm first identifies the set of existing micro-clusters $\mathbb{P}_k$ that are density-reachable from task i that could be expressed as

$$\mathbb{P}_k = \left\{ \mathcal{P}_k \middle| \max_{p \in \mathcal{P}_k} \mathcal{D}(i,p) \le r_d \right\}, \tag{23}$$

where r_d is a predefined radius threshold. It then constructs a candidate micro-cluster $\mathcal{P}_i$ by including the union of points in all micro-clusters in $\mathbb{P}_i$ and the new task i itself as $\mathcal{P}_i = \mathbb{P}_k \cup i$. The density $\phi(\mathcal{P}_i)$ of this candidate micro-cluster is computed as

$$\phi(\mathcal{P}_i) = \frac{|\mathcal{P}_i|}{V(\mathcal{P}_i)}, \tag{24}$$

where $|\mathcal{P}_i|$ is the number of points in $\mathcal{P}_i$, and $V(\mathcal{P}_i)$ is the multi-dimensional volume of the space occupied by $\mathcal{P}_i$.

The candidate micro-cluster $\mathcal{P}_i$ is accepted as a new micro-cluster if $\phi(\mathcal{P}_i) \ge \phi_s$, where ϕ_s is a predefined density

threshold. In the case, the algorithm further checks if $\mathcal{P}_i$ can be merged with any existing micro-clusters $\mathcal{P}_k$ that satisfy

$$\mathcal{D}(c_i, c_k) < r_r(\mathcal{P}_i) + r_r(\mathcal{P}_k) + r_m, \tag{25}$$

where c_i, c_k are the centroids, r_m is a merge radius threshold, and $r_r(\mathcal{P}_k)$ is the entry radius of $\mathcal{P}_k$ defined as

$$r_r(\mathcal{P}_k) = \max_{p \in \mathcal{P}_k} \mathcal{D}(c_k, p). \tag{26}$$

The merged micro-cluster then replaces the separate micro-clusters $\mathcal{P}_k$ and $\mathcal{P}_{k'}$ in the set of micro-clusters maintained by the algorithm. Otherwise, if the density satisfies $\phi(\mathcal{P}_i) < \phi_s$, a new micro-cluster containing only i is created.

The algorithm dynamically maintains a set of micro-clusters, adding newly accepted dense micro-clusters, and merging existing ones based on the centroid distance condition, while removing older micro-clusters that no longer remain dense as new tasks arrive. It enables rapid detection of evolving dense regions within the stream.

2) Prototype-based stream clustering: The prototype-based stream clustering algorithm refines and optimizes the morphology of micro-clusters from a global perspective to reflect the evolution of dynamic task streams more accurately. It uses a hierarchical time window model with window sizes specified as 2^h time slots to track statistical changes across multiple time scales, where $h = 0, 1, \ldots$. For each micro-cluster $\mathcal{P}_k$ and time scale of 2^h time slots, the algorithm updates $n(\mathcal{P}_k)^{(2^h)}$ and $c_k^{(2^h)}$ to reflect the changes in stream statistics. At each time scale of 2^h time slots, the outlier factor $O_f(\mathcal{P}_k)$ for $\mathcal{P}_k$ is computed as

$$O_f(\mathcal{P}_k) = \sum_{p \in \mathcal{N}_n(\mathcal{P}_k)} \mathcal{D}(c_k, p). \tag{27}$$

The outlier factor $O_f(\mathcal{P}_k)$ indicates the distance between $\mathcal{P}_k$ and its neighborhood $\mathcal{N}_n(\mathcal{P}_k)$. The prototype-based stream clustering algorithm identifies micro-clusters with lower O_f at each 2^h, as they are closer to potential high-density centers. It then updates its statistics via

$$n_m(\mathcal{P}_k)^{(2^h)} = \lambda n_m(\mathcal{P}_k)^{(2^{h-1})} + (1-\lambda) m^{(2^h)}, \tag{28a}$$

$$c_k^{(2^h)} = \lambda c_k^{(2^{h-1})} + \frac{(1-\lambda) m^{(2^h)}}{1-\lambda^{2^h}}, \tag{28b}$$

where $0 < \lambda < 1$ denotes the inertia weight, $m^{(2^h)}$ represents the number of tasks newly joined within the time scale 2^h, and $n_m(\mathcal{P}_k)^{(2^h)}$ and $c_k^{(2^h)}$ are the member count and centroid at the end of the update, respectively. It ensures that the statistical information of micro-cluster $\mathcal{P}_k$ accurately reflects its evolution at this time scale.

Algorithm 1 presents the task clustering algorithm that integrates the density-based stream clustering and prototype-based stream clustering algorithms. Lines 1-8 process each task i in the task stream $\mathcal{I}$, assigning it to the nearest micro-cluster or creating a new micro-cluster based on the density threshold ρ. Lines 9-16 update the micro-cluster statistics over multiple time scales 2^h. The overall time complexity of the algorithm is $O(I \cdot M + H \cdot M)$, where M is the number of micro-clusters and H is the number of time scales considered.

Algorithm 1 Simplified Integrated Density and Prototype Stream Clustering

Input: Task stream $\mathcal{I}$, radius r_d, density threshold ϕ_s, merge radius r_m, time scales 2^h, inertia weight λ
Output: Task blocks $i_{B_1}, \ldots, i_{B_M}$
Initialize: Empty set of micro-clusters $\mathbb{P} = \emptyset$

1: **for** each task $i \in \mathcal{I}$ **do**
2: Find density-reachable micro-clusters: $\mathbb{P}_i = \{\mathcal{P}_k \in \mathbb{P} | \max_{p \in \mathcal{P}_k} \mathcal{D}(i, p) \leq r_d\}$
3: Construct candidate micro-cluster $\mathcal{P}_i = \mathbb{P}_i \cup i$
4: **if** $\phi(\mathcal{P}_i) \geq \phi_s$ **then**
5: add $\mathcal{P}_i$ to $\mathbb{P}$ and merge as $\mathcal{P}_k$ if satisfy eq.(26)
6: **else** Create a micro-cluster $\mathcal{P}_{k'} = \{i\}$ and add it to $\mathbb{P}$
7: **end if**
8: **end for**
9: **for** each time scale 2^h **do**
10: **for** each micro-cluster $\mathcal{P}_k$ **do**
11: Compute outlier factor using eq. (27)
12: **end for**
13: **for** micro-clusters with low O_f **do**
14: Update $n_m(\mathcal{P}_k)^{(2^h)}$ and $c_k^{(2^h)}$ using eq. (28)
15: **end for**
16: **end for**
17: Obtain task blocks i_{B_k} from final micro-clusters $\mathcal{P}_k$
18: **return** Task blocks $i_{B_1}, \ldots, i_{B_M}$

Integrated incremental clustering algorithms can efficiently process large-scale dynamic task streams. The density-based stream clustering algorithm swiftly identifies dense regions within data streams and forms micro-clusters, which then serve as input for the prototype-based stream clustering algorithm. The latter further refines the structure of micro-clusters to more accurately reflect the overall distribution of task streams. The approach clusters the dynamic task stream $\mathcal{I}$ based on critical attributes such as deadline, computational requirements, and task generation location. This results in several distinct task blocks denoted as $i_{B_1}, i_{B_2}, \ldots, i_{B_M}$. Each task block is characterized by its number of members $n_m(\mathcal{P}_i)$ and centroid c_i. These task blocks ensure that the original dynamic task stream $\mathcal{I}$ is completely reconstructed by the union of all task blocks, i.e.,

$$\mathcal{I} = \bigcup_{k=1}^{M} i_{B_k}. \tag{29}$$

The practical implementation of Algorithm 1 in wireless networks involves a three-phase process. In the initialization phase, each EN maintains a local CF-Tree structure to track task patterns. During the streaming phase, when IoT devices generate new tasks, they send metadata $(\zeta_i, \gamma_i \varepsilon_i, l_i)$ to their nearest EN. The EN then updates its CF-Tree structure and performs local clustering, requiring only $O(M)$ memory space where M is the number of micro-clusters. Finally, in the refinement phase, ENs periodically exchange cluster statistics with neighboring nodes to optimize cluster boundaries, incurring $O(J)$ communication overhead where J is the number of ENs in communication range.

C. Geometrized task assignment using weighted Voronoi diagram

An effective regional partitioning strategy should consider the geographical distribution and dynamic resource availability of ENs, and optimally determine w_1 and w_2, to optimize load distribution and resource efficiency, thereby minimizing task completion times. Therefore, the multiplicative weighted Voronoi diagram (MWVD) is employed in regional partitioning. The MWVD is an extension of the classic Voronoi diagram that considers the position and weight of sites when dividing space into distinct regions. The regional partitioning within the MWVD is based on a finite set of site positions $\mathcal{S}$, where each site $s \in \mathcal{S}$ is associated with a corresponding weight μ_s. The methodology employs the weighted Euclidean distance to demarcate the area of influence for each site, resulting in what is known as Voronoi cells $\mathcal{V}_s$. Here, the weighted Euclidean distance is calculated as $d(o,s)/\mu_s$, where $d(o,s)$ represents the ordinary Euclidean distance from a point o to the site s. For any point $o \in \mathcal{V}_s$, the weighted Euclidean distance to the site s is less than that to any other site $s' \in \mathcal{S} \setminus \{s\}$, the relationship is defined as

$$\mathcal{V}_s = \left\{ o \;\middle|\; \frac{1}{\mu_s} d(o,s) < \frac{1}{\mu_{s'}} d(o,s'), \forall s' \in \mathcal{S} \setminus \{s\} \right\}. \quad (30)$$

In applying the MWVD to large-scale task processing in edge computing, the EN j is designated as a site, while task block i_{B_k} is mapped to a point based on its generation location $l_{i_{B_k}}$. The weight of a site is considered to be the remaining computational resources (RCR) $R_j(t)$ of the EN j in the time slot t. Since the weight is positive, the resulting weighted Euclidean distance ensures that each EN's distance to its neighboring boundaries is finite. Each EN thus possesses an enclosed Voronoi cell that contains only itself, ensuring the device layer is geographically partitioned into J subregions, equal to the number of ENs. This subregion is denoted as $\mathcal{V}_j^{(a)}$. The boundary of each $\mathcal{V}_j^{(a)} \in \mathcal{V}^{(a)}$ is composed of the weighted partitioning lines $B_{jj'}(t)$ between adjacent ENs, given by

$$\mathcal{V}_j^{(a)} = \bigcup_{j' \in \mathcal{J} \setminus j} B_{jj'}(t), \quad (31)$$

where $\mathcal{J}$ is the set of all ENs, and $\mathcal{V}^{(a)}$ denotes the set of all such subregions. The RCR $R_j(t)$ are occupied by task execution and then released upon task completion, causing the range of the Voronoi cell $\mathcal{V}_j^{(a)}$ for EN j fluctuate. When the generation location $l_{i_{B_k}}$ of task block i_{B_k} is located within the Voronoi cell $\mathcal{V}_j^{(a)}$ assigned to EN j, the task is assigned to that EN. Consequently, the set of tasks allocated to EN j could be expressed as

$$\mathcal{I}_j^{(a)} = \left\{ i_{B_k} \;\middle|\; \frac{\|l_{i_{B_k}} - l_j\|_2^{1-\kappa}}{R_j^{\kappa}(t)} < \frac{\|l_{i_{B_k}} - l_{j'}\|_2^{1-\kappa}}{R_{j'}^{\kappa}(t)}, j' \neq j \right\}, \quad (32)$$

where $\kappa \in [0,1]$ is a tuning parameter used to balance the influence of weights and distances on task assignment.

In the example depicted in Figure 3, the MWVD partitions the device layer into 10 subregions. Each subregion serviced by an EN is responsible for tasks generated by the IoT devices within that area. ENs with relatively higher RCR $R_j(t)$ are allocated larger Voronoi cells $\mathcal{V}_j^{(a)}$ in the MWVD. The approach enables these ENs to handle a greater number of tasks, thereby achieving effective load balancing among heterogeneous ENs.

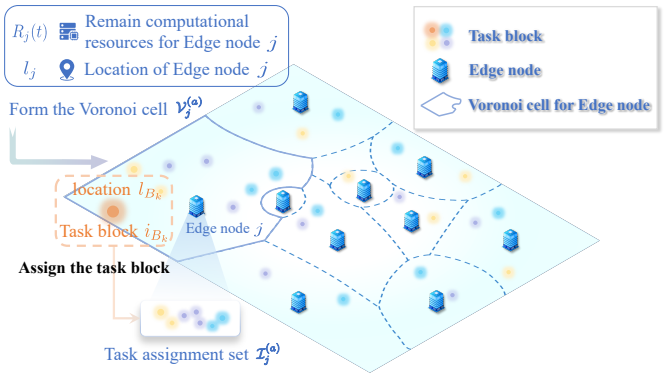


Fig. 3. An illustration of the multiplicative weighted Voronoi diagram for task assignment.

