International Journal on Semantic Web and Information Systems
Volume 20 • Issue 1 • January-December 2024

Efficient Workflow Scheduling in Edge Cloud-Enabled Space-Air-Ground-Integrated Information Systems

Yunke Jiang
Aerospace Information Research Institute, China

Xiaojuan Sun
Aerospace Information Research Institute, China

ABSTRACT

To address the challenges posed by the dynamism, high latency, and resource scarcity in integrated air-space-ground hybrid edge cloud environments on task completion times and node load, we designed a task scheduling system for scenarios involving the transmission and processing of interdependent tasks. This system integrates a graph neural network with attention mechanism and deep reinforcement learning. Specifically, we employ a graph encoder to extract features from DAG tasks and resources. Task scheduling solutions for dynamic environments are then generated using attention mechanism-equipped graph decoder, which are subsequently optimized based on performance metrics through the use of an Advantage Actor-Critic algorithm. Experimental results indicate that this algorithm performs well in terms of completion time and node load balance across tasks with different workflow structures, demonstrating its adaptability to highly dynamic edge cloud environments.

KEYWORDS

Space-Air-Ground integrated Networks, Graph Convolutional Network, Advantage Actor–Critic, Dynamic Task Scheduling

In recent years, significant advancements have been made in satellite manufacturing, spot beam antennas, and laser transmission, which have made satellites, particularly low Earth orbit (LEO) satellites, more economical and more miniaturized and given them a higher throughput (Yu et al., 2021). These developments have substantially accelerated the growth of space-air-ground integrated networks (SAGINs). As a novel network architecture, SAGIN achieves a comprehensive and three-dimensional coverage by seamlessly integrating satellite networks in space, aerial platforms such as UAVs (unmanned aerial vehicles), and traditional terrestrial wireless and wired networks (Cui et al., 2022). This integration provides a broader platform for edge computing, an emerging distributed computing paradigm that shifts data processing from data centers closer to the data source, thereby reducing data transmission latency and enhancing task execution speed (Shi et al., 2016). By applying edge computing within the SAGIN environment, it is possible to decentralize data processing across a composite network of terrestrial, aerial, and satellite networks. This setup reduces data transmission delays, improves data processing efficiency and speed, and ensures data security (Hamidi & Mohammadi, 2006; Nilchi et al., 2008) and privacy (Wang et al., 2019). Implementing edge computing at various levels within the SAGIN facilitates the global provision of seamless services and responsiveness, offering benefits such as low cost, flexible networking, and real-time data sensing

DOI: 10.4018/IJSWIS.345935

and processing, It also provides a suitable platform for applications that are latency-sensitive and require large amounts of data analysis (Liu et al., 2018; Zhao et al., 2022).

However, edge computing within the context of SAGIN still faces numerous challenges. First, because of the inclusion of terrestrial, aerial, and satellite layers, network connections between nodes—particularly those involving aerial platforms and satellite networks—exhibit a high degree of dynamism (Zhu & Jiang, 2023). Second, devices across these different layers are heterogeneous, necessitating intelligent task scheduling and resource allocation strategies to efficiently utilize computational resources, storage, and bandwidth. Additionally, applications within SAGIN are highly sensitive to latency and structurally complex. Such applications, including disaster monitoring and emergency response, as well as smart city initiatives, require real-time processing and analysis of large volumes of multimodal data, which may include, but are not limited to, structured data, unstructured text, images, audio, and video. These applications also feature complex execution processes with interdependent modules, where the sequence of execution significantly impacts the overall workflow performance (Farid et al., 2023). In summary, given the high dynamism of the network, the heterogeneity of the devices, and the latency sensitivity and complexity of the applications, task scheduling algorithms become particularly crucial. Edge cloud task scheduling algorithms in SAGIN are designed to intelligently distribute and manage tasks, storage resources, and bandwidth among heterogeneous and unstable computing resources to optimize resource use and enhance data processing efficiency.

A key issue in edge computing task scheduling (Liu et al., 2016) is that the determination of the execution site for tasks typically allows for selection among local systems, cloud data centers, or edge servers. This dilemma is addressed primarily through optimization algorithms and machine learning solutions (Kar et al., 2023). In terms of edge cloud scheduling algorithms, traditional optimization techniques in edge cloud scheduling focus on management plane decisions, while machine learning is applied predominantly to control plane decisions. Traditional optimization scheduling algorithms in edge cloud environments strive to optimize objective functions, seeking optimal or near-optimal solutions under various constraints. These include convex optimization methods (Wang et al., 2014), mixed-integer nonlinear programming (Zhang et al., 2021), game-theoretic approaches (Guo & Liu, 2018), and heuristic methods (Soltani et al., 2017), such as Monte Carlo tree search (Yu et al., 2020), and swarm intelligence algorithms (Chen et al., 2023; Zhang et al., 2022). Represented by heuristic algorithms, these optimization methods are easy to implement, have low computational complexity, and can quickly find near-optimal solutions; hence they are widely used. In cloud environments, Ajmal et al. (2021) combined genetic algorithms and ant colony algorithms. Partitioning tasks and their associated computational resources (commonly virtual machines and other cloud computing assets) into smaller groups effectively reduces the complexity of search operations. They introduced a new mutation strategy by swapping gene values to increase diversity and accelerate the algorithm's convergence speed and search efficiency, aiming to reduce response times and minimize workflow runtime. Alakbarov (2022) proposed a mobile cloud computing task allocation and resource optimization scheme based on cloudlets. This approach utilizes a differential evolution algorithm to optimize task allocation, selecting appropriate cloudlets to minimize energy consumption and communication latency during task execution, thereby enhancing system performance. Bisht and Vampugani (2021) modified the min-min algorithm by segmenting it into two phases: cost-aware minimization and resource load-balancing. This adaptation allows for the minimization of energy consumption and delay through optimized load distribution when operations are constrained by deadlines. For highly dynamic environments, Hajvali et al. (2023) proposed a lightweight hybrid scheduling cluster method designed to manage workflow tasks within mobile computing resources, significantly improving task execution success rates and resource utilization. Zheng et al. (2022) designed a priority-based level heuristic scheduling algorithm for uncertain structured DAG task scheduling problems, effectively reducing the computational overhead and runtime while maintaining performance. Baburao et al. (2021) introduced an enhanced particle swarm optimization (PSO) algorithm capable of achieving rational resource allocation by removing long-term inactive tasks

from RAM, thereby reducing task waiting times and addressing network congestion issues. Chen et al. (2024) used triangular fuzzy numbers to describe the dynamic changes in server computational power and fluctuating bandwidth connections under dynamic conditions. They integrated a quadratic penalty function (QPF) with the PSO algorithm, significantly enhancing convergence speed, reducing execution costs under deadline constraints, and mitigating the impact of network conditions on task execution in uncertain edge environments.

In recent years, task scheduling algorithms based on machine learning have gained widespread application due to their ability to autonomously improve operations by interacting with the environment online or learning from existing data offline, thus better adapting to the dynamics of edge cloud environments (Srikanth & Geetha, 2023). For highly dynamic settings, Ai et al. (2023) employed a deep Q-network-based reinforcement learning algorithm to predict current channel state information from system history and used convex optimization methods for dynamic task offloading. For scenarios involving multiple users and tasks, Elgendy et al. (2021) introduced the concept of computation task caching, in which completed task codes are cached on edge servers. They proposed a reinforcement learning approach that defines the state space on the basis of transitions between different states and all potential solutions, achieving optimal solutions through the DQN algorithm, effectively reducing overhead on mobile devices. Niu et al. (2023) framed the competitive task scheduling problem for mobile users in fixed heterogeneous edge computing systems as a noncooperative stochastic game, solved using a multiagent proximal policy optimization (PPO) algorithm, and extended the use of meta-reinforcement learning to nonstationary heterogeneous edge computing systems. Sheng et al. (2021) proposed a REINFORCE policy-based task scheduling algorithm for edge computing, significantly enhancing task satisfaction. For a multiaccess edge computing environment, Sun et al. (2023) implemented task scheduling by modeling it as a graph state transition problem. By integrating the DROO algorithm, they achieved an enhancement in convergence speed as well as an improvement in task response times, addressing the reliance on precise mathematical models in scheduling strategies and considering the relationships between devices, which enhances adaptability and scalability. To effectively handle uncertainties in the dynamic IoT (internet of things) edge cloud environment, Wang et al. (2024) introduced a DRL (deep reinforcement learning algorithm) that combines policy gradient and temporal difference learning, optimizing server load balance and application response times. Xiu et al. (2023) emphasized the adaptability of algorithms, improving the focus of DRL algorithms on specific environments by using meta-reinforcement learning to reduce the time required to adapt to new environments, achieving the goals of shortening task scheduling times and enhancing server utilization. Li et al. (2024) proposed an adaptive PPO algorithm that makes decisions regarding task prioritization and offloading actions to enhance the scalability and flexibility of the vehicle edge computing (VEC) paradigm. These studies indicate that DRL algorithms, because of their high adaptability and scalability, are becoming a trend in task scheduling algorithms for edge computing environments.