Algorithm 2 Regional partitioning and task assignment based on multiplicative weighted Voronoi diagram

Input: Set of ENs $\mathcal{J}$, RCR $R_j(t)$ for each EN j at time slot t and task stream $\mathcal{I}$
Output: MWVD $\mathcal{V}^{(a)}$ and $\mathcal{I}_j^{(a)}$ for each EN j
Initialize:Initialize an empty set $\mathcal{V}^{(a)}$ to store the Voronoi cells, initialize an empty set $\mathcal{I}_j^{(a)}$ for each EN j to store the assigned tasks

1: **for** each $j \in \mathcal{J}$ **do**
2: Create a Voronoi cell $\mathcal{V}_j^{(a)}$ for EN j
3: Initialize $\mathcal{V}_j^{(a)}$ as the entire plane
4: **for** each $j' \in \mathcal{J} \setminus j$ **do**
5: Compute the weighted bisector $B_{jj'}(t)$:
6: $B_{jj'}(t) \leftarrow \left\{ p \in \mathbb{R}^2 \mid \frac{\|p-l_j\|_2^{1-\kappa}}{R_j^{\kappa}(t)} = \frac{\|p-l_{j'}\|_2^{1-\kappa}}{R_{j'}^{\kappa}(t)} \right\}$
7: ▷ The weighted bisector $B_{jj'}(t)$ divides the plane
8: Update $\mathcal{V}_j^{(a)}$ by intersecting it with the region
9: **end for**
10: Add $\mathcal{V}_j^{(a)}$ to $\mathcal{V}^{(a)}$
11: **end for**
12: **for** each task block $i_{B_k} \in \mathcal{I}$ **do**
13: **for** each EN $j \in \mathcal{J}$ **do**
14: **if** task block location $l_{i_{B_k}} \in \mathcal{V}_j^{(a)}$ **then**
15: Add task block i_{B_k} to the set $\mathcal{I}_j^{(a)}$
16: ▷ Assign task block i_{B_k} to EN j
17: Break the inner loop
18: **end if**
19: **end for**
20: **end for**
return task assignment sets $\mathcal{I}_j^{(a)}$ for each EN j

Algorithm 2 details the process of task assignment to ENs for each time slot t. Lines 1-11 are dedicated to generating the MWVD based on the computational resources of ENs. Lines 12-20 focus on assigning tasks to ENs based on the generated

MWVD. The implementation of Algorithm 2 leverages the distributed nature of edge computing. Each EN maintains its Voronoi cell by exchanging only boundary information with neighboring nodes. The information exchange occurs in three key situations:(1) when EN resources change significantly ($> 20\%$), (2) when task block patterns shift notably, or (3) at regular intervals (typically every 100 time slots). This approach requires each EN to maintain only local topology information and communicate with immediate neighbors, resulting in $O(log\ J)$ average communication overhead per node. The total storage requirement at each node is $O(K)$, where K is the average number of neighboring ENs.

D. Tetris-like task offloading evaluation

ENs should leverage their resources to accept and process as many tasks as possible, just as Tetris players try to fit pieces onto the game board. However, this does not guarantee that an EN will have the capacity to handle all tasks assigned to it, especially when receiving multiple tasks simultaneously, potentially leading to insufficient resource allocation and the inability to meet all task deadlines. Drawing inspiration from how Tetris players must quickly assess and place pieces, ENs should evaluate the tasks they receive to ensure completion within their deadlines. The complexity of the evaluation arises from the fact that the actual computation time of a task depends on the amount of resources allocated to it. The resource allocation is dynamically adjusted based on the set of tasks already accepted by the EN, making task completion time difficult to predict precisely. Therefore, assessing the satisfaction of task deadlines can be translated into assessing the availability of resources at the EN, paralleling the way Tetris players must constantly evaluate the available space on the game board. By analyzing the resource availability of an EN during a specific period, one can estimate whether a task can be completed before its deadline, taking cues from how Tetris players plan their piece placements to succeed in the game.

To assess whether EN j can accept and successfully process task block $i_{B_k} \in \mathcal{I}_j^{(a)}$, the deadline $\zeta_{i_{B_k}}$ of task block i_{B_k} should be considered, similar to how Tetris pieces are placed based on their shape and available space. The task set $\mathcal{I}_j$ already accepted by EN j is subdivided into two subsets according to the deadline $\zeta_{i_{B_k}}$. The first subset $\mathcal{I}^-_{j,i_{B_k}}$ consists of all task blocks with deadlines no later than $\zeta_{i_{B_k}}$, i.e. $\mathcal{I}^-_{j,i_{B_k}} = \{i \mid \zeta_i \leq \zeta_{i_{B_k}}\}$ and the second subset $\mathcal{I}^+_{j,i_{B_k}}$ consists of all task blocks with deadlines later than ζ_i, i.e. $\mathcal{I}^+_{j,i_{B_k}} = \{i \mid \zeta_i > \zeta_{i_{B_k}}\}$, as illustrated in Figure 4.

The task blocks in subset $\mathcal{I}^+_{j,i_{B_k}}$ are sorted in order of their deadlines, i.e, $\zeta_{i_{B_k}} \leq \zeta_{i_q} \leq \zeta_{i_{q+1}} \leq \cdots \leq \zeta_{i_M}$. The emphasis initially is placed on the task block $i_q \in \mathcal{I}^+_{j,i_{B_k}}$ with the earliest deadline ζ_{i_q}. The amount of data that is processed for task block i before time slot t_k is represented as

$$\psi_{i,j}(t_k) = \tau \sum_{t=t_i}^{t_k} r_{i,j}(t). \tag{33}$$

It considers the situation where a portion of the task block has already been processed. Accounting for the number of

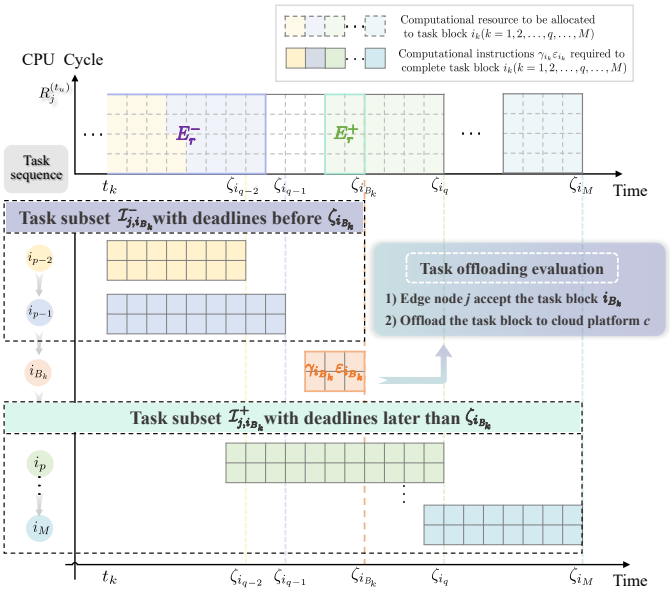


Fig. 4. Tetris-like task offloading evaluation.

instructions required to process task block i_q before the current time slot t_k, the remaining resources in the interval $[\zeta_{i_{q-1}}, \zeta_{i_q}]$ are given by

$$\Delta\mathcal{R}_{j,\zeta_{i_q}} = \begin{cases} (\zeta_{i_q} - \zeta_{i_{B_k}})R_j^{(t_u)} - \gamma_{i_q}\varepsilon_{i_q} + \psi_{i_q,j}(t_k), & q = 1 \\ (\zeta_{i_q} - \zeta_{i_{q-1}})R_j^{(t_u)} - \gamma_{i_q}\varepsilon_{i_q} + \psi_{i_q,j}(t_k) & \\ \quad + \max(0, \Delta\mathcal{R}_{j,\zeta_{i_{q-1}}}), & q > 1 \end{cases}. \tag{34}$$

If $\Delta\mathcal{R}_{j,\zeta_{i_q}} \geq 0$, it indicates that task block i_q can be completed within the interval $[\zeta_{i_{q-1}}, \zeta_{i_q}]$, and EN j has additional resources to process other task blocks, similar to how Tetris pieces can be placed in available spaces. These resources can be used to supplement the available resources in the next interval $[\zeta_{i_q}, \zeta_{i_{q+1}}]$. Conversely, if $\Delta\mathcal{R}_{j,\zeta_{i_q}} < 0$, it implies that the resources in the interval $[\zeta_{i_{q-1}}, \zeta_{i_q}]$ are insufficient to complete task block i_q, and additional resources need to be allocated before the deadline $\zeta_{i_{B_k}}$, similar to how Tetris pieces may not fit in the available space and require adjustments.

For the subset $\mathcal{I}^+_{j,i_{B_k}}$, the resources required to be allocated before the deadline $\zeta_{i_{B_k}}$ are determined as

$$E_r^+ = \sum_{i \in \mathcal{I}^+_{j,i_{B_k}}} \left(\Delta\mathcal{R}_{j,\zeta_i} \min(0, \mathrm{sgn}(\Delta\mathcal{R}_{j,\zeta_i}))\right), \tag{35}$$

where $\mathrm{sgn}(\cdot)$ is the sign function. For the task subset $\mathcal{I}^-_{j,i_{B_k}}$ with deadlines before $\zeta_{i_{B_k}}$, considering that some task blocks may be allocated resources and partially processed before current time slot t_k, the remaining resources required to complete these task blocks are represented by

$$E_r^- = \sum_{i \in \mathcal{I}^-_{j,i_{B_k}}} \left(\gamma_i\varepsilon_i - \psi_{i,j}(t_k)\right). \tag{36}$$

The resources needed between the current time slot t and the deadline of task block i_{B_k} include the additional resources E_r^+ required by the task subset $\mathcal{I}^+_{j,i_{B_k}}$ before $\zeta_{i_{B_k}}$,

the resources E_r^- to complete the task subset $\mathcal{I}_{j,i_{B_k}}^-$, and the resources $\gamma_{i_{B_k}}\varepsilon_{i_{B_k}}$ required by task block i_{B_k}. It should be assessed whether the aggregate resource demand will exceed the resource constraints for the interval $[t_k, \zeta_{i_{B_k}}]$. Hence, the acceptance of task block i_{B_k} by EN j can be determined by

$$E_r^+ + E_r^- + \gamma_{i_{B_k}}\varepsilon_{i_{B_k}} \geq \left(\zeta_{i_{B_k}} - t_k\right) R_j^{(t_u)}. \tag{37}$$

If the task block i_{B_k} meets the condition, then the EN j can accept it, similar to a Tetris piece fitting into the available space. Otherwise, the task block i_{B_k} is offloaded to the cloud for processing, similar to a Tetris piece being discarded when it cannot fit.

Algorithm 3 Tetris-like task offloading evaluation.

Input: Task block i_{B_k} deadline $\zeta_{i_{B_k}}$, required resources $\gamma_{i_{B_k}}\varepsilon_{i_{B_k}}$, set of tasks $\mathcal{I}_j$ already accepted by EN j, current time slot t_k, EN j resource capacity $R_j^{(t_u)}$.
Output: Boolean result indicating if task block i_{B_k} can be offloaded to EN j
Initialize: Resource surplus $E_r^+ \leftarrow 0$, resource deficit $E_r^- \leftarrow 0$
Steps:
1: Partition $\mathcal{I}_j$ into two subsets based on deadlines: $\mathcal{I}_{j,i_{B_k}}^- \leftarrow \{i \in \mathcal{I}_j | \zeta_i \leq \zeta_{i_{B_k}}\}$, $\mathcal{I}_{j,i_{B_k}}^+ \leftarrow \{i \in \mathcal{I}_j | \zeta_i > \zeta_{i_{B_k}}\}$
2: **for** each task block $i_q \in \mathcal{I}_{j,i_{B_k}}^+$ sorted ζ_{i_q} **do**
3: Calculate remaining resources $\Delta\mathcal{R}_{j,\zeta_{i_q}}$ using eq. (34)
4: $E_r^+ \leftarrow E_r^+ + \Delta\mathcal{R}_{j,\zeta_{i_q}} \cdot \min(0, \mathrm{sgn}(\Delta\mathcal{R}_{j,\zeta_{i_q}}))$
5: **end for**
6: **for** each task block $i \in \mathcal{I}_{j,i_{B_k}}^-$ **do**
7: Calculate required resources $\gamma_i\varepsilon_i - \psi_{i,j}(t_k)$
8: $E_r^- \leftarrow E_r^- + \gamma_i\varepsilon_i - \psi_{i,j}(t_k)$
9: **end for**
10: Calculate total resource needs: $E_r \leftarrow E_r^+ + E_r^- + \gamma_{i_{B_k}}\varepsilon_{i_{B_k}}$
11: **if** $E_r \leq (\zeta_{i_{B_k}} - t_k) \cdot R_j^{(t_u)}$ **then**
12: **return** true
13: ▷ EN j can accommodate task block i_{B_k} on schedule
14: **else**
15: **return** false
16: ▷ Task block i_{B_k} should be offloaded to the cloud
17: **end if**

The evaluation algorithm is presented in Algorithm 3. Lines 2-5 process the task subset $\mathcal{I}_{j,iB_k}^+$ with deadlines later than $\zeta_{i_{B_k}}$, The time complexity is $O(M^+)$, where M^+ is the number of task blocks in $\mathcal{I}_{j,iB_k}^+$. Lines 6-9 process the task set $\mathcal{I}_{j,i_{B_k}}^-$ with deadlines no later than $\zeta_{i_{B_k}}$. The time complexity is $O(M^-)$, where M^- is the number of task blocks in $\mathcal{I}_{j,i_{B_k}}^-$. The overall complexity is $O(M)$, linear in the total number of task blocks in EN.

V. Adaptive resource allocation

Following the task offloading assessment, an EN forms an accepted task set $\mathcal{I}_j$. Although the resources at the EN are sufficient to support the execution of task blocks within $\mathcal{I}_j$, devising a specific resource allocation strategy and determining task waiting times remain challenging. Traditional resource allocation methods rely on a predetermined distribution of task requests and utilize queuing theory to estimate waiting times. However, these methods do not sufficiently account for the randomness and high concurrency of task requests, nor the uncertainty of resource requirements for tasks in each time interval. These factors necessitate the ability to adaptively adjust resource allocations according to the task set $\mathcal{I}_j$.

An adaptive resource allocation strategy is introduced to address the issue, the strategy segments task blocks and allocates resources for processing within the sliding window $[t_k, t_k + \Delta t]$. Furthermore, it employs an online learning algorithm to continuously optimize the length of the sliding window Δt, thereby enabling adaptive responses to evolving conditions. The characteristics of containerization technology in edge computing support the implementation. Container technology offers millisecond-level startup times and, thanks to its resource isolation mechanism, allows for dynamic adjustments of resource quotas without restarting containers. Moreover, even when deployed at scale, containers do not adversely affect the overall performance of ENs. These features are highly compatible with our strategy.

A. Deadline-aware task segmentation and dynamic resource allocation

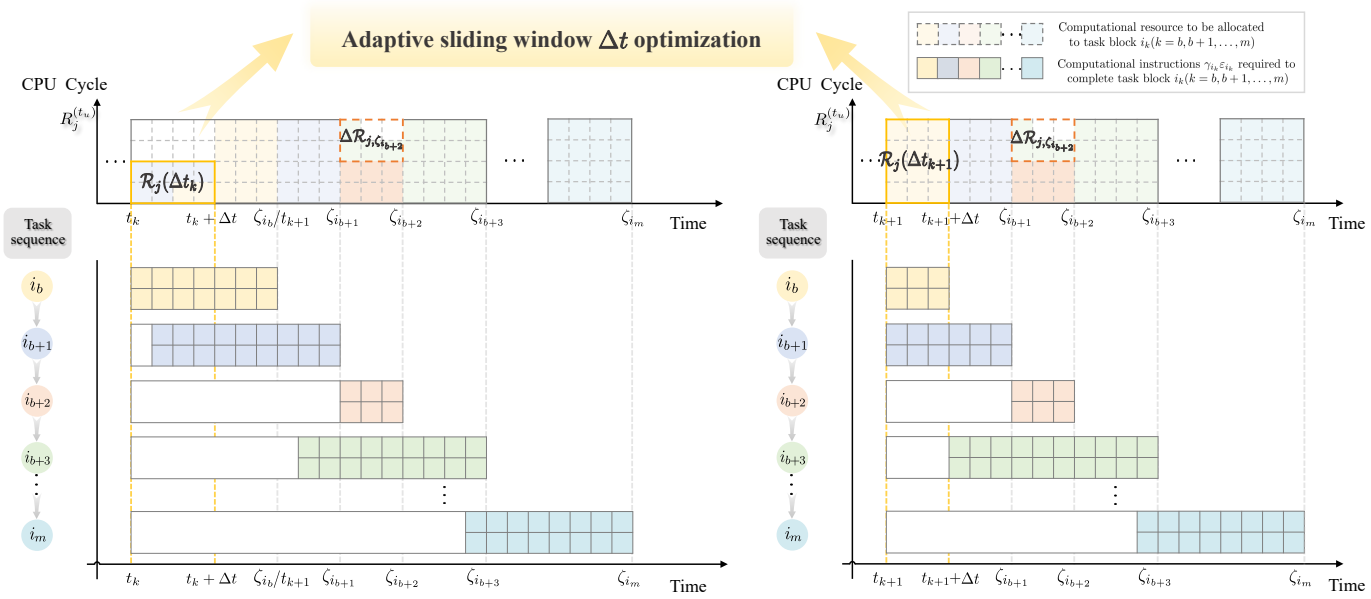
To allocate the computational resources for the sliding window $[t_k, t_k + \Delta t]$, task blocks received by EN j within the task set $\mathcal{I}_j$ should first be sorted according to their respective deadlines. Subsequently, the remaining available resources for each task block before its deadline are evaluated. The remaining available resources for the task block i_{b+1} within the interval $[\zeta_{i_{b-1}}, \zeta_{i_b}]$ can be calculated as

$$\Delta\mathcal{R}_{j,\zeta_{i_b}} = \begin{cases} (\zeta_{i_b} - t_k - \Delta t) R_j^{(t_u)} - \gamma_{i_b}\varepsilon_{i_b} + \psi_{i_b,j}(t_k), & b = 1 \\ (\zeta_{i_b} - \zeta_{i_{b-1}}) R_j^{(t_u)} - \gamma_{i_b}\varepsilon_{i_b} + \psi_{i_b,j}(t_k) & \\ \quad + \max(\Delta\mathcal{R}_{j,\zeta_{i_{b-1}}}, 0), & b > 1 \end{cases}, \tag{38}$$

where $b = 1$ represents the first task block, and $b > 1$ represents the subsequent task blocks. If $\Delta\mathcal{R}_{j,\zeta_{i_b}} \geq 0$, the task block i_b could be completed before its deadline ζ_{i_b}, and EN j has surplus resources for subsequent tasks. These surplus resources could be carried over to the next interval for task processing. Conversely, if $\Delta\mathcal{R}_{j,\zeta_{i_b}} < 0$, it indicates that there are insufficient resources to complete the task block i_b within the interval $[\zeta_{i_{b-1}}, \zeta_{i_b}]$. The task block i_b should be segmented for resource allocation and processing the segmented segment within the sliding window $[t_k, t_k + \Delta t]$. The required resource amount for processing the segment of the task block is given by $|\Delta\mathcal{R}_{j,\zeta_{i_b}}|$.

Following the above procedure, iterate through the subsequent remaining task blocks in the task set $\mathcal{I}_j$. The total amount of resources demanded for all segmented task segments that EN j should process during the sliding window $[t_k, t_k + \Delta t]$ could be calculated as

$$\mathcal{R}_j(\Delta t) = \sum_{\substack{i \in \mathcal{I}_j \\ \zeta_i > t_k}} \left(\Delta\mathcal{R}_{j,\zeta_i} \min(0, \mathrm{sgn}(\Delta\mathcal{R}_{j,\zeta_i}))\right), \tag{39}$$



(a) Dynamic resource allocation at time slot t_k

(b) Dynamic resource allocation at time slot t_{k+1}

Fig. 5. An illustration of adaptive resource allocation.

where $\mathrm{sgn}(\cdot)$ is the sign function. The formula determines the resource allocation for the sliding window $[t_k, t_k + \Delta t]$. Moreover, the RCRs $R_j(t)$ of the EN j for task assignment could be represented as $R_j^{(t_u)} - \mathcal{R}_j(\Delta t)$.

B. Adaptive sliding window optimization driven by non-parametric Gaussian process

Based on the aforementioned resource allocation strategies, an online learning-based adaptive sliding window length optimization method is introduced [32]. It dynamically adjusts the length of the sliding window to adapt to the continuously changing task requests and resource demands, achieving joint optimization of task violation rate and resource utilization, as illustrated in Figure 5.

Specifically, a non-parametric Gaussian process (NNGP) model is adopted to establish the mapping relationship between the sliding window length and the objective function value. Given a set of historical sample points $\mathcal{H} = (\Delta t_k, y)_{k=1}^m$, where Δt_k represents the sliding window length, and y_i represents the corresponding objective function value, the NNGP model can be expressed as

$$f(\Delta t) \sim GP(M(\Delta t), K(\Delta t, \Delta t')), \tag{40}$$

where $M(\Delta t)$ is the mean function and $K(\Delta t, \Delta t')$ is the kernel function used to measure the similarity between different sliding window lengths. In order to capture the complex relationship between the sliding window length and the objective function value better, a composite kernel function is designed as

$$\begin{aligned} K(\Delta t, \Delta t') =& K_1 \varsigma_1^2 \exp\left(-\frac{2\sin^2(\pi|\Delta t - \Delta t'|/\varphi)}{\ell_1^2}\right) \\ &+ K_2 \varsigma_2^2 \exp\left(-\frac{(\Delta t - \Delta t')^2}{2\ell_2^2}\right), \end{aligned} \tag{41}$$

where ς_1^2 and ς_2^2 control the signal intensities of the periodic kernel and the Gaussian kernel, respectively, ℓ_1 and ℓ_2 control the smoothness of the two kernel functions, and φ is the period parameter of the periodic kernel function.

At the end of each sliding window, the average resource utilization rate $\bar{\mathcal{U}}_{\Delta t}$ and task violation rate $\bar{\Theta}_{\Delta t}$ are computed and serve as the objective function value, expressed as

$$y = F_1 \bar{\Theta}_{\Delta t} + F_2 \bar{\mathcal{U}}_{\Delta t}, \tag{42}$$

where F_1 and F_2 are weight coefficients used to balance the importance of the two optimization objectives, task violation rate and resource utilization. Specifically, the task violation rate $\bar{\Theta}_{\Delta t}$ can be expressed as

$$\bar{\Theta}_{\Delta t} = \frac{\sum_{i=1}^{I(\Delta t)} \mathcal{V}_i(i)}{I(\Delta t)}, \tag{43}$$

where $I(\Delta t)$ represents the number of tasks whose deadlines fall within $[t_k, t_k + \Delta t]$, and $\mathcal{V}_i(i)$ represents the deadline violation status of task block i and can be converted to the following form as

$$\mathcal{V}_i(i) = \mathbb{I}(\psi_{i,j}(\zeta_i) \leq \gamma_i \varepsilon_i), \tag{44}$$

where $\mathbb{I}(\cdot)$ is an indicator function. The average resource utilization rate $\bar{\mathcal{U}}_{\Delta t}$ within the sliding time window is given by

$$\bar{\mathcal{U}}_{\Delta t} = \frac{\sum_{j=1}^{J} \frac{\mathcal{R}_j(\Delta t)}{R_j^{(t_u)} \Delta t}}{J}. \tag{45}$$

Subsequently, $(\Delta t, y)$ is treated as a new sample point to update the NNGP model. When selecting the optimal sliding window length, the algorithm balances the exploration and exploitation strategies to balance the trade-off between local optimality and global optimality. The exploration rate β is

dynamically adjusted over time, with a higher rate initially and a lower rate later, i.e.,

$$\beta(t) = \beta_0 \exp\left(-\frac{t}{\nu_1}\right) + \beta_\infty \left(1 - \exp\left(-\frac{t}{\nu_2}\right)\right), \quad (46)$$

where β_0 is the initial exploration rate, β_∞ is the final exploration rate, and ν_1 and ν_2 are two time-scale parameters that control the short-term and long-term decay rates of the exploration rate, respectively. The algorithm continuously optimizes the sliding window length to adapt to dynamic task requests and resource demands through the online learning approach. It ensures that the algorithm explores new possibilities while also utilizing experience. In addition, it possesses strong adaptability and robustness, capable of self-adaptively adjusting optimization strategies in dynamic scenarios to achieve joint optimization of task violation rate and resource utilization. Furthermore, benefiting from the efficiency of the NNGP model, the algorithm maintains low computational overhead even on large-scale datasets.