Compared to traditional terrestrial edge computing, applications of space-air-ground integrated edge computing have a broader scope. Mao et al. (2021) designed a SAGIN utilizing UAVs, in which the tasks, resource allocation, and positioning of UAVs are jointly optimized. This design employs an enhanced alternating optimization algorithm to minimize computational delays between devices. Cao et al. (2022) proposed an edge cloud architecture for the SAGIN for Internet of Vehicles (SAGIN-IoV) and developed an optimization model tailored to service demands, which ameliorates the integration challenges posed by hardware discrepancies and communication issues among various communication systems within the integrated network environment. Li and Chen (2023) designed a task offloading algorithm for mobile edge computing (MEC) within a space-air-ground integrated environment that handles system uncertainties through risk-aware and distributionally robust optimization, significantly enhancing quality of service (QoS). Yoo et al. (2023) introduced a cache-assisted edge computing system for SAGIN; the system leverages LEO satellites and UAVs, employing the Dinkelbach method and successive convex approximation (SCA) algorithm to optimize caching issues during

the offloading process, thereby maximizing system energy efficiency. Mao et al. (2023) developed a two-tiered DRL model for addressing UAV task assignment and route planning, incorporating an interactive training strategy that enhances training outcomes by allowing upper and lower models to interact during the training process. In the context of satellite internet environments, Lee et al. (2024) introduced a mixed-integer linear programming model based on the concept of space-time networks. This model establishes a unified scheduling for missions and communications of satellite constellations, solving the problem of maximizing the use of multiple satellites in environments with multiple satellites and ground stations. Zhang et al. (2019) utilized dynamic network virtualization technology and collaborative computing offloading models to facilitate parallel computations within high-speed satellite terrestrial networks, integrating network resources to reduce user-perceived latency. Tang et al. (2021) introduced a LEO satellite network covering hybrid cloud and edge computing with a three-tier computational architecture and proposed a distributed algorithm based on the alternating direction method of multipliers (ADMM) to reduce complexity while meeting computational capacity and coverage constraints of LEO satellites, effectively lowering the overall energy consumption of terrestrial users. Han et al. (2020) introduced modifications to the HEFT algorithm to accommodate link variations between satellite networks, employing a dynamic priority queue to process data on orbiting satellites. This enhancement effectively reduced task processing times and increased the parallelism of data handling. Wang et al. (2021) considered varying task sensitivities, with latency, bandwidth, and connection duration as resource allocation factors, and introduced an advanced K-means algorithm to guide resource segmentation in satellite edge computing. Chai et al. (2023) introduced a training method combining attention mechanisms and PPO to achieve cost-effective joint offloading in multitask systems. Li et al. (2023) designed a hierarchical scheduling method that integrates DQN networks with ant colony algorithms, optimizing algorithm response times and task completion rates in multi-satellite systems. The dynamic nature of vehicular networks also provides insights for task scheduling in space-air-ground integrated edge computing environments, as the network topology of vehicular networks, unstable by nature, has many similarities with that of space-air-ground integrated edge computing systems. Addressing frequent vehicle joins and departures during system operation, Shi et al. (2020) proposed a distributed vehicle-to-vehicle (V2V) task offloading scheme that considers vehicle mobility and availability as bases for selecting service vehicles on the basis of link duration and vehicle status, maximizing the average latency-aware utility of offloading tasks over a period. Liu et al. (2023) introduced a dynamic task scheduling system for vehicular clouds based on DRL, using multi-head graph attention mechanisms to extract features from each subtask within the task topology and incorporating nonuniform neighborhood sampling to ensure that the scheduling algorithm adapts well to varying topologies. Wang et al. (2023) proposed a proactive task migration strategy to vehicles, using gated recurrent units (GRU) and graph convolutional layers to extract features from spatial road traffic and multitemporal scale driving data, achieving optimal utility migration decisions with Lyapunov optimization.

Below is a comparison of task scheduling algorithms in the context of space-air-ground integrated edge cloud environments:

In highly dynamic environments, DRL scheduling algorithms are becoming the mainstream trend. The integration of DRL with transformers and graph neural networks has enhanced its scalability, making it suitable for serialized or graph-structured data. This integration offers substantial advantages in terms of reducing task execution latency and optimizing energy pre-costs. In the context of SAGINs, Mao et al. (2023) demonstrated that compared to heuristic algorithms, DRL provides significant benefits in decision space and computational efficiency, while avoiding exhaustive search and complex heuristic rules (Kar et al., 2023). However, scheduling algorithms for hybrid edge cloud workflows in space-air-ground integrated environments still faces several challenges: (1) Most existing task scheduling algorithms are designed for specific scenarios, such as satellite internet, UAVs, and vehicular networks, which limits their scalability and general applicability. (2) The majority of current algorithms for space-air-ground integrated edge cloud task scheduling focus

Table 1. Recent Related Work on Integrated Air-Space-Ground Task Scheduling

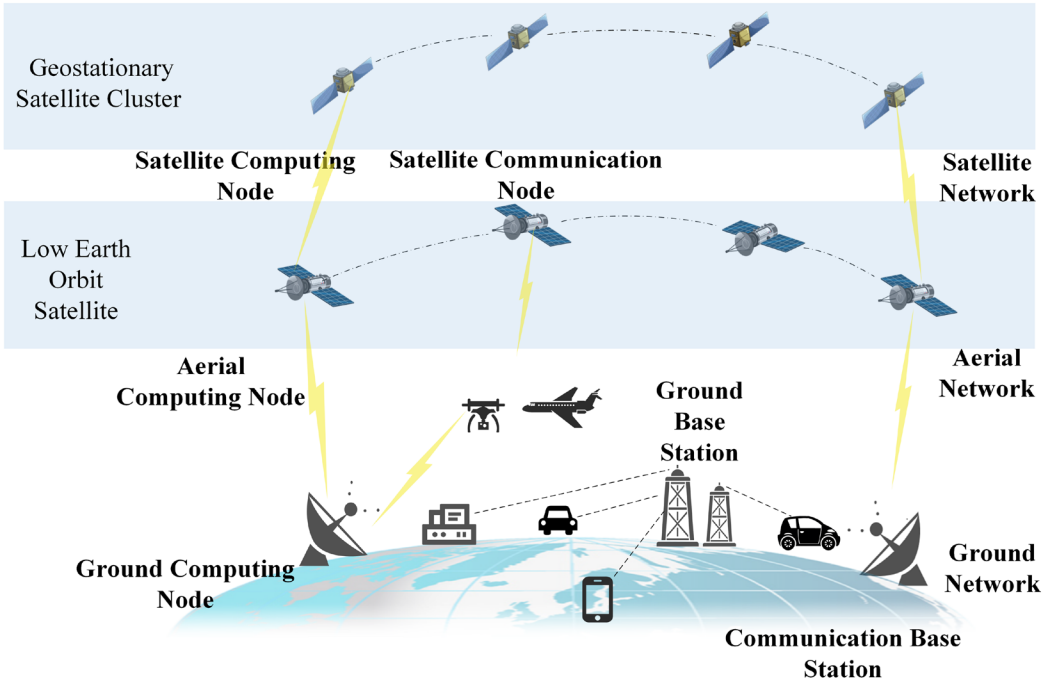
Related works		Our paper	2020		2021	2022	2023					
			Han et al.	Shi et al.	Tang et al.	Cao et al.	Yoo et al.	Wang, C. et al.	Chai et al.	Mao et al.	Li, Z. et al.	Liu et al.
System Architecture	SAGIN	√				√	√				√	
	Satellite IoT architecture		√		√				√			
	UAVs								√	√		
	Internet of Vehicles			√		√		√				√
	Task with dependencies	√	√						√			√
Algorithm	Heuristic algorithms		√			√						
	Optimization				√		√				√	
	Attention mechanism	√						√	√	√		√
	Deep reinforcement learning	√		√				√	√	√		√
	Graph neural network	√						√				√
Performance analysis	Latency minimization	√	√	√	√	√						√
	Cost minimization					√		√	√		√	
	Load balance	√										
	Task execution success rate			√		√		√				
	Resource utilization				√	√	√					

primarily on overall task offloading, with little consideration for dependencies among tasks. (3) In space-air-ground integrated environments, certain nodes are prone to resource node overload, leading to increased response times and energy consumption, reduced system security, and other adverse outcomes. Current algorithms predominantly aim to optimize energy consumption, with less attention given to the load on resource nodes.

To address these issues, this paper employs DRL techniques, designing an advantage actor-critic (A2C) algorithm with a graph attention mechanism to solve the edge cloud workflow scheduling problem, aiming to optimize task completion times and device load balancing. The contributions of this paper are as follows:

(1) Tasks and resources are represented as graph structures, which offer excellent universality. We propose a scheduling algorithm based on a graph encoder-decoder structure that effectively captures and utilizes task characteristics and environmental changes through an attention mechanism, enhancing scheduling efficiency and adaptability to dynamic environments.
(2) We introduce a task ordering algorithm based on a dynamic priority queue, which considers not only the topological order of tasks but also the impact of network dynamics on task execution, suitable for scenarios with complex subtask relationships and high parallelism.
(3) For the multi-objective optimization problem, we select task completion times and device load balancing as optimization metrics. On the basis of the A2C algorithm, we have designed a reward function with varying weights to optimize these metrics, offering solutions to issues such as task waiting and increased energy consumption caused by resource node overload. Experimental results validate that our algorithm outperforms others in terms of task completion time while avoiding node overload.