Algorithm 4 Deadline-aware task segmentation and dynamic resource allocation with adaptive sliding window optimization.

Input: Set of tasks $\mathcal{I}_j$ received by EN j, current time slot t_k, and historical sample set $\mathcal{H}$
Output: Resource allocation for tasks in the sliding window $[t_k, t_k + \Delta t]$
Initialize: Total resource requirement for split tasks $\mathcal{R}_j(\Delta t) \leftarrow 0$, exploration rate $\beta \leftarrow \beta_0$, and sliding window length $\Delta t \leftarrow \Delta t_0$
Steps:
1: Sort tasks in $\mathcal{I}_j$ based on their respective deadlines
2: **for** each task $i_b \in \mathcal{I}_j$ in the sorted order **do**
3: **if** $\zeta_{i_b} > t_k$ **then**
4: Compute resources $\Delta\mathcal{R}_{j,\zeta_{i_b}}$ using eq. (38)
5: Calculate demand $\mathcal{R}_j(\Delta t)$ using eq. (39)
6: **end if**
7: **end for**
8: Compute the task violation rate $\bar{\Theta}_{\Delta t}$ and average resource utilization $\bar{\mathcal{U}}_{\Delta t}$
9: Compute the objective function value y using eq. (42)
10: Update the historical sample set $\mathcal{H} \leftarrow \mathcal{H} \cup (\Delta t, y)$
11: Update the NNGP model using the sample set $\mathcal{H}$
12: Update the exploration rate β using eq. (46)
13: Select a new length Δt based on the exploration-exploitation strategy
14: **return** $\mathcal{R}_j(\Delta t)$ as the resource allocation for the sliding window $[t_k, t_k + \Delta t]$.

The resource allocation algorithm shown in Algorithm 4 first sorts the task set according to deadlines in Line 2, and then performs task segmentation and resource allocation based on the remaining resources in lines 2-7. After the resource allocation, the algorithm computes the optimization objective and updates the NNGP model using the new sample point in lines 8-13. The main complexity of the algorithm comes from the task segmentation and resource allocation part. Its time complexity is $O(I \cdot \log I)$. Moreover, updating the NNGP model involves an additional complexity that depends on the size of the historical sample set and the kernel function used.

VI. Simulation and Results

All methods are implemented in MATLAB R2022a and tested on a Windows 11 computer configured with an Intel Core i7-12700, 2.10 GHz, and 32 GB RAM.

A. Dataset and experimental setup

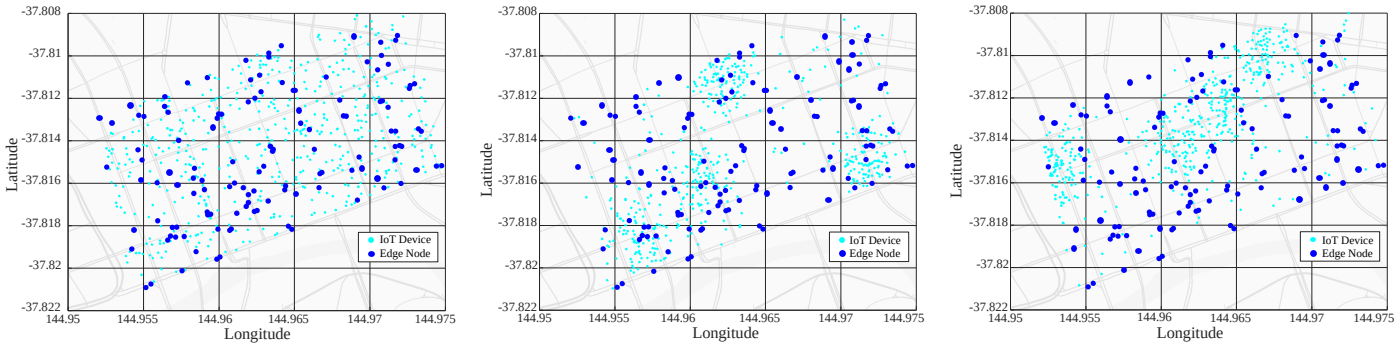
In the study, a real-world EUA dataset [33] from the Melbourne CBD area in Australia is employed to validate the algorithm proposed. The dataset is widely used in the field of mobile edge computing research [7], [31], [34], [35]. It includes geographical location data for 125 ENs and 500 IoT device clusters, with each cluster containing 5 IoT devices, as depicted in Figure 6a. To further simulate diverse device distributions, two groups of devices following Gaussian distributions are randomly generated to replace the devices in Figure 6a, forming two new datasets. One group of devices is positioned near the EN clusters, as illustrated in Figure 6b. Conversely, the other group of devices is positioned far from the EN clusters, as depicted in Figure 6c,

In the simulations, three different scenarios are considered, including random and high-concurrency task requests, different numbers of compute-intensive task requests, and different numbers of ENs. The communication radius of each EN ranges from 150 to 200 meters. The additional experimental parameters for these scenarios are summarized in Table I.

B. Introduction of Compared Algorithms

To further validate the proposed approaches, we implemented five schemes for comparison: RS-QRA, GWS-QRA, GEOS-QRA, GEOS-DRAF, and GEOS-DRAA. The first two schemes are based on random scheduling and greedy workload scheduling, respectively. The latter three adopt the proposed geometrized scheduling strategy with different resource allocation strategies. The algorithms are summarized as follows.

1) *Random scheduling + queuing resource allocation (RS-QRA)* [34]. In the scheme, computational tasks are assigned to either randomly selected ENs or the cloud for processing. ENs process the tasks in the order of their arrival, following the sequence of deadlines within the current task set.
2) *Greedy workload scheduling + queuing resource allocation (GWS-QRA)* [35]. The scheme evaluates the additional workload of all the ENs and assigns tasks to the EN with the lowest workload. Tasks are processed in the order of their deadlines by the ENs.
3) *Geometrized scheduling + queuing resource allocation (GEOS-QRA)*. The scheme adopts a geometrized scheduling strategy, including using the integrated incremental streaming clustering algorithm for dynamic task flow clustering, using the weighted Voronoi diagram for region-based task assignment, and conducting a Tetris-like task offloading evaluation. Tasks are processed in the order of their deadlines by the ENs.



(a) EUA dataset 1 (b) EUA dataset 2 (c) EUA dataset 3

Fig. 6. Different distributions of Edge notes and IoT devices in the Melbourne CBD, Australia.

TABLE I
SIMULATION PARAMETERS FOR DIFFERENT SCENARIOS.

Parameter	Description	Scenario 1	Scenario 2	Scenario 3	Unit
J	the total number of edge nodes j	125	125	[25, 120]	-
$R_j^{(t_u)}$	Data processing capacity of edge node j	2×10^3	2×10^3	2×10^3	MIPS
f^c	Data processing capacity of the cloud platform	3.5×10^5	3.5×10^5	3.5×10^5	MIPS
γ_i	Amount of data to be uploaded for task i	[5, 20]	[5, 20]	[5, 20]	MB
ε_i	Number of instructions per bit for task i	[100, 1000]	[8000, 10000]	[1000, 3000]	MIPS/bit
ι_i	Container image size for task i	[10, 100]	[10, 100]	[10, 100]	MB
ζ_i	Deadline for task i	[5, 10]	[25, 30]	[5, 10]	s
τ	Length of one time slot	0.1	0.1	0.1	s
P_n	Transmission power of IoT device	0.6	0.6	0.6	dBm
B	Uplink bandwidth between IoT devices and edge node	10	10	10	MHz
$w_{n,j}$	Uplink rate between IoT device n and edge node j	1	1	1	MB/s
v	Path loss exponent	2	2	2	-
$\lvert h\rvert^2$	Channel gain of uplink	1	1	1	-
σ_0^2	Power of Gaussian white noise	-100	-100	-100	dBm
$w_{n,j}^{(wired)}$	Transmission rate for wired connections	30	30	30	MB/s
$w_{c,j}$	Downlink rate between edge node and cloud	10	10	10	MB/s
$w_{j,c}$	Uplink rate between edge node and cloud	1	1	1	MB/s
$w_{n,c}$	Uplink rate between IoT device n and cloud	0.1	0.1	0.1	MB/s

4) *Geometrized scheduling + dynamic resource allocation with fixed sliding window length (GEOS-DRAF).* The scheme adopts the same geometrized scheduling strategy as GEOS-QRA, but employs a different strategy for resource allocation. The ENs adopt a deadline-aware task segmentation and dynamic resource allocation strategy based on a fixed sliding window length.
5) *Geometrized scheduling + dynamic resource allocation with adaptive sliding window length (GEOS-DRAA).* The scheme adopts the same geometrized scheduling strategy as GEOS-DRAF but employs a different strategy for resource allocation. The ENs adopt a dynamic resource allocation strategy based on adaptive optimization of the sliding window length.

C. Simulation results

QoS represents a critical metric in edge computing, reflected by the average task deadline violation rate $\bar{\Theta}$. The study primarily utilizes $\bar{\Theta}$ as the principal measure to evaluate the performance of various schemes. The average offloading rate $\bar{O}$ indicates the proportion of tasks offloaded from ENs to the cloud platform. It is considered to assess the effectiveness of edge computing in alleviating the computational pressure on the ENs. The average resource utilization rate $\bar{U}$ of the EN cluster evaluates the efficient utilization of computing resources in the edge computing situation. Each category of solution undergoes 100 repeated experiments, and their average results are recorded.

1) Random and high-concurrency task requests with simulation time: A combination of various probability distributions is employed to simulate random and high-concurrency task requests as encountered in the real world, for the purpose of assessing the effectiveness of scheduling algorithms. Regular requests are generated using a Poisson distribution, while a normal distribution is utilized to simulate expected peak requests within specific time intervals. Sporadic high-traffic events are randomly replicated using heavy-tailed distributions. Additional random factors are introduced to enhance the realism of the simulation. Given the complexity and

unpredictability of such requests, traditional queuing theory is inadequate for estimating queue times in this context. Hence, the simulation approach provides a comprehensive examination of an algorithm's performance in responding to random and dynamically changing request scenarios.

TABLE II
RESULTS OF DIFFERENT EUA DATASETS IN RANDOM AND HIGHLY CONCURRENT TASK REQUEST SCENARIOS, THE AVG $\bar{\Theta}$ REPRESENTS THE OVERALL AVERAGE OF THE AVERAGE TASK DEADLINE VIOLATION RATE $\bar{\Theta}$ ACROSS ALL SIMULATION DURATIONS BEING EVALUATED.

Methods	Dataset1	Dataset2	Dataset3
	AVG $\bar{\Theta}$	AVG $\bar{\Theta}$	AVG $\bar{\Theta}$
RS-QRA	23.90%	30.16%	22.69%
GWS-QRA	12.41%	19.46%	13.90%
GEOS-QRA	7.74%	6.81%	6.50%
GEOS-DRAF	3.37%	3.00%	2.31%
GEOS-DRAA	1.30%	1.55%	1.07%

To test the long-term stability of performance parameters, the simulation lasts until 50,000 time units. Figures 7, 8, and 9 illustrate the variation in the average resource utilization rate $\bar{\mathcal{U}}$ and the average offloading rate $\bar{\mathcal{O}}$ for five strategies across different simulation durations. Table II presents the experimental results of different EUA datasets in random and highly concurrent task request scenarios. The best and second-best values in each column are marked in dark gray and light gray, respectively, as the simulation duration varies. Compared to other methods, GEOS-DRAF and GEOS-DRAA achieve the lowest average task deadline violation rate $\bar{\Theta}$ across all EUA datasets, with GEOS-DRAA performing optimally in all datasets. Specifically, GEOS-DRAF outperforms other baseline methods on EUA dataset 3. Compared to RS-QRA, the average task deadline violation rate $\bar{\Theta}$ of GEOS-DRAF is reduced to 10.19% of that of RS-QRA. Similarly, GEOS-DRAF's $\bar{\Theta}$ is 16.62% of the performance achieved by GWS-QRA and 35.54% of the performance achieved by GEOS-QRA.