Figure 1. Space-Air-Ground Integrated Network Architecture



The structure of this paper is as follows: Section 2 delineates the problem definition and modeling approach for workflow scheduling in edge cloud-enabled space-air-ground integrated information systems. Section 3 provides the overall framework and detailed design of the task scheduling algorithm. Optimization strategies based on the A2C scheduling algorithm are discussed in Section 4. Section 5 presents some experimental details and demonstrates the results. Finally, Section 6 concludes the paper, and Section 7 discusses the limitations of the algorithm and proposes future work directions.

SYSTEM MODEL AND PROBLEM FORMULATION

The SAGIN system is comprised of satellite edge computing nodes, aerial computing nodes, and terrestrial computing nodes, along with the networks that interconnect them. The task scheduling process includes the generation of tasks and task information by users, the creation of task offloading decisions based on scheduling algorithms and task information, the execution of these offloading decisions, executing tasks locally or uploading them to appropriate computing nodes, and returning the results to users. Initially, we define the resources and task models within the task scheduling of the SAGIN. Subsequently, we decompose the task offloading process and define the metrics, and finally, we establish optimization objectives.

Satellite Mobile Edge Computing System

In the SAGIN, as illustrated in **Figure 1**, a multitask MEC system typically comprises terrestrial nodes, aerial nodes, and satellite nodes. The satellite nodes are divided into LEO and high-Earth orbit (HEO) clusters. Terrestrial and aerial nodes are interconnected by ground and air network segments, respectively. In the satellite network, the LEO cluster is positioned in orbits closer to the Earth's surface, which results in lower communication latencies and provides certain computational

capabilities. These nodes fulfill both computational and communication functions. However, they exhibit high mobility, unstable network connections, and limited computational resources. In contrast, the HEO cluster remains relatively fixed in relation to the Earth, featuring a stable network topology and broad coverage.

Satellite computing nodes, aerial computing nodes, and terrestrial computing nodes together constitute the structure of the space-air-ground integrated edge computing system. This structure represents a comprehensive, multilevel collaborative computing system that integrates computing resources from terrestrial, aerial, and space layers to form a cross-domain, efficient data processing network. Within this architecture, cloud computing nodes are at the core of the system, typically deployed in data centers with robust data processing and storage capabilities. They are responsible for executing complex data analysis tasks and long-term data storage, while also managing and deploying strategies across the entire network. Edge computing nodes are positioned closer to the data sources and may include terrestrial base stations, routers, aerial drones, airships, or space-based satellite platforms. These nodes have the capability for data processing and temporary storage, allowing them to preprocess and initially analyze collected data to reduce data transmission volumes, lower network latency, and support real-time or near-real-time application requirements. Terminal computing nodes represent the network's endpoints, such as smartphones, sensors, and other IoT devices that directly generate data. Some devices are also capable of basic data processing and filtering to further optimize data flow and enhance system responsiveness. This layered design of the space-air-ground integrated edge computing architecture not only achieves global service coverage but also enhances data processing efficiency and speed, meeting the demands for latency sensitivity and computational power across various application scenarios.

Task Model and Resource Model

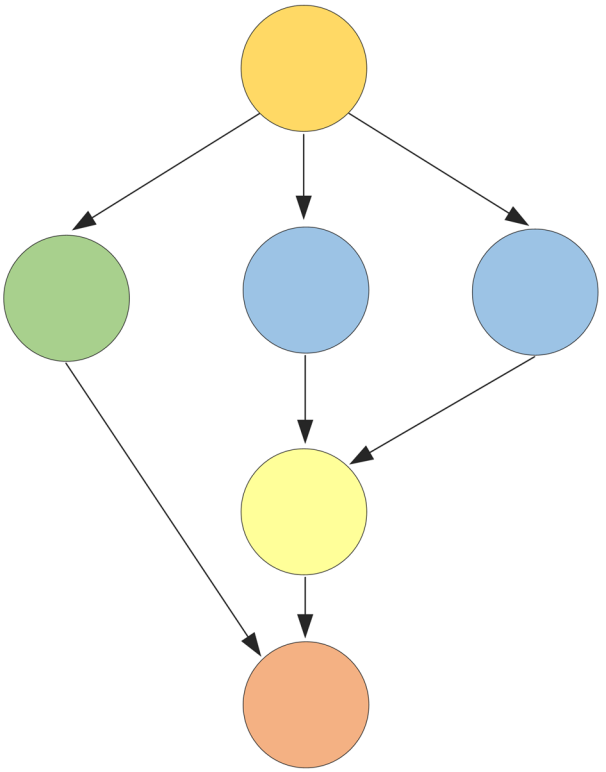
To establish task and resource models, tasks and resources with features are represented in graph data form, facilitating feature extraction by GCNs (graph convolutional networks). Tasks are typically represented using a directed acyclic graph (DAG), denoted as $G_{task} = (V_{task}, E_{task})$, as shown in Figure 2. V_{task} contains n nodes, each representing a subtask. Each task $v_i \in V_{task}$ is the smallest unit placed on a resource node, characterized by two parameters: $v_{i.cpu}$ (the required CPU clock cycles) and $v_{i.mem}$ (the occupied memory). The set of predecessor tasks for task v_i is denoted as $pred(v_i)$, and the set of successor tasks as $succ(v_i)$. The edges $e_{ij} \in E_{task}$ represent the data volume transmitted between two subtasks (v_i, v_j). All nodes with zero in-degree are called entry tasks, denoted as v_{entry}, and those with zero out-degree are called exit tasks, denoted as v_{exit}.

The resource cluster model is represented by a fully connected undirected graph $G_{res} = (V_{res}, E_{res})$. Each node $u_m \in V_{res}$ represents a computing node in the satellite MEC system. The node u_m is characterized by four features: $u_{m.cpu}$ represents the existing CPU computational capacity of the node, $u_{m.cpu.total}$ denotes the total computational capacity of the computing node, $u_{m.mem}$ indicates the current memory resources of the node, and $u_{m.mem.total}$ represents the total memory resources of the node. Different computing node layers in the space-air-ground integrated edge computing architecture have varying computational and storage capabilities. The edge $d_{mn} \in E_{res}$ represents the communication bandwidth between nodes (u_m, u_n).

Task Offloading Model

To shorten task completion time and maintain load balance across devices, part of the tasks are scheduled to other qualified computing nodes for execution, known as task offloading. A task generally consists of multiple subtasks. Initially, on the basis of the operational status of satellite edge cloud devices and the data volume of the task, the readiness time of the device and the data transmission time of the subtask are calculated. This calculation determines the earliest start time of the task on the device. Subsequently, by estimating the task execution time through the task load and device capability, the earliest completion time of the task is calculated.

Figure 2. Task Workflow



1) Node Readiness Time: The earliest readiness time *Avaliable*(*n*) of a computing node where a task resides indicates the completion time of the most recently assigned task to node u_n. After completing this task, node u_n can start executing new tasks.
2) Task Transmission Time:

$$TT_{i,m;j,n} = \begin{cases} c_{ij}(t), m \neq n \\ 0, m = n \end{cases} \tag{1}$$

$$c_{ij}(t) = \frac{e_{ij}}{d_{mn}(t)} \tag{2}$$

Here, $TT_{i,m;j,n}$ represents the data transmission time when subtask v_i and its successor task v_j are allocated to devices u_m and u_n, respectively. When $m = n$, the transmission time is 0. The term $d_{mn}(t)$ denotes the transmission bandwidth between devices at time t, and e_{ij} represents the volume of data transmitted. Thus, $c_{ij}(t)$ is the time required for task data transmission at time t.

3) The Earliest Start Time of the Task:

$$EST(v_j,u_n)=\max\left\{Avaliable(n),\max_{v_i\in pred(v_j)}(AFT_i+TT_{i,m;j,n})\right\},v_i,v_j\in V_{task} \tag{3}$$

Here, AFT_i denotes the actual finish time of all predecessor subtasks $v_i \in pred(v_j)$ of subtask v_j. Consequently, the earliest start time $EST(v_j,u_n)$ for subtask v_j on node u_n is determined by the greater value between the node's readiness time $Avaliable(n)$ and the time when all data output from preceding tasks arrives at node u_n.

4) The Earliest Completion Time of the Task:

$$EFT(v_j,u_n)=EST(v_j,u_n)+\frac{v_{j.cpu}}{u_{n.cpu}} \tag{4}$$

The earliest completion time of a task is the sum of its earliest start time and the task execution time. Here, $v_{j.cpu}$ denotes the computational resource requirement of task v_i, and $u_{n.cpu}$ represents the computational capacity of the device executing that task.

Problem Formulation and Optimization Constraints

Aiming to maintain load balance across devices and to minimize the overall task completion time, we define an optimization cost function for satellite MEC task scheduling:

$$COST = \alpha Makespan + (1-\alpha)Load \tag{5}$$

$$Makespan = \max_{v_j\in V_{task},u_n\in V_{res}}\left\{\xi_{j,n}EFT_{j,n}\right\} \tag{6}$$

$$Load = \frac{1}{2}load_{cpu}+\frac{1}{2}load_{mem} \tag{7}$$

$$load_{cpu} = \sqrt{\frac{1}{|V_{res}|}\sum_{n=1}^{|V_{res}|}(A_{n,j}^{cpu}-\frac{\sum_{m=1}^{|V_{res}|}A_{m,i}^{cpu}}{|V_{res}|})^2} \tag{8}$$

$$load_{mem} = \sqrt{\frac{1}{|V_{res}|}\sum_{n=1}^{|V_{res}|}(A_{n,j}^{mem}-\frac{\sum_{m=1}^{|V_{res}|}A_{m,i}^{mem}}{|V_{res}|})^2} \tag{9}$$

$$A_{n,j}^{cpu} = \frac{\frac{v_{j.cpu}}{u_{n.cpu}}-\frac{v_{j.cpu}}{u_{n.cpu.total}}}{\frac{v_{j.cpu}}{u_{n.cpu.total}}} \tag{10}$$

$$A_{n,j}^{mem} = \frac{u_{m.mem.total}-u_{m.mem}+v_{i.mem}}{u_{m.mem.total}} \tag{11}$$

Constraint Conditions:

$$\sum_{u_n \in \mathcal{V}} \xi_{j,n} = 1 \tag{12}$$

$$\xi_{j,n} \in \{0,1\} \tag{13}$$

$$EST_{j,n} \geq EFT_{i,m} \tag{14}$$

The cost function*COST*is a weighted combination of task completion time and load balancing degree(5). *Makespan* is the task completion time, which is the maximum completion time of all subtasks in the task graph (6). Here, $\xi_{j,n}$ is a binary variable, meaning that when task v_j is executed on device u_n, $\xi_{j,n} = 1$; otherwise, $\xi_{j,n} = 0$. The load is the sum of the standard deviations of all computing CPU resource loads and memory resource loads(7) (8) (9). The smaller the standard deviation, the more balanced the resource load among nodes. Wherein, $A_{n,j}^{cpu}$ represents the CPU load when the current subtask v_j is executed on node u_n (10). $A_{n,j}^{cpu}$is calculated by comparing the execution time of the task under the current computing power of the node $\frac{v_{j,cpu}}{u_{n,cpu}}$ with the execution time under full computing power $\frac{v_{j,cpu}}{u_{n,cpu,total}}$. $\frac{\sum_{m=1}^{|V_m|} A_{m,j}^{cpu}}{|V_{res}|}$ denotes the average CPU resource load across all computing nodes under the current task. The squared sum of the difference between the node's resource load and the average load $(A_{n,j}^{cpu} - \frac{\sum_{m=1}^{|V_m|} A_{m,j}^{cpu}}{|V_{res}|})^2$ is averaged and then square-rooted to obtain the standard deviation of resource load, which is the cost of CPU load balancing. $A_{n,j}^{mem}$ represents the memory load brought by the current subtask v_j when executed on node u_n(11), calculated by comparing the overall load of node u_m with the current load. The standard deviation of memory load is calculated in the same way as the standard deviation of CPU load.

The constraint conditions represent the following:

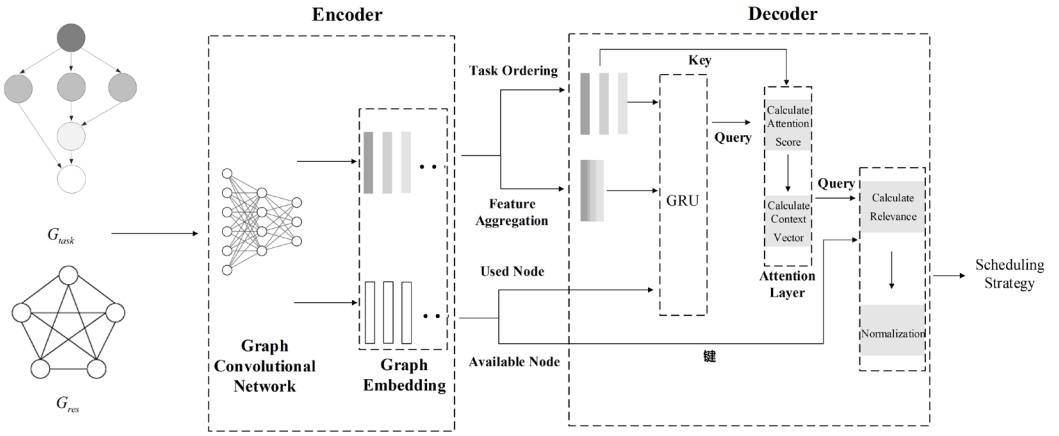
Each subtask is assigned to only one computing node (12).
The allocation variable $\xi_{j,n}$ is constrained to be a binary variable, $\xi_{j,n} \in (0,1)$ (13).
The start time of each task must be after the completion time of its subtasks (14).

GRAPH ATTENTION ENCODER-DECODER TASK SCHEDULING ALGORITHM

We model the space-air-ground integrated edge computing task scheduling problem as an optimization problem and establish optimization objectives. To find the optimal solution in a short time, we designed a graph encoder-decoder model with a task priority queue, based on the GCN and according to the task and resource models. This model can extract features of tasks and resources, better reflecting the high dynamism of the computing node network topology. It determines the matching results between tasks and nodes based on the attention mechanism.

We utilize an encoder-decoder model to generate a task scheduling scheme. Given a set of$G_{task}$and G_{res},the scheduling scheme provides a placement strategy $\mathcal{P}$:$V_{task} \rightarrow V_{res}$, which is the mapping between subtasks and computing nodes. Initially, in the encoder part, graph embeddings are used to extract features and associations of subtasks, encoding G_{task} and G_{res} into node vectors. Subsequently, the decoder, based on a recurrent neural network, employs an attention mechanism to optimize the scheduling scheme for each subtask in order of priority (21), generating the optimal position for each task. **Figure 3** illustrates the overall architecture of the encoder-decoder model.

Figure 3. Encoder-Decoder Structure



Graph Encoder

For the task graph structure and resource graph, we employ a GCN (Kipf & Welling, 2017) to encode the information in the graph into a set of embedding vectors, representing the intrinsic features of the nodes and their interconnections. This process involves inputting the attributes of the graph's nodes and outputting a matrix of embedding vectors, where each row represents the feature embedding of a node.

For the G_{task} with a DAG structure, the intrinsic features of nodes v_{task} are first transformed into fixed-dimension vectors using a fully connected layer. Then, the information from the predecessor and successor neighbor nodes of v_{task} is aggregated respectively. The predecessor neighbor nodes are denoted as $N_u(v)$, and the successor neighbor nodes as $N_d(v)$. For each node $u \in N_u(v)$, its embedding at the k-th step is e_u^k. By concatenating e_u^k with the edge weights e_{ij}, which represent the data volume of task transmission, and multiplying with the weight matrix $\mathbf{W}_1^{(\mathbf{up})}$, then applying the tanh activation function, we obtain the predecessor embedding $e_u^{(up)}$ of node u at the k-th step:

$$e_u^{(up)} = \tanh(\mathbf{W}_1^{(\mathbf{up})}[e_u^k : e_{ij}]) \tag{15}$$

Subsequently, the embedding of the predecessor neighbor information of node v is calculated on the basis of its current embedding and the average of the embeddings of all its predecessor neighbors:

$$e_v^{(up)} = \tanh(\mathbf{W}_2^{(\mathbf{up})}\left[e_v^k \cdot \frac{\sum_{u \in N_u(v)} e_u^{(up)}}{N_u(v)}\right]) \tag{16}$$

Similarly, the k-th step embedding $e_v^{(down)}$ of v_{task} is calculated after incorporating the information of its successor neighbor nodes. In the next iteration, these two embeddings are concatenated:

$$e_v^{k+1} = \left[e_v^{(up)} : e_v^{(down)}\right] \tag{17}$$

Finally, an aggregation function is used to aggregate the node features. The output after the aggregation operation is a one-dimensional vector representing the aggregated features of all nodes and edges in the graph. This vector is used to calculate the hidden state in the decoder, where

$$e_{G_{task}} = \text{Aggregate}(\left\{ e_v \middle| v_i \in V_{task} \right\}) \tag{18}$$

Since G_{res} is a fully connected undirected graph without predecessor and successor nodes, the connections between nodes represent the communication bandwidth between them at time t, having edge characteristics. The neighbor node embeddings with edge weights are updated using these edge features:

$$e_u^{(neighbor)} = ReLU(\mathbf{W}_1^{(\mathbf{res})}\left[e_u^k : d_{(u_m,v_n)}\right]) \tag{19}$$

The features of the nodes are updated, where the features of the neighbor nodes include edge information:

$$e_v^{k+1} = \text{ReLU}(\mathbf{W}_2^{(\mathbf{res})}\left[e_v^k : \frac{\sum_{u \neq v} e_u^{(neighbor)}}{|V_{task}| - 1}\right]) \tag{20}$$

After k iterations, the embeddings of all nodes are computed, and the information from the task and resource graphs can be obtained by feeding the embedding of each task into the fully connected layer and the max pooling layer. The output of the encoder is the embedding vector of the nodes, where the embedding of the task v_i on G_{task} is e_{v_i}, and the embedding of the computing nodes on G_{res} is e_{u_m}.