In addition, GEOS-DRAA exhibits further superior performance in reducing the average task deadline violation rate $\bar{\Theta}$. The $\bar{\Theta}$ achieved by GEOS-DRAA is only 4.72% of that of RS-QRA, 7.70% of that of GWS-QRA, 16.46% of that of GEOS-QRA, and 46.32% of the performance achieved by GEOS-DRAF. The remarkable improvement by GEOS-DRAA could be attributed to its employment of a variable sliding time window length. It enables adaptive optimization for both the average task deadline violation rate and the average resource utilization rate. As a result of the adaptive optimization approach, building upon the foundation of GEOS-DRAF, GEOS-DRAA further reduces the average task deadline violation rate while simultaneously enhancing the average resource utilization rate of ENs, as evidenced by Figure 9b. Additionally, GEOS-DRAA constrains the average offloading rate within a more reasonable range, as illustrated in Figure 9b. Through these measures, GEOS-DRAA effectively adapts to scenarios involving random and highly concurrent task requests.

To evaluate the effectiveness of the proposed geometry-based scheduling method, a comparative analysis of its performance against the GWS-QRA and RS-QRA approaches is conducted, as illustrated in figures 7, 8, and 9. While the GEOS-QRA variant exhibits a lower average resource utilization rate $\bar{\mathcal{U}}$ compared to GWS-QRA, as depicted in Figure 9b, it demonstrates superior performance in terms of the average task deadline violation rate $\bar{\Theta}$. The enhancement in the average task deadline violation rate $\bar{\Theta}$ performance can be ascribed to GEOS-QRA's offloading strategy. Computationally intensive tasks demanding substantial computational resources are offloaded to the cloud platform for processing. The offloading approach effectively reduces the computational load on EN, consequently improving the overall performance. In contrast, although RS-QRA also offloads some computationally intensive tasks to the cloud platform, the offloading process is carried out randomly without proper evaluation. As a result, RS-QRA falls behind GEOS-QRA in terms of the average task deadline violation rate $\bar{\Theta}$ performance metric.

Comparing the performance of GEOS-QRA and GEOS-DRAF in Figure 9, it can be observed that at the beginning of the simulation, the difference in the average task deadline violation rate $\bar{\Theta}$ between the two methods is insignificant. However, as the simulation progresses, GEOS-QRA maintains a constant level of the average task deadline violation rate $\bar{\Theta}$, while GEOS-DRAF further converges to a lower value. It indicates that GEOS-DRAF is capable of continuously adapting to the task requests and coordinating the resources of ENs and the cloud platform to achieve a better performance in terms of the average task deadline violation rate. As depicted in figures 9b and 9c, although both methods employ the same task scheduling approach, GEOS-DRAF exhibits a higher EN resource utilization rate and a higher task offloading rate compared to GEOS-QRA. It indicates that during the resource allocation process for ENs, the queuing method struggles to adapt to random and highly concurrent task requests, failing to fully consider the task processing status and promptly adjust resource allocation. Consequently, the performance in terms of the average task deadline violation rate $\bar{\Theta}$ is suboptimal. To some extent, it reflects the effectiveness of dynamic resource allocation.

The superior performance of GEOS-DRAF and GEOS-DRAA in the EUA dataset 3 could be attributed to the presence of numerous IoT devices located in the central region, albeit distant from the ENs. The distribution allows for effective load balancing, as the surrounding ENs provide sufficient computational and communication resources to handle the random and highly concurrent requests. In contrast, while the IoT devices in the EUA dataset 1 are more evenly distributed, a certain number of devices are situated at the edges of the region, leading to difficulties in efficiently processing the requests generated by these devices. Regarding the EUA dataset 2, the IoT devices are located in close proximity to the ENs, resulting in low transmission delays. However, this distribution pattern also leads to a significant number of tasks being scheduled to a subset of ENs, causing excessive load on these specific ENs. It is evidenced by the findings presented in Figure 8b. Furthermore, in combination with figures 7b and 9b, it can be noted that GEOS-DRAF exhibits a noticeable decrease in the average resource utilization rate $\bar{\mathcal{U}}$ for the EUA

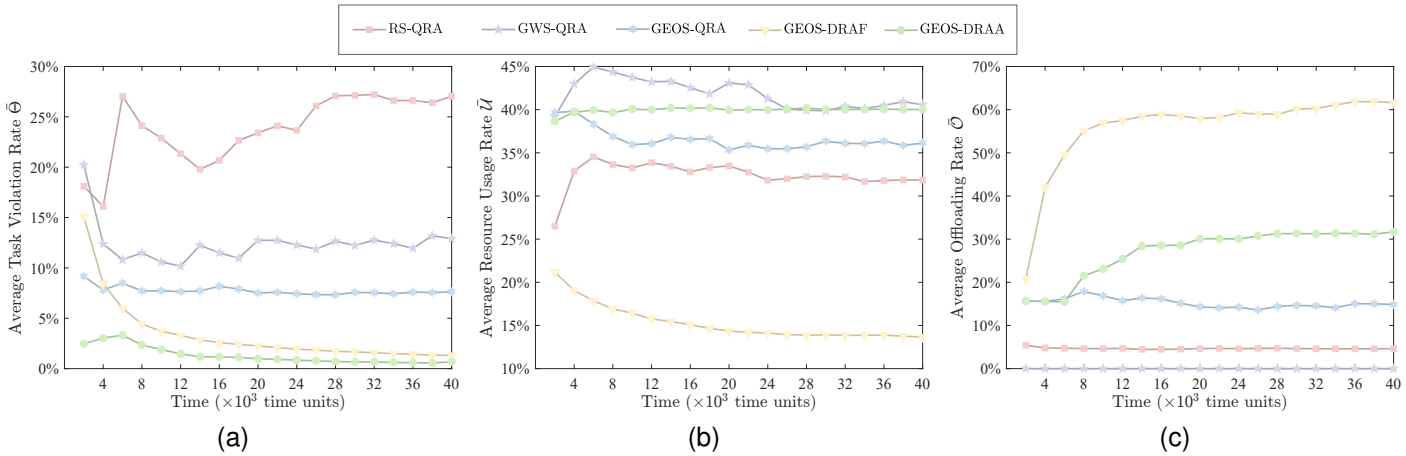


Fig. 7. Performance comparisons of (a) average task violation rate $\bar{\Theta}$, (b) average resource utilization rate $\bar{\mathcal{U}}$, and (c) average offloading rate $\bar{\mathcal{O}}$ on EUA dataset 1 across different simulation durations.

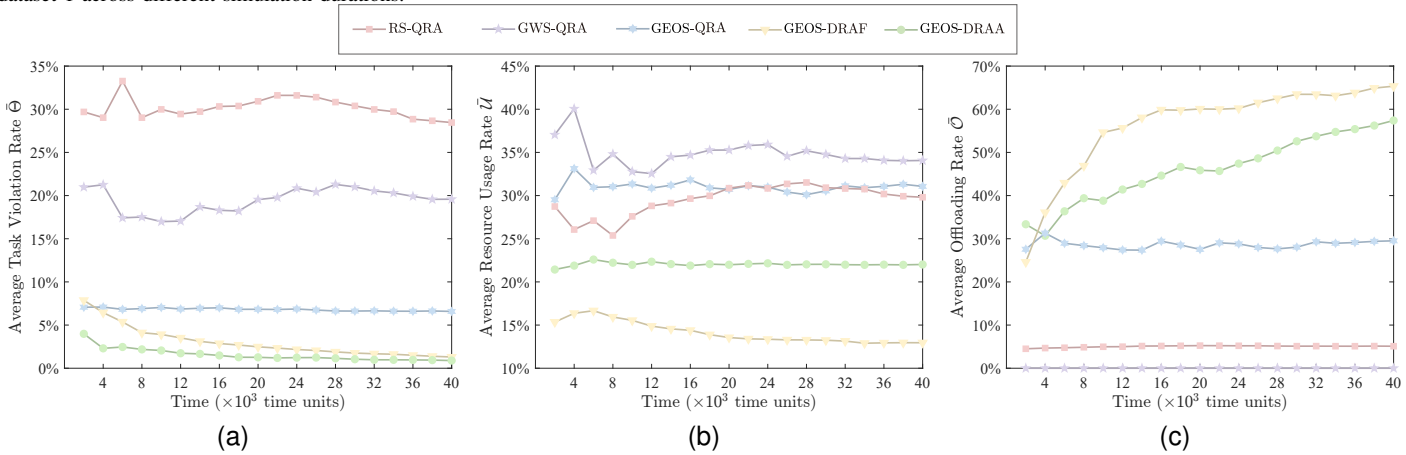


Fig. 8. Performance comparisons of (a) average task violation rate $\bar{\Theta}$, (b) average resource utilization rate $\bar{\mathcal{U}}$, and (c) average offloading rate $\bar{\mathcal{O}}$ on EUA dataset 2 across different simulation durations.

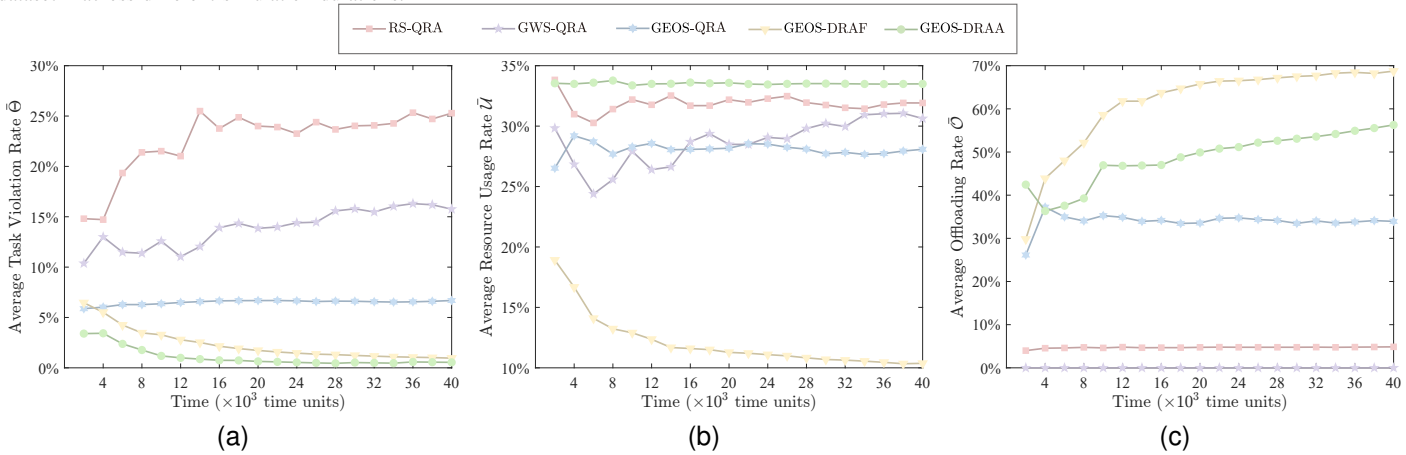


Fig. 9. Performance comparisons of (a) average task violation rate $\bar{\Theta}$, (b) average resource utilization rate $\bar{\mathcal{U}}$, and (c) average offloading rate $\bar{\mathcal{O}}$ on EUA dataset 3 across different simulation durations.

dataset 2 compared to the other two datasets.

2) Different numbers of compute-intensive task requests: To assess the capability of different strategies to handle large-scale compute-intensive tasks, the number of instructions required to process each bit of data ε_i is increased to the range of [8000, 10000] MIPS, and the deadline of task i is correspondingly extended to the range of [25, 30] s. A total of 10000 tasks are set to simulate a compute-intensive scenario.

Figures 10, 11 ,and 12 show the variation in average resource utilization rate $\bar{\mathcal{U}}$, and average offloading rate $\bar{\mathcal{O}}$ across five strategies as the number of completed tasks changes. Table III presents the experimental results for different EUA datasets as the number of computationally intensive tasks varies. The best and second-best values in each column are marked with dark gray and light gray, respectively.

The results demonstrate that GEOS-DRAF and GEOS-

DRAA exhibit strong adaptability in scenarios involving large-scale computationally intensive tasks, with GEOS-DRAA achieving the optimal average task deadline violation rate $\bar{\Theta}$ across all EUA datasets. Specifically, the average task deadline violation rate $\bar{\Theta}$ achieved by GEOS-DRAA is only 12.48% of that of RS-QRA, 22.83% of the performance achieved by GWS-QRA, 26.33% of that of GEOS-QRA, and 59.46% of the performance achieved by GEOS-DRAF in the EUA dataset 1.