Task Ordering Algorithm Based on Dynamic Priority Queue

To address the limitations of topology-based task ordering algorithms and adapt to task scheduling scenarios with complex subtask relationships and high parallelism, and to select the best task scheduling strategy given the highly dynamic state of the satellite network, we propose a task ordering algorithm based on a dynamic priority queue. This algorithm determines the priority of tasks on the basis of the predecessor task level, execution cost, and transmission cost of the node, and in the graph decoder, processes the nodes in this priority order by choosing the appropriate processor or satellite node. The specific process involves obtaining the priority of all predecessor tasks R_i, the average execution duration $\overline{AEC_j}$ across all nodes, and the average latency $\overline{c_{ij}(t)}$ of data transmission from the predecessor task to all available nodes. Summing these attributes for each predecessor task, the maximum value is selected as the priority R_j for that task:

$$R_j = \max_{v_i \in pred(v_j)} \left\{ R_i + \overline{AEC_j} + \overline{c_{ij}(t)} \right\} \tag{21}$$

$$\overline{AEC_j} = \frac{\sum_{n=1}^{|V_{res}|} \frac{v_{j.cpu}}{u_{n.cpu}}}{|V_{res}|} \tag{22}$$

$$\overline{c_{ij}(t)} = \frac{\sum_{v_i \in pred(v_j)} \sum_{u_n \in V_{res}} \frac{e_{ij}}{d_{mn}(t)}}{|pred(v_j)| \cdot |V_{res}|} \tag{23}$$

R_j comprehensively considers the task dependencies, the computation cost of the task, and the transmission cost on the basis of the current network connection bandwidth. Here, $\overline{AEC_j}$ represents

the average computation cost of subtask v_j on each computing node. $\overline{c_{ij}(t)}$ denotes the average transmission latency at time t for the data output of predecessor subtask v_i from device u_m to the executing node u_n of v_j.

Graph Attention Mechanism Graph Decoder

In the design of the graph decoder structure, we improved upon the traditional topological sorting algorithm (Huang et al., 2021), which is focused solely on task execution order. Instead, we adopted the task ordering algorithm on the basis of a dynamic priority queue described in the preceding subsection. This approach ensures that task ordering not only considers the sequential dependencies of task topology but also takes into account the network connections of nodes and their execution durations for scheduling tasks.

Initially, tasks are sorted according to their priority R_j (21). The graph decoder generates scheduling actions for each task in the order of their priority. In this context, the embedding of task v_i on G_{task} is denoted as e_{v_i}, and the embedding of the computing node on G_{res} is e_{u_m}. The objective of the algorithm is to place $v_i \in V_{task}$ on the computing node $u_m \in V_{res}$. The final placement scheme generated by the decoder is represented as $\mathscr{P}$:

$$p(\mathscr{P}\Big| G_{task}, G_{res}) = \prod_i p(u_{v_i}\Big| \mathscr{R}^{(up)}(v_i), G_{task}, G_{res}) \tag{24}$$

p is a conditional probability, representing the likelihood of a scheduling scheme $\mathscr{P}$ given the task and resource graph, where $\mathscr{R}^{(up)}(v_i)$ is the set of embeddings for the target devices of all subtasks scheduled before the priority of v_i. u_{v_i} represents the node where task v_i is placed; thus $p(u_{v_i}\Big| \mathscr{R}^{(up)}(v_i), G_{task}, G_{res})$ is the probability of each task's being placed on the corresponding computing node under the scheduling scheme of higher-priority tasks.

Next, to capture the long-term dependencies between decisions and tasks, we use GRU to update the current state representation h_{v_i} of task v_i. This state representation captures the features and state representation of the previous task, as well as the placement of all tasks with a higher priority than the current task, to ensure that the decoder considers sufficient context information when deciding the placement of each task. Here, $e_{v_{i-1}}$ represents the embedding of the previous task in the task ordering, and $h_{v_{i-1}}$ is the state representation of the previous task:

$$h_{v_i} = GRU\_Cell(e_{v_{i-1}}, h_{v_{i-1}}, \mathscr{R}^{(up)}(v_i)) \tag{25}$$

The GRU updates the current subtask's state by merging information from the previous subtask, the states of the network's hidden layers $e_{v_{i-1}}$ and$h_{v_{i-1}}$, and the target node information $\mathscr{R}^{(up)}(v_i)$ of the already scheduled subtasks. This merging enables the model to remember long-term task dependencies and historical scheduling decisions.

Next, the attention score $e_{ij} = h_{v_i} e_{v_j}$ is computed, which represents the relevance of the task v_i to the current placement context h_{v_i} through dot product. If two vectors are more 'proximate' in high-dimensional space, their dot product is larger, indicating that the task v_j has a greater influence on the placement decision of task v_i. The e_{ij} is then passed into a *softmax* layer to be transformed into a distribution probability α_{ij}, quantifying the importance of each task for the current placement decision. The final context vector c_i is the concatenation of the current state representation h_{v_i} and the weighted task embeddings. It contains both the current state information and relevant information from other tasks, providing the decoder with a comprehensive context:

$$c_i = [h_{v_i} : \sum_j \alpha_{ij} e_{v_j}] \tag{26}$$

The context vector c_i aggregates information relevant to the current task v_i, including task dependencies and historical task scheduling decisions, thereby establishing a comprehensive representation for subsequent decision making. α_{ij} is a weight that reflects the relative influence of other tasks' decisions on the current task's decision when scheduling v_i. α_{ij} is dynamically calculated on the basis of the current task state and allows for adjustments in the importance of different tasks over time as task statuses change. $\sum_j \alpha_{ij} e_{v_j}$ computes a weighted sum that integrates the significance of all other tasks regarding task v_i. Finally, by concatenating the current task state h_{v_i} with the importance levels of other tasks to v_i, a complete view of task v_i is provided.

Then, the relevance between the query q_{c_i} and each key k_j is calculated on the basis of the self-attention mechanism, where q_{c_i} represents the current state and k_j denotes the embedding of each available computing node:

$$q_{c_i} = \mathbf{W^Q} c_i, k_j = \mathbf{W^K} e_{u_j}, v_j = \mathbf{W^V} e_{u_j} \tag{27}$$

In this context, $\mathbf{W^Q}$, $\mathbf{W^K}$, and $\mathbf{W^V}$ are model parameters that are learned from training data. They transform the original embeddings into a suitable space for computing attention.

The attention score is given by (28). This is achieved by computing the dot product of the query vector $q_{c_i}^T$ and the key k_j, which assesses the degree of match between the current task v_i and the computational resource node e_{u_j}. d_k represents the dimension of the computational node embeddings, and $\sqrt{d_k}$ is a scaling factor used to prevent the dot product from becoming excessively large in higher dimensions, which could lead to gradient vanishing or exploding. The activation function constrains the range of the attention scores to between [-1, 1], and a constant C is used to further adjust the range of scores to obtain the final attention scores $\omega_{(c_i)j}$.

$$\omega_{(c_i)j} = C \cdot \tanh\left(\frac{q_{c_i}^T k_j}{\sqrt{d_k}}\right) \tag{28}$$

Finally, by applying a softmax transformation to $\omega_{(c_i)j}$, we calculate the probability $p(u_{v_i}, c_i)$ that task v_i is scheduled to a specific computing node under the context of c_i, $p(u_{v_i}, c_i)$ effectively transforms attention scores into a probability distribution, enabling the model to select the most suitable node on the basis of the current context. Here, c_i includes a weighted summary of information from the current task v_i as well as from other tasks related to task v_i:

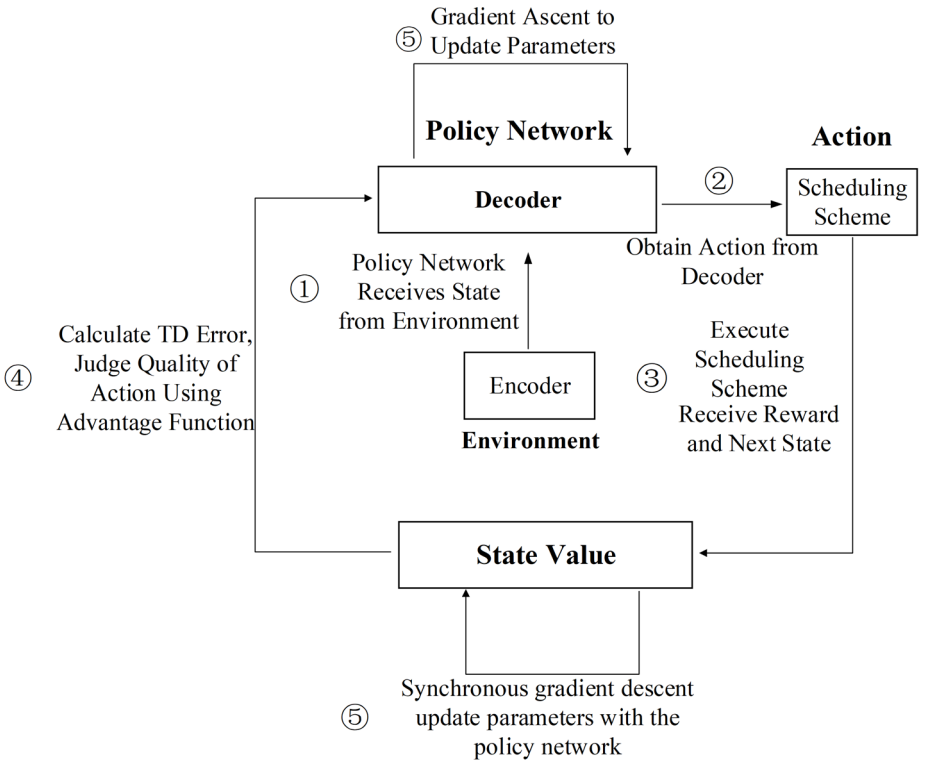
$$p_m = p_\theta(u_{v_i} = u_m \Big| c_i) = \frac{e^{\omega_{(c_i)m}}}{\sum_j e^{\omega_{(c_i)j}}} \tag{29}$$

$p_\theta(u_{v_i} = u_m \Big| c_i)$ represents the probability that task v_i is scheduled to computing node u_m. Here, $e^{\omega_{(c_i)m}}$ is the value of node u_m after being transformed by the exponential function. The denominator $\sum_j e^{\omega_{(c_i)j}}$ represents the normalization of scores across all computational nodes, ensuring that the sum of probabilities for all nodes equals 1. After normalization, by predicting the position of each task in the order of task priorities, the model will generate the final scheduling scheme $\mathcal{P}$.

OPTIMIZATION STRATEGY FOR SCHEDULING ALGORITHM BASED ON A2C

We employ the A2C algorithm (Chen et al., 2019) to train the parameters of the GCN so that the task scheduling schemes generated in the graph decoder (see the previous subsection) are more aligned with the objectives of minimizing task completion time and balancing the load. The overall

Figure 4. A2C-Based Task Scheduling Optimization Process



cost of the task scheduling system is minimized through the interaction between the A2C algorithm, the attention mechanism, and the environment.

Task Scheduling Optimization Method

The method for optimizing task scheduling includes steps such as state awareness, action acquisition, action execution, advantage gaining, policy network updating, and value network updating. The process begins with state awareness: collecting the encodings of tasks and resources after priority sorting. Then, action acquisition takes place: the node encoding of the task, the execution node encoding of the higher-priority subtasks, and the node encoding of the available resources are input in sequence into the decoder with an attention mechanism. The decoder provides a current task scheduling scheme through the action probability distribution of the A2C policy network. Following this step, action execution occurs: the reward for the action and the state for the next moment are obtained by executing the task scheduling scheme. Finally, the value network evaluates the quality of the action on the basis of the reward and the state value, and the parameters in the decoder are updated to optimize the task scheduling decision. The specific process is illustrated in **Figure 4**:

To optimize the scheduling strategy with the A2C algorithm, the task scheduling process is established as a Markov decision process (MDP) composed of state space, action space, and rewards. The MDP model can be denoted as $M = (S, A, P, R, \gamma)$, where S represents the state space of the problem, A represents the action space, and P denotes the state transition probability, defined as $P(s_{t+1} | s_t, a_t)$, which signifies the probability of transitioning from state s_t to state s_{t+1} given action a_t. R represents the reward function, and γ indicates the discount factor. The MDP process for the satellite internet task scheduling algorithm is depicted as follows:

(1) State:

$$s^{(t)} = \left\{ e_{G_{task}}, e_{v_i}(t), e_{u_m}(t), \mathscr{R}^{(up)}(v_i) \right\} \tag{30}$$

Here, $e_{v_i}(t)$ represents the embedding vector of subtasks at the current moment, formed after encoding by the encoder. $\mathscr{R}^{(up)}(v_i)$ denotes the embeddings of the target nodes for scheduling of higher-priority subtasks, and $e_{u_m}(t)$ is the embedding vector formed after the encoding of the currently available computing node resources.

(2) Action:

At each time step, we need to determine the computing node for the placement of the subtask on the basis of the system state $s^{(t)}$. For the current subtask v_t, the action $a^{(t)}$ is defined as. Here, $a^{(t)} = 1$ indicates that the current subtask is processed locally, and $a^{(t)} \in \left\{2, \cdots, |V_{res}|\right\}$ signify that the current subtask v_t is assigned to other computing nodes.

(3) Reward Function:

The reward function represents the outcome of executing action $a^{(t)}$ in state $s^{(t)}$. In the task scheduling process, the arrival of tasks or changes in network topology may lead to updates in operation selection and state transitions. In this scheduling algorithm, the reward function is the weighted sum of the task completion delay (6) and the load balancing degree of the devices executing the task (7).

$$R(s_t, a_t, s_{t+1}) = -(\alpha Makespan + (1-\alpha) Load) = -COST \tag{31}$$

A2C Algorithm

A2C is an improvement on the actor-critic algorithm. It replaces the policy gradient in the policy network with a baseline-enhanced policy gradient. Specifically, it uses an advantage function $A(s_t, a_t) = Q(s_t, a_t) - v(s_t)$ instead of the original action-value function. The advantage function reflects the superiority of taking action a_t under state s_t relative to the average value $v(s_t)$ of all possible actions in that state, indicating the quality of the action.

The improvement in the advantage function can enhance learning efficiency while reducing variance, thus preventing overfitting. A2C consists of two parts: the policy network, which outputs actions on the basis of the action probability distribution and updates parameters based on the policy gradient, and the value network, which evaluates actions using the state-value function and updates parameters on the basis of the temporal-difference (TD) algorithm.

(1) State Value Function (Critic):

First, the state value function is defined as follows: under the current state s_t, the expected total return obtained by adopting policy π. Our objective is to maximize this expected total return:

$$V_\pi(s_t) = \mathbb{E}_{A_t \sim \pi(\cdot|s_t;\boldsymbol{\theta})}[\mathbb{E}[R_t + \gamma \cdot V_\pi(S_{t+1})]] \tag{32}$$

After approximation, the TD target y_t is obtained, representing the estimation made by the value network at time $t+1$ for $V_\pi(s_t)$, where γ is the discount factor:

$$y_t = r_t + \gamma v(s_{t+1};\omega) \tag{33}$$

Then, using the TD algorithm, the loss function is calculated. For the state value function, the TD error is defined as follows:

$$\delta_t = v(s_t;\omega) - y_t = v(s_t;\omega) - (r_t + \gamma v(s_{t+1};\omega)) \tag{34}$$

The loss function is then as follows:

$$L(\omega) = \frac{1}{N}\sum_{n=1}^{N}\delta_t^2 \tag{35}$$

where N is the batch size, and parameters ω are updated using gradient descent to minimize the loss function:

$$\omega \leftarrow \omega - \alpha \cdot \nabla_\omega L(\omega) \tag{36}$$

(2) Policy Function (Actor):

The policy function is $\pi(a|s;\theta)$, which controls the agent to make actions. Therefore, the objective of the policy network is to increase the probability of those actions with a positive advantage function through gradient ascent, optimizing the performance of the policy. The baseline-enhanced policy gradient in the A2C algorithm is defined as follows:

$$\begin{aligned} g(s_t,a_t,\theta) &= \nabla_\theta \ln\pi(a_t|s_t;\theta)\cdot A(s_t,a_t) \\ &= \nabla_\theta \ln\pi(a_t|s_t;\theta)\cdot (r_t + \gamma v(s_{t+1};\omega) - v(s_t;\omega)) \end{aligned} \tag{37}$$

The term $g(s_t,a_t,\theta)$ represents the gradient direction of policy $\pi(a_t|s_t;\theta)$ when action a_t is taken in state s_t. Here, $r_t + \gamma v(s_{t+1};\omega) - v(s_t;\omega)$ is an approximation of $A(s_t,a_t) = Q(s_t,a_t) - v(s_t)$, and the policy gradient is optimized through gradient ascent:

$$\theta \leftarrow \theta + \alpha g(s_t,a_t,\theta) \tag{38}$$

Parameters θ are updated through gradient ascent to increase the probability of actions that yield higher rewards. Here, α is the learning rate.

In summary, the training process of the A2C algorithm is as follows:

The Robustness of the GAT-A2C Algorithm in Dynamic Environments

During the training process, where the algorithm interacts with the environment, it first generates embeddings for the current environment's task graph G_{task} and resource graph G_{res} as and using a graph encoder. At each timestep, the algorithm receives encoded features $e_{v_i}(t)$ from the current subtask pending scheduling, features $e_{u_m}(t)$ from available nodes, and features $\mathcal{R}^{(up)}(v_i)$ from nodes that have a higher priority than the current subtask. Both resource and task features are updated at

Table 2. A2C Training Process

Algorithm 1 Training GAT_A2C
Input: state
output: placement $\pi(a\|s;\theta)$
Instantiate network parameter vectors ω and θ
Initialize batch size N
For each episode **do**:
for time step t=1,2,…**do**
obtain s_t by G_{task} and G_{res} embedding
perform tasks scheduling action a_t according to policy
get r_t and s_{t+1} from environment
get TD target $y_t = r_t + \gamma v(s_{t+1};\omega)$
calculate TD error $\delta_t = v(s_t;\omega) - y_t$
update the policy network $\theta \leftarrow \theta + \alpha g(s_t, a_t, \theta)$
update the value network $\omega \leftarrow \omega - \alpha \cdot \nabla_\omega L(\omega)$
$s \leftarrow s_{t+1}$
End for
End for

each timestep to ensure that the parameters input into the algorithm reflect the most current task and resource statuses.