TABLE III
RESULTS OF DIFFERENT EUA DATASETS IN COMPUTE-INTENSIVE SCENARIOS WITH A LARGE NUMBER OF TASKS, THE AVG $\bar{\Theta}$ REPRESENTS THE OVERALL AVERAGE OF THE AVERAGE TASK DEADLINE VIOLATION RATE $\bar{\Theta}$ ACROSS ALL THE TASKS BEING EVALUATED.

Methods	Dataset1	Dataset2	Dataset3
	AVG $\bar{\Theta}$	AVG $\bar{\Theta}$	AVG $\bar{\Theta}$
RS-QRA	19.39%	19.81%	20.74%
GWS-QRA	10.60%	10.51%	13.97%
GEOS-QRA	9.19%	9.33%	6.91%
GEOS-DRAF	4.07%	4.54%	3.18%
GEOS-DRAA	2.42%	3.01%	2.96%

Figures 10b, 11b, and 12b illustrate that as the number of completed tasks increases, the average resource utilization rate $\bar{\mathcal{U}}$ of GEOS-DRAA and GEOS-DRAF remains at a relatively lower level compared to other methods, while their average task offloading rate $\bar{\mathcal{O}}$ is significantly higher and continues to grow, as shown in figures 10c, 11c, and 12c. Consequently, the average task deadline violation rate $\bar{\Theta}$ of GEOS-DRAA and GEOS-DRAF further decreases as the number of completed tasks increases, indicating that the dynamic resource allocation approach effectively handles computationally intensive tasks while maintaining the average task offloading rate $\bar{\mathcal{O}}$ within a reasonable range. The approach leverages the low communication latency of ENs while also utilizing the powerful computational capabilities of the cloud platform for intensive tasks.

For this large-scale computationally intensive task scenario, GEOS-DRAF and GEOS-DRAA achieve the best performance in the EUA dataset 1, primarily due to the more uniform distribution of IoT devices. In situations where the average load on ENs is high, tasks require an efficient scheduling strategy to ensure timely completion and to fully utilize the powerful computational capabilities of the cloud platform. With sufficient bandwidth, tasks can be offloaded to the cloud platform for processing.

3) Different numbers of ENs: The simulations are designed to evaluate the impact of the number of ENs on the performance of scheduling strategies. In these simulations, the number of ENs progressively increases from 25 to 120, with an increment of 5 ENs in each simulation. Simultaneously, the number of instructions required to process each bit of data ε_i is adjusted in the range of [1000, 3000]. All other parameters remain consistent across simulations, except for the varying number of ENs. Figures 7a, 8a, and 9a depict the variation in the average resource utilization rate $\bar{\mathcal{U}}$ and the average offloading rate $\bar{\mathcal{O}}$ for the five strategies as the number of ENs changes. Table IV presents the results for different EUA datasets as the number of ENs varies, with the best and second-best values in each column marked with dark gray and light gray, respectively. Compared to other methods, GEOS-DRAF and GEOS-DRAA achieve the lowest average task deadline violation rate $\bar{\Theta}$ across all EUA datasets, with GEOS-DRAA performing optimally in all datasets. Specifically, in the EUA dataset 3, the average task deadline violation rate $\bar{\Theta}$ achieved by GEOS-DRAA is only 3.07% of that of RS-QRA, 7.90% of the performance achieved by GWS-QRA, 18.24% of that of GEOS-QRA, and 32.51% of the performance achieved by GEOS-DRAF.

TABLE IV
RESULTS OF DIFFERENT EUA DATASETS FOR VARIOUS NUMBERS OF EDGE NODES, AVG $\bar{\Theta}$ REPRESENTS THE OVERALL AVERAGE OF THE AVERAGE TASK DEADLINE VIOLATION RATE $\bar{\Theta}$ ACROSS THE EVALUATED EDGE NODE COUNTS.

Methods	Dataset1	Dataset2	Dataset3
	AVG $\bar{\Theta}$	AVG $\bar{\Theta}$	AVG $\bar{\Theta}$
RS-QRA	52.74%	52.15%	51.49%
GWS-QRA	16.96%	20.50%	19.99%
GEOS-QRA	10.24%	9.49%	8.66%
GEOS-DRAF	7.97%	6.36%	4.86%
GEOS-DRAA	2.48%	1.92%	1.58%

The comparative analysis results, shown in Figures 13a, 14a, and 15a, reveal that GEOS-QRA, GEOS-DRAF, and GEOS-DRAA consistently maintain an average task deadline violation rate $\bar{\Theta}$ below 10%. These three methods, employing the geometrized scheduling approach, outperform both GWS-QRA and RS-QRA. In contrast, the average task deadline violation rate $\bar{\Theta}$ of GWS-QRA is relatively high when the number of ENs is limited, even approaching the performance of the random scheduling method RS-QRA. This observation reflects the fact that a limited number of ENs lack sufficient resources to process and ensure that all tasks meet their deadline requirements. As the number of ENs increases, the performance of GWS-QRA improves, approaching that of GEOS-DRAA only when the number of ENs is sufficient. On the other hand, regardless of whether the number of ENs is limited or sufficient, the average task deadline violation rate $\bar{\Theta}$ of RS-QRA consistently exceeds 40%, indicating its poor adaptability to the scenario. Among the three geometrized scheduling methods, GEOS-DRAA consistently achieves the lowest average task deadline violation rate $\bar{\Theta}$, outperforming both GEOS-QRA and GEOS-DRAF.

Figures 13b, 14b, and 15b reveals that the average resource utilization rate $\bar{\mathcal{U}}$ of the various methods fluctuates as the number of ENs changes. Although the number of ENs varies, the other parameters remain unchanged, implying that the total number of tasks remains constant. Therefore, the overall trend indicates that the average resource utilization rate $\bar{\mathcal{U}}$ is higher when the number of ENs is limited and decreases as the number of ENs increases. Similarly, when the number of ENs is limited, the offloading rates of GEOS-QRA, GEOS-DRAF, and GEOS-DRAA are higher, demanding greater cloud-edge communication bandwidth. However, as the number of ENs increases, the average task offloading rate $\bar{\mathcal{O}}$ exhibits a decreasing trend, as illustrated in figures 13c, 14c, and 15c.

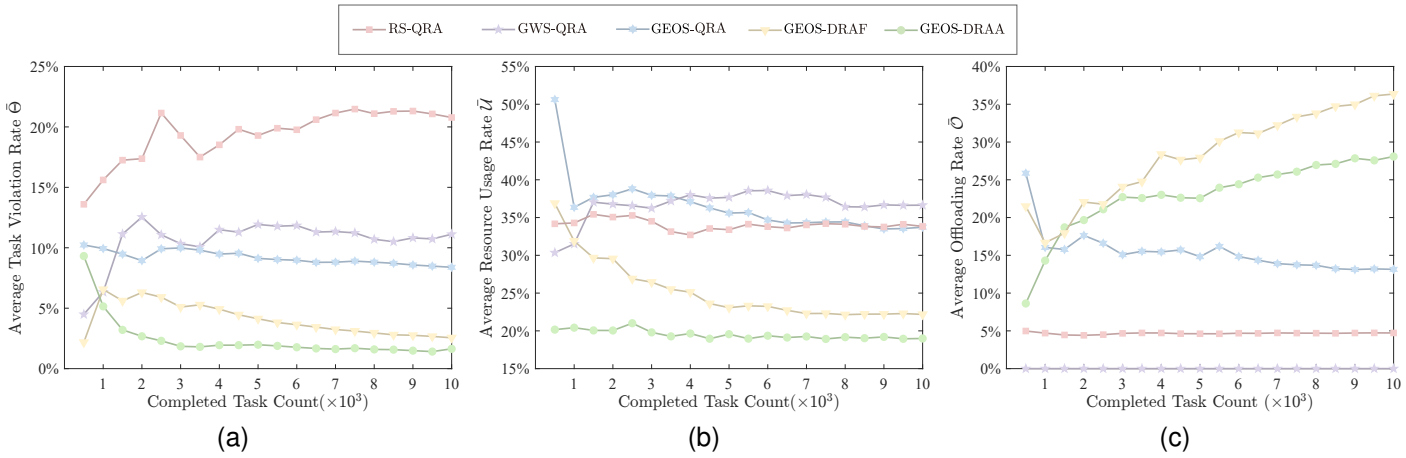


Fig. 10. Performance comparisons of (a) average task violation rate $\bar{\Theta}$, (b) average resource utilization rate $\bar{\mathcal{U}}$, and (c) average offloading rate $\bar{\mathcal{O}}$ on EUA dataset 1 with varying numbers of completed task count.

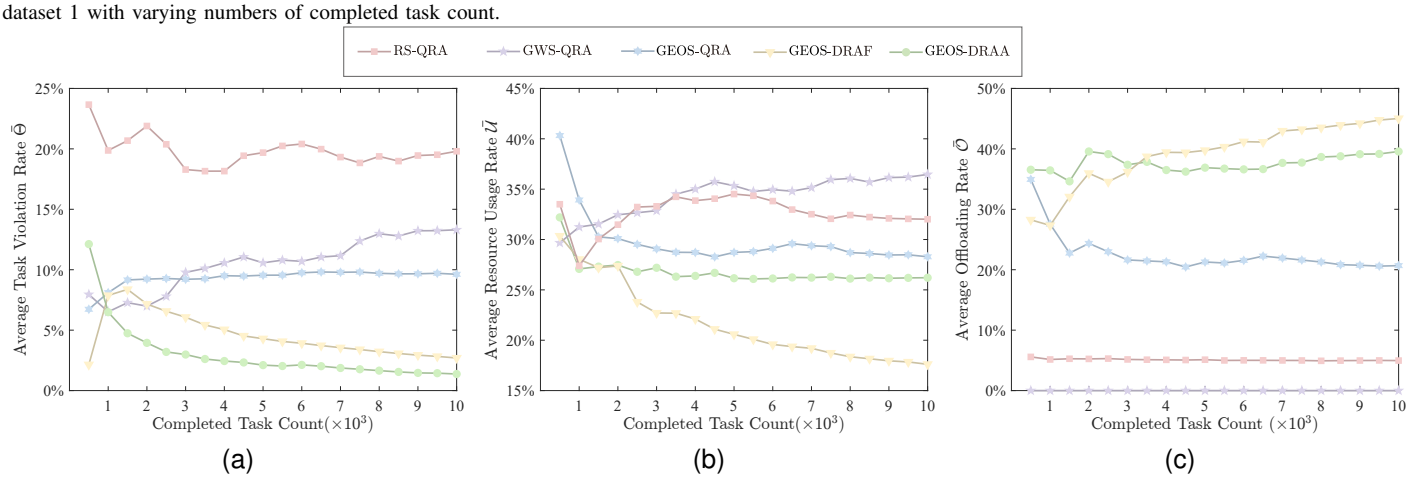


Fig. 11. Performance comparisons of (a) average task violation rate $\bar{\Theta}$, (b) average resource utilization rate $\bar{\mathcal{U}}$, and (c) average offloading rate $\bar{\mathcal{O}}$ on EUA dataset 2 with varying numbers of completed task count.

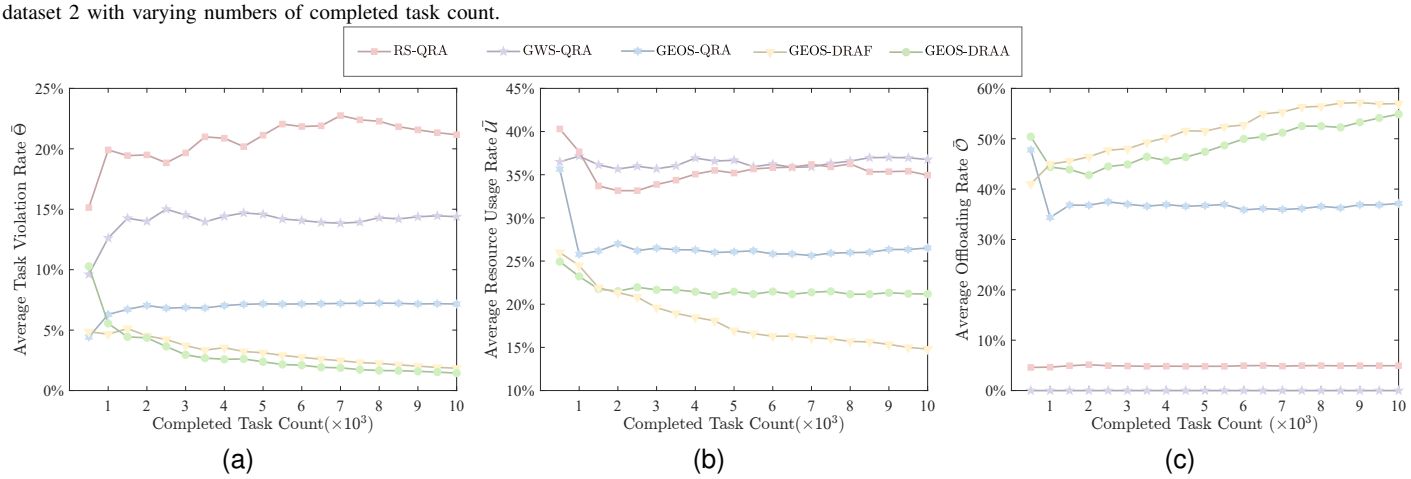


Fig. 12. Performance comparisons of (a) average task violation rate $\bar{\Theta}$, (b) average resource utilization rate $\bar{\mathcal{U}}$, and (c) average offloading rate $\bar{\mathcal{O}}$ on EUA dataset 3 with varying numbers of completed task count.