Subsequently, in the task ordering algorithm based on a dynamic priority queue, tasks are prioritized on the basis of the precedence of the node's tasks, execution costs, and transmission costs. This process enables a more scientific ordering of parallel tasks, accommodating complex subtask relationships and high levels of task parallelism.

Then, in the attention mechanism graph decoder segment, the algorithm first inputs the collected state into a GRU neural network. The GRU considers the state of the previous task and the scheduling information of higher-priority tasks when updating the task state. This recursive-state updating enables the algorithm to continuously adapt to changes during task execution. Subsequently, by calculating attention scores between the current state and task embeddings, and merging them into a context vector c_t, the algorithm focuses on matching the global information of tasks with resource node information at each decision making step to generate the task scheduling decision $\pi(a|s;\theta)$. This decision, determined by the state representation and the current network parameters θ, is then executed in the environment to enact the scheduling plan and receive real-time feedback $R(s_t, a_t, s_{t+1})$ from the environment, which is used to update the algorithm's parameters. Afterward, the environmental state is updated for the next round of scheduling.

EXPERIMENTAL RESULTS

Cluster System and Parameter Configuration

To simulate the space-air-ground integrated edge computing environment, we established a computing cluster on Google Cloud Platform (GCP), consisting of 9 nodes: 5 terminal nodes, 3 edge nodes, and 1 cloud service center. The cloud service center node was configured with 8 vCPUs and

32 GB of memory, edge nodes with 4 vCPUs and 8 GB of memory, and terminal nodes with multiple 2 vCPUs and 4 GB of memory each. To simulate different network segments in the space-air-ground integrated edge computing environment, we set up 3 virtual private networks.

To test the performance of the scheduling algorithm in a satellite edge cloud environment, we selected Montage and CyberShake tasks from the Pegasus workflow management tool (Deelman et al., 2015) for testing: Montage is a toolkit for reprojecting images into a common coordinate system and blending them into seamless astronomical mosaics. It comes in four scales: Montage_25, Montage_50, Montage_100, and Montage_1000, involving steps like reprojection, background rectification, and image merging. CyberShake is used for processing and analyzing seismic simulation data to produce more accurate seismic ground motion predictions. It includes steps such as source generation, wavefield generation, ground motion calculation, and hazard analysis. This task workflow comes in three scales: CyberShake_30, CyberShake_50, and CyberShake_100.

The model training dataset was created using a Python simulator, comprising 1,000 task execution flowcharts and 500 resource topology graphs. The features of tasks and resources were developed on the basis of the features of G_{task} and G_{res} as mentioned in Section 3, with initial features randomly assigned to each graph vertex. The resource graph set included both homogeneous and heterogeneous devices, with each training sample composed of workflow topology and resource topology.

The network structure hyperparameters were defined as follows: The number of iterations K for graph embedding in both the task and resource graphs was set to 2, and the lengths of task embeddings and resource embeddings were set to 128. The batch size N was established at 20. A two-layer RNN network based on GRU, with each layer containing 256 hidden units, was utilized. The learning rate α was 0.0001, employing the Adam optimizer for gradient descent. The discount factor for returns was set to 0.95. The weight values for completion latency and load balancing were set at 0.5, and the number of training epochs is 1,500.

Evaluation of Algorithm Performance in a Clustered Environment

To validate the performance of the GAT-A2C method in terms of task completion time and device load balancing, the Montage_25, Montage_50, Montage_100, Montage_1000, and CyberShake_30, CyberShake_50, CyberShake_100 tasks were run in the aforementioned space-air-ground integrated edge computing environment using greedy algorithms, heterogeneous earliest finish time (HEFT) algorithm (Topcuoglu et al., 2002), and the Satellite-based dynamic priority list scheduling (SDPLS) algorithm (Han et al., 2020). The task completion times and load balancing degrees of the four algorithms were recorded, and a comparison was made with the GAT-A2C algorithm. The offloading strategies for the comparison algorithms were as follows:

1) greedy Algorithm: All tasks were offloaded to the cloud service center for processing.
2) HEFT Algorithm: Tasks were first sorted according to priority; then each task was scheduled at the lowest offloading cost.
3) SDPLS Algorithm: Tasks in the queue were sorted on the basis of priority, selecting the computing node with the minimum completion time as the mapping node for the subtasks.

Before running the test tasks, the GAT-A2C model was trained, and the reward curve was as follows:

The results show that in the Montage_workflow task performance test, as illustrated in **Figure. 6**, the GAT-A2C algorithm significantly outperformed the other three algorithms in terms of completion time. In the Montage_workflow_1000 task, the task completion times for the greedy algorithm, HEFT algorithm, SDPLS algorithm, and GAT-A2C algorithm were 2167.42, 2079.68, 2061.10, and 1829.24 seconds, respectively. Compared to these, the GAT-A2C algorithm showed improvements of 15.63%, 12.03%, and 11.25%, respectively. In the CyberShake_100 task (**Figure. 7**), the task completion times for the greedy algorithm, HEFT algorithm, SDPLS algorithm, and GAT-A2C algorithm were 836.32,

Figure 5. Training Process Rewards

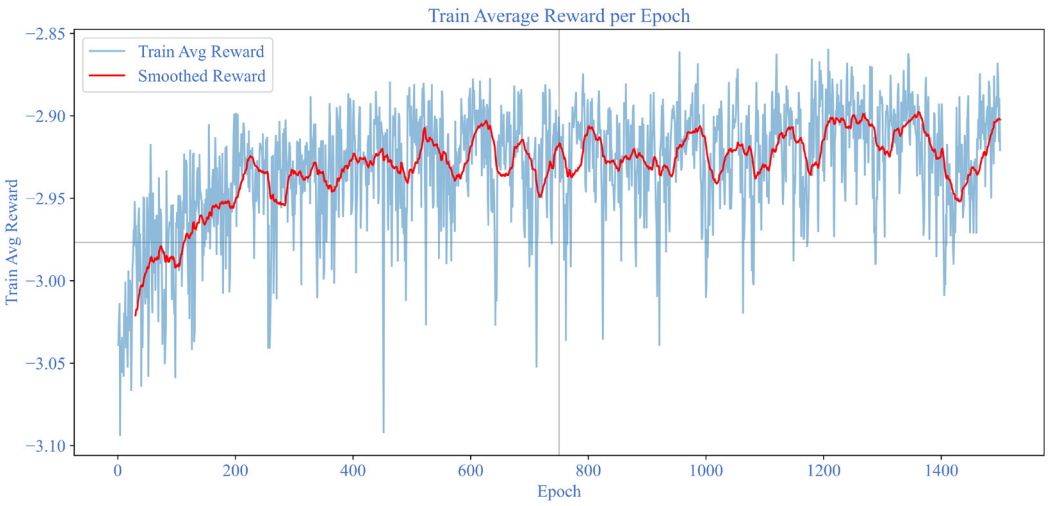
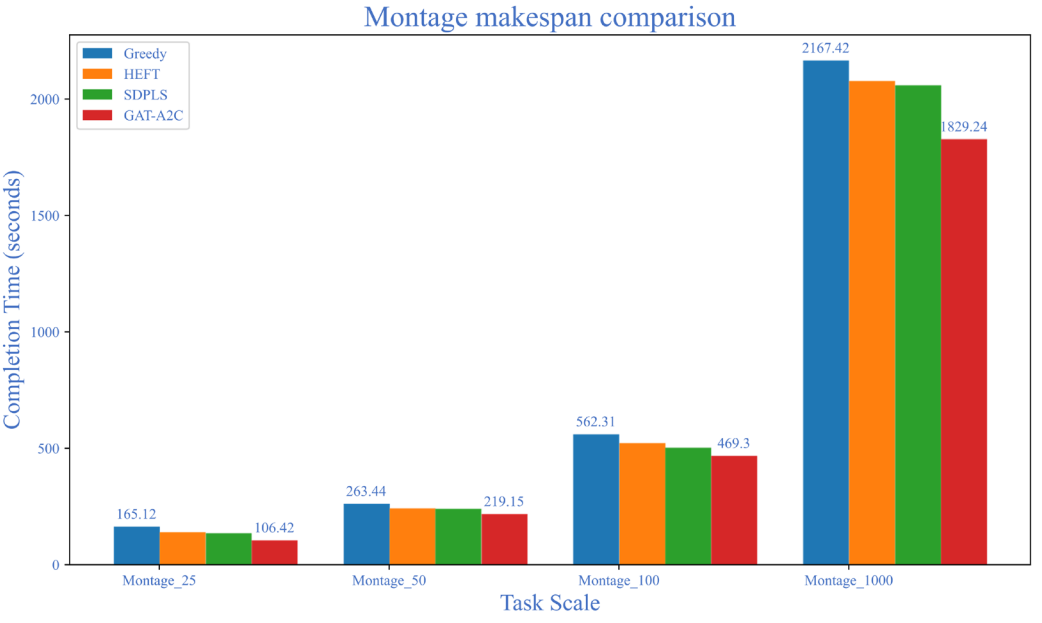