Consequently, in scenarios with sufficient cloud-edge communication bandwidth, regardless of the number of ENs, GEOS-QRA, GEOS-DRAF, and GEOS-DRAA consistently demonstrate better adaptability to such scenarios, effectively showcasing their advantages in efficient task scheduling and dynamic resource allocation.

D. Use Case: Traffic Monitoring Analysis

Based on our experiments with the Melbourne CBD dataset, we demonstrate the framework implementation through a traffic monitoring system and further analyze its practical benefits from an end-user QoE perspective.

The system processes traffic monitoring tasks from 500 video sensors distributed across Melbourne CBD, with pro-

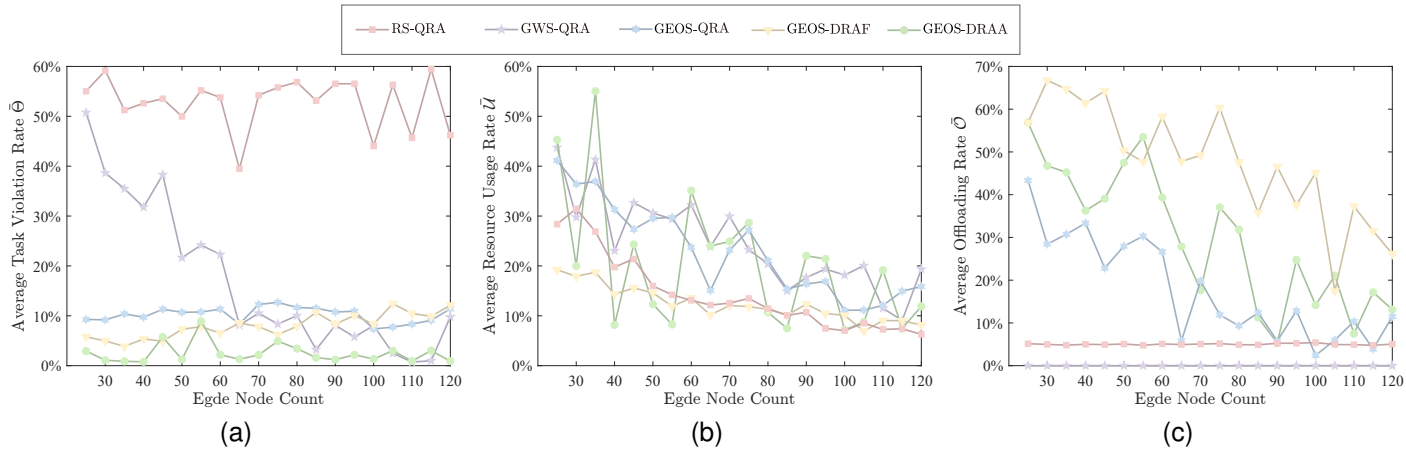


Fig. 13. Performance comparisons of (a) average task violation rate $\bar{\Theta}$, (b) average resource utilization rate $\bar{\mathcal{U}}$, and (c) average offloading rate $\bar{\mathcal{O}}$ on EUA dataset 1 with varying numbers of edge nodes.

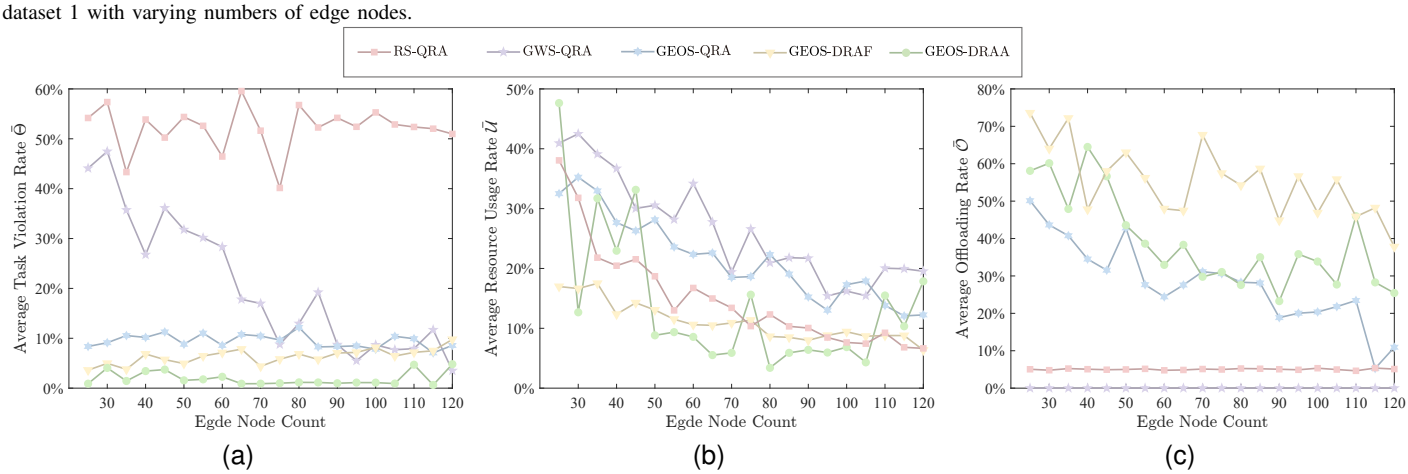


Fig. 14. Performance comparisons of (a) average task violation rate $\bar{\Theta}$, (b) average resource utilization rate $\bar{\mathcal{U}}$, and (c) average offloading rate $\bar{\mathcal{O}}$ on EUA dataset 2 with varying numbers of edge nodes.

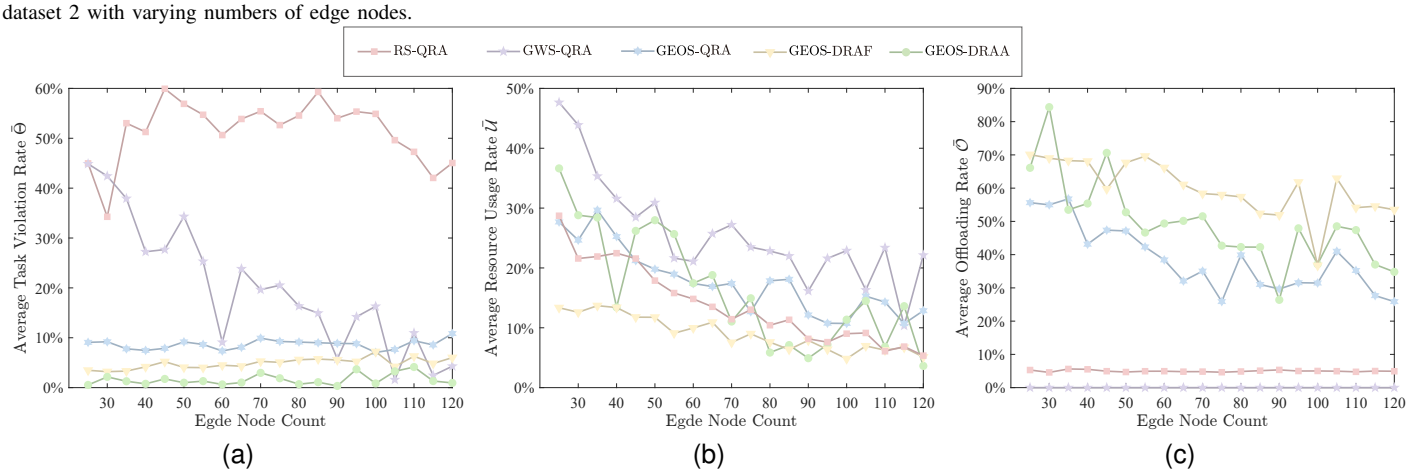


Fig. 15. Performance comparisons of (a) average task violation rate $\bar{\Theta}$, (b) average resource utilization rate $\bar{\mathcal{U}}$, and (c) average offloading rate $\bar{\mathcal{O}}$ on EUA dataset 3 with varying numbers of edge nodes.

cessing distributed across 125 ENs of varying computational capabilities. Each sensor generates detection requests every 3 seconds, producing video frames of 5-20MB depending on traffic density and environmental conditions. Processing requirements vary from 1000 MIPS/bit for basic vehicle detection to 3000 MIPS/bit for complex traffic pattern analysis. Critical tasks, such as congestion detection and incident monitoring, operate under strict deadlines of 5 seconds, while routine traffic flow analysis allows up to 10 seconds response time, these parameters match our experimental setup in Table I.

The streaming clustering algorithm processes incoming tasks using a spatial-temporal correlation mechanism. For instance, during peak hours, 50 concurrent detection tasks from a 500-meter radius are typically consolidated into 5-8 task blocks based on their geographic proximity and processing

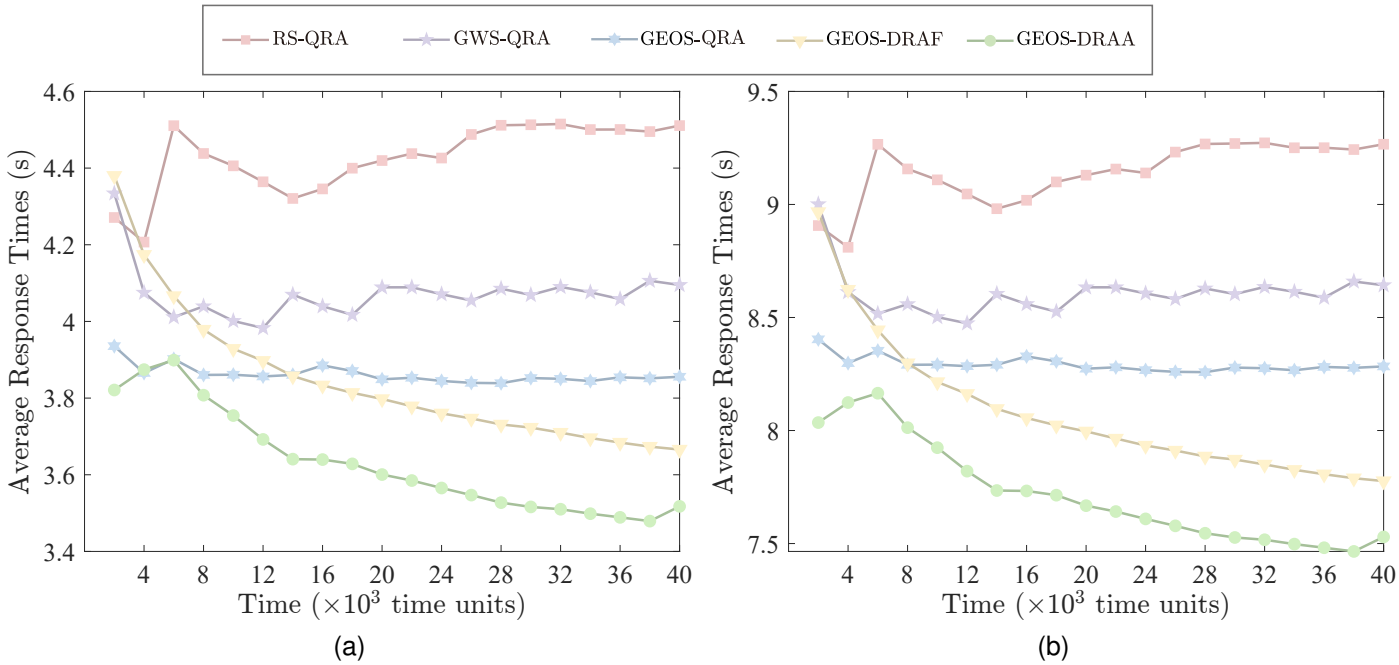


Fig. 16. Performance comparisons of average response times for traffic monitoring tasks: (a) critical incident detection and (b) routine flow analysis across different time periods

requirements. The weighted Voronoi diagram then partitions the CBD area into processing zones, with boundaries dynamically adjusting based on EN capacity utilization. Complex analysis tasks consuming over 2500 MIPS/bit are selectively offloaded to cloud resources when EN utilization exceeds their rated capacity. The sliding window optimization maintains a 15-second monitoring window, continuously adjusting resource allocation based on traffic patterns and computational demands, particularly during peak traffic periods.

The system has demonstrated robust performance metrics under varying load conditions. From a QoE perspective, GEOS-DRAA demonstrates superior service metrics. As shown in 8a, it maintains a high service reliability with 98.45% task completion rate during peak periods. Figure 16 further illustrates enhanced service responsiveness, achieving average response times of 3.63 s for critical tasks and 7.72 s for routine analysis, significantly outperforming the traditional RS-QRA queuing method which requires 4.43 s and 9.14 s for equivalent tasks. As illustrated in Figures 7c, 8c, and 9c, the resource utilization maintains a stable range of 20% 40% during peak periods, ensuring robust processing performance. These results validate the framework's effectiveness in managing large-scale tasks through geometric transformation and adaptive resource allocation mechanisms.

VII. Conclusion

This study initially examines the challenges in edge computing for smart city applications, where state-of-the-art methodologies fall short in effectively handling large-scale task scheduling, offloading evaluation, and efficient resource allocation for lightweight container virtualization under highly dynamic task requests as the number of ENs and tasks scale up.

To address these challenges, a comprehensive framework is proposed in this paper. The framework includes a geometrized task scheduling (stream clustering techniques, a task assignment approach based on regional partition, and a Tetris-like offloading mechanism) and a dynamic resource allocation strategy. The stream clustering techniques are employed to reduce the scale of dynamic task streams, while a regional partitioning approach is utilized to allocate resources efficiently across different areas. Furthermore, a Tetris-like offloading mechanism is introduced to facilitate offloading compute-intensive tasks to the cloud platform by assessing available resources at ENs, ensuring task finalization before deadlines. Moreover, the study presents a sliding window-based dynamic resource allocation strategy with adaptive window optimization to cope with the unpredictability and high concurrency of containerized task requests. The incorporation of task segmentation and advanced processing guarantees the timely completion of tasks.

The simulation results demonstrate that the proposed GEOS-DRAA enhances performance by reducing the average task deadline violation rate $\bar{\Theta}$ to only 4.72% of that of RS-QRA, 7.70% of that of GWS-QRA, 16.46% of that of GEOS-QRA, and 46.32% of the performance achieved by GEOS-DRAF when managing large-scale task scheduling. It shows the efficiency in dynamic resource allocation for compute-intensive tasks as well as in scenarios with random and high-concurrency task requests.

The framework is more adaptive for large-scale task scheduling. However, it might not be optimal for scenarios requiring ultra-low latency. When configuring resources dynamically, it is also necessary to consider rational container placement and the practical feasibility of task segmentation. Future work will focus on addressing these issues to further

reduce task execution time and meet low latency constraints. Additionally, the framework will be applied to other complex scenarios with a high density of sensors and terrains akin to smart cities, such as large-scale offshore wind farms and oil platform [36].

ACKNOWLEDGEMENT

This work was supported by National Key R&D Program of China (2022YFC3801100).

REFERENCES

[1] Statista, "Number of internet of things (iot) connected devices worldwide from 2019 to 2033," 2024.
[2] M. Chiang, S. Ha, F. Risso, T. Zhang, and I. Chih-Lin, "Clarifying fog computing and networking: 10 questions and answers," *IEEE Communications Magazine*, vol. 55, no. 4, pp. 18–20, 2017.
[3] E. Ahmed, A. Naveed, A. Gani, S. H. Ab Hamid, M. Imran, and M. Guizani, "Process state synchronization for mobility support in mobile cloud computing," in *2017 IEEE International Conference on Communications (ICC)*, pp. 1–6, IEEE, 2017.
[4] X. Chen, L. Jiao, W. Li, and X. Fu, "Efficient multi-user computation offloading for mobile-edge cloud computing," *IEEE/ACM transactions on networking*, vol. 24, no. 5, pp. 2795–2808, 2015.
[5] K. Gai, M. Qiu, H. Zhao, L. Tao, and Z. Zong, "Dynamic energy-aware cloudlet-based mobile cloud computing model for green computing," *Journal of network and computer applications*, vol. 59, pp. 46–54, 2016.
[6] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE internet of things journal*, vol. 3, no. 5, pp. 637–646, 2016.
[7] Y. Xu, L. Chen, Z. Lu, X. Du, J. Wu, and P. C. Hung, "An adaptive mechanism for dynamically collaborative computing power and task scheduling in edge environment," *IEEE Internet of Things Journal*, vol. 10, no. 4, pp. 3118–3129, 2021.
[8] X. Li, J. Wan, H.-N. Dai, M. Imran, M. Xia, and A. Celesti, "A hybrid computing solution and resource scheduling strategy for edge computing in smart manufacturing," *IEEE Transactions on Industrial Informatics*, vol. 15, no. 7, pp. 4225–4234, 2019.
[9] G. Zhang, F. Shen, N. Chen, P. Zhu, X. Dai, and Y. Yang, "Dots: Delay-optimal task scheduling among voluntary nodes in fog networks," *IEEE Internet of Things Journal*, vol. 6, no. 2, pp. 3533–3544, 2018.
[10] M. Molina, O. Muñoz, A. Pascual-Iserte, and J. Vidal, "Joint scheduling of communication and computation resources in multiuser wireless application offloading," in *2014 IEEE 25th Annual International Symposium on Personal, Indoor, and Mobile Radio Communication (PIMRC)*, pp. 1093–1098, IEEE, 2014.
[11] P. Mach and Z. Becvar, "Mobile edge computing: A survey on architecture and computation offloading," *IEEE communications surveys & tutorials*, vol. 19, no. 3, pp. 1628–1656, 2017.
[12] H. Li, X. Li, C. Sun, F. Fang, Q. Fan, X. Wang, and V. C. M. Leung, "Intelligent content caching and user association in mobile edge computing networks for smart cities," *IEEE Transactions on Network Science and Engineering*, vol. 11, no. 1, pp. 994–1007, 2024.
[13] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A survey on mobile edge computing: The communication perspective," *IEEE communications surveys & tutorials*, vol. 19, no. 4, pp. 2322–2358, 2017.
[14] C. Dou, N. Huang, Y. Wu, and T. Q. Quek, "Energy-efficient hybrid noma-fdma assisted distributed two-tier edge-cloudlet multi-access computation offloading," *IEEE Transactions on Green Communications and Networking*, vol. 7, no. 3, pp. 1234–1249, 2023.
[15] Z. Wang, G. Xue, S. Qian, and M. Li, "Campedge: Distributed computation offloading strategy under large-scale ap-based edge computing system for iot applications," *IEEE internet of things journal*, vol. 8, no. 8, pp. 6733–6745, 2020.
[16] X. Chen, H. Zhang, C. Wu, S. Mao, Y. Ji, and M. Bennis, "Performance optimization in mobile-edge computing via deep reinforcement learning," in *2018 IEEE 88th Vehicular Technology Conference (VTC-Fall)*, pp. 1–6, IEEE, 2018.
[17] Y. Chen, S. Deng, H. Ma, and J. Yin, "Deploying data-intensive applications with multiple services components on edge," *Mobile Networks and Applications*, vol. 25, pp. 426–441, 2020.
[18] Y. Chen, S. Deng, H. Zhao, Q. He, Y. Li, and H. Gao, "Data-intensive application deployment at edge: A deep reinforcement learning approach," in *2019 IEEE International Conference on Web Services (ICWS)*, pp. 355–359, IEEE, 2019.
[19] L. Zhou, Q. Sun, S. Ding, S. Han, and A. Wang, "A machine-learning-based method for ship propulsion power prediction in ice," *Journal of Marine Science and Engineering*, vol. 11, no. 7, p. 1381, 2023.
[20] T. Zhang and Z. Ren, "Restricted isometry property in wave buoy analogy and application to multispectral fusion," *IEEE Transactions on Intelligent Transportation Systems*, 2025.
[21] Y. Xu, J. Li, Z. Lu, J. Wu, P. C. Hung, and A. Alelaiwi, "Arvmec: Adaptive recommendation of virtual machines for iot in edge–cloud environment," *Journal of Parallel and Distributed Computing*, vol. 141, pp. 23–34, 2020.
[22] A. Xiao, Z. Lu, X. Du, J. Wu, and P. C. Hung, "Orhrc: Optimized recommendations of heterogeneous resource configurations in cloud-fog orchestrated computing environments," in *2020 IEEE International Conference on Web Services (ICWS)*, pp. 404–412, IEEE, 2020.
[23] D. Zhang, Z. Chen, L. X. Cai, H. Zhou, S. Duan, J. Ren, X. Shen, and Y. Zhang, "Resource allocation for green cloud radio access networks with hybrid energy supplies," *IEEE Transactions on Vehicular Technology*, vol. 67, no. 2, pp. 1684–1697, 2017.
[24] D. Zhang, Z. Chen, M. K. Awad, N. Zhang, H. Zhou, and X. S. Shen, "Utility-optimal resource management and allocation algorithm for energy harvesting cognitive radio sensor networks," *IEEE Journal on Selected Areas in Communications*, vol. 34, no. 12, pp. 3552–3565, 2016.
[25] C. Anderson, "Docker [software engineering]," *Ieee Software*, vol. 32, no. 3, pp. 102–c3, 2015.
[26] T. X. Tran, A. Hajisami, P. Pandey, and D. Pompili, "Collaborative mobile edge computing in 5g networks: New paradigms, scenarios, and challenges," *IEEE Communications Magazine*, vol. 55, no. 4, pp. 54–61, 2017.
[27] A.-C. Pang, W.-H. Chung, T.-C. Chiu, and J. Zhang, "Latency-driven cooperative task computing in multi-user fog-radio access networks," in *2017 IEEE 37th international conference on distributed computing systems (ICDCS)*, pp. 615–624, IEEE, 2017.
[28] A. P. Miettinen and J. K. Nurminen, "Energy efficiency of mobile clients in cloud computing," HotCloud'10, (USA), p. 4, USENIX Association, 2010.
[29] Q. Wu and R. Zhang, "Common throughput maximization in uav-enabled ofdma systems with delay consideration," *IEEE Transactions on Communications*, vol. 66, no. 12, pp. 6614–6627, 2018.
[30] Y. Wang, M. Sheng, X. Wang, L. Wang, and J. Li, "Mobile-edge computing: Partial computation offloading using dynamic voltage scaling," *IEEE Transactions on Communications*, vol. 64, no. 10, pp. 4268–4282, 2016.
[31] X. He and S. Wang, "Peer offloading in mobile-edge computing with worst case response time guarantees," *IEEE Internet of Things Journal*, vol. 8, no. 4, pp. 2722–2735, 2020.
[32] J. Wang, Y. Ren, W. Shi, M. Collu, V. Venugopal, and X. Li, "Multi-objective optimization design for a 15 mw semisubmersible floating offshore wind turbine using evolutionary algorithm," *Applied Energy*, vol. 377, p. 124533, 2025.
[33] P. Lai, Q. He, M. Abdelrazek, F. Chen, J. Hosking, J. Grundy, and Y. Yang, "Optimal edge user allocation in edge computing with variable sized vector bin packing," in *Service-Oriented Computing: 16th International Conference, ICSOC 2018, Hangzhou, China, November 12-15, 2018, Proceedings 16*, pp. 230–245, Springer, 2018.
[34] F. Liu, J. Huang, and X. Wang, "Joint task offloading and resource allocation for device-edge-cloud collaboration with subtask dependencies," *IEEE Transactions on Cloud Computing*, vol. 11, no. 3, pp. 3027–3039, 2023.
[35] G. Zou, Z. Qin, S. Deng, K.-C. Li, Y. Gan, and B. Zhang, "Towards the optimality of service instance selection in mobile edge computing," *Knowledge-Based Systems*, vol. 217, p. 106831, 2021.
[36] S. Wang and T. Moan, "Methodology of load effect analysis and ultimate limit state design of semi-submersible hulls of floating wind turbines: With a focus on floater column design," *Marine Structures*, vol. 93, p. 103526, 2024.