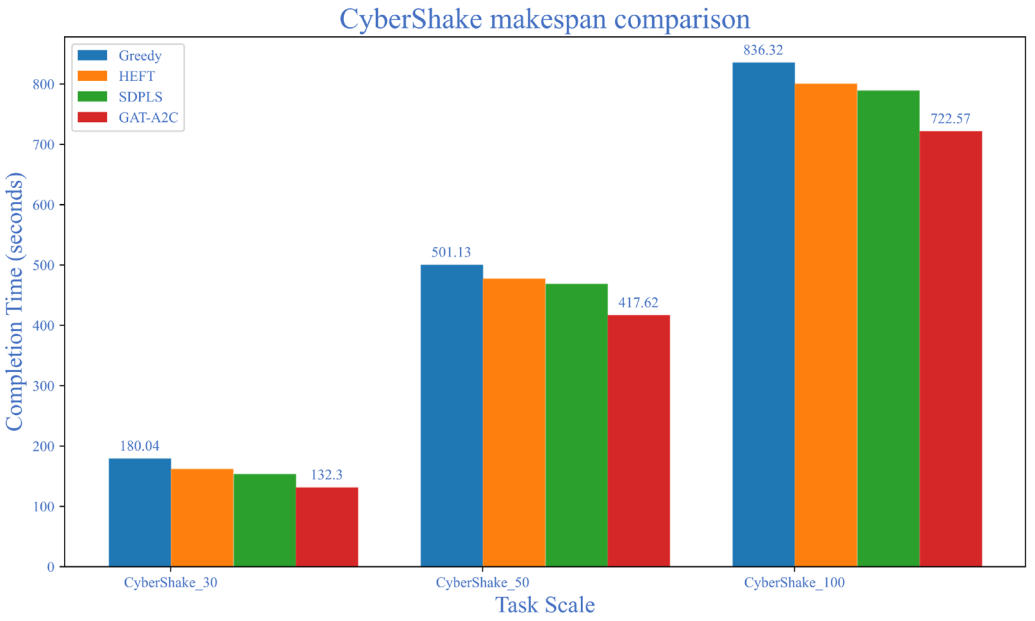
Figure 6. Montage Task Completion Time



801.36, 790.09, and 722.57 seconds, respectively. The GAT-A2C algorithm improved by 13.60%, 9.83%, and 8.55% compared to these algorithms.

According to the experimental results, the greedy algorithm offloads all tasks to a terrestrial cloud service center for execution. This algorithm has low complexity and can quickly determine the processing location and sequence for each task. However, transmitting each task to the cloud center may lead to network congestion, resulting in longer transmission delays and task waiting times. The algorithm's performance is also influenced by network conditions and lacks effective solutions when

Figure 7. CyberShake Task Completion Time



network disruptions occur, making the greedy algorithm's execution time the shortest among the four algorithms studied.

The HEFT algorithm accounts for dependencies between tasks and the computational capabilities of different nodes, thus outperforming the greedy algorithm. Its limitation lies in being a static scheduling algorithm, focusing on scheduling in scenarios where task characteristics, system resources, and workload information are known beforehand. In such scenarios, node resources continually change with task execution, and task arrival times are not known in advance. The HEFT algorithm does not interact with the environment until all tasks have been scheduled, resulting in lower performance compared to other algorithms.

The SDPLS algorithm incorporates network characteristics and updates current network changes through the resource monitoring tool, Cloud Monitoring, after a task scheduling cycle. However, its drawback is that each scheduling decision is based on the network and resource conditions available before task scheduling begins, and it cannot interact in real time with the environment during the subtask scheduling process. Environmental information is updated only after all subtasks have been scheduled, and before the next task's scheduling begins. Thus, it is suitable for task scheduling in multilayered network environments with varying bandwidths but has lower adaptability to the environment compared to DRL algorithms.

In the GAT-A2C algorithm, there is notable adaptability to the heterogeneity and dynamism of the space-air-ground integrated edge computing environment. This algorithm effectively combines the graph structure's capability to represent environmental conditions with the adaptability of DRL. By using monitoring tools that input current network bandwidth and resource utilization, constructed as graph structures, it allows for real-time acquisition of the environment's state at each subtask scheduling instance: features $e_{v_i}(t)$ of the current subtask, features $\mathscr{R}^{(up)}(v_i)$ of the scheduling target node for higher-priority subtasks, and features $e_{u_m}(t)$ of the currently available computing node resources. The representation of the state provides a comprehensive description of the current tasks and resources. Additionally, $\mathscr{R}^{(up)}(v_i)$, as historical environmental information input into the network, helps to prevent resource conflicts and competition caused by scheduling high-priority tasks. In

Figure 8. Montage_1000 Task Node CPU Load

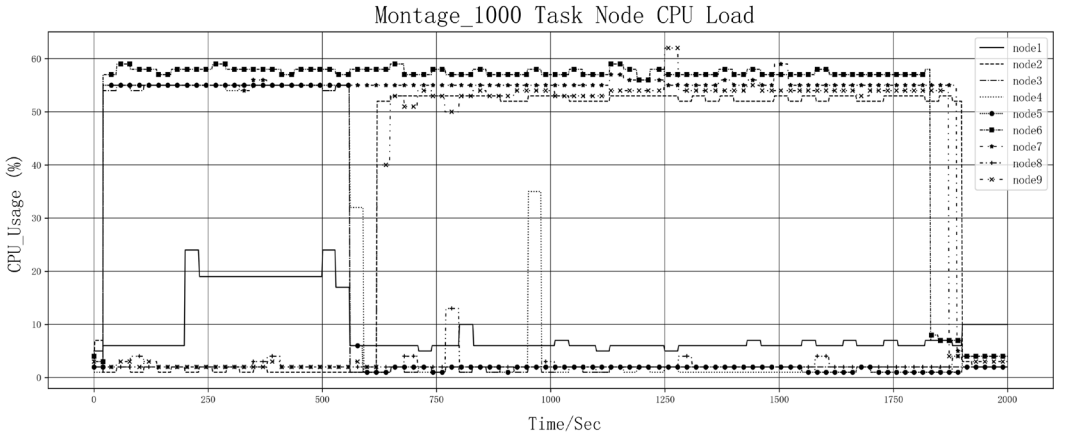
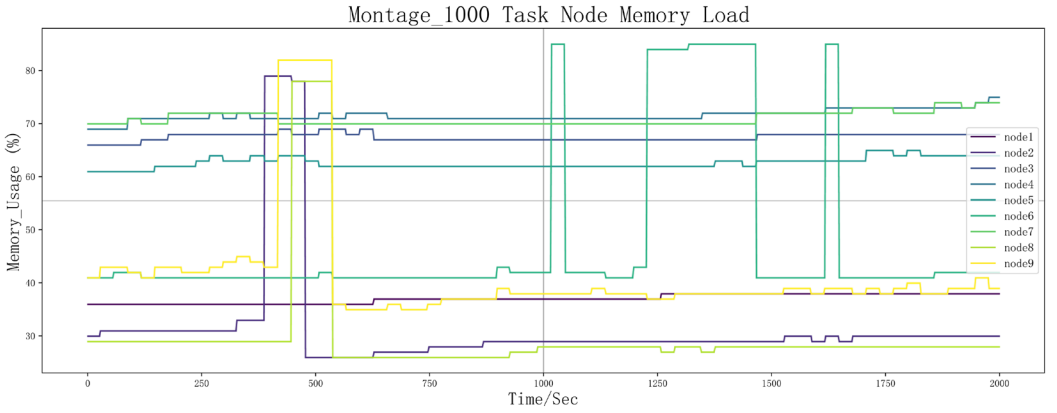


Figure 9. Montage_1000 Task Node Memory Load



contrast, the HEFT and SDPLS algorithms select the processor that can complete tasks the earliest on the basis of device and task information, without considering that previous task allocations to that node might lead to resource occupation and consequently increase the waiting time for current tasks, thereby extending task execution time. Moreover, by adjusting the reward function and retraining the model, it can be adapted to different environmental needs. In scenarios involving real-time data processing or high-performance computing, it is advisable to increase the weight of task execution duration. Conversely, in high-concurrency processing environments, increasing the device load weight can help prevent certain nodes from becoming bottlenecks as a result of excessive load. Here, considering the generic environment, we set the task execution duration and device load balancing as parameters with equal weight.

In the load balancing test of the GAT-A2C algorithm, the CPU and memory loads of the nine nodes were documented. **Figure 8** and **Figure 9**, as well as **Figure 10** and **Figure 11**, depict the load curves for CPU and memory of the nine nodes during the execution of the Montage_1000 and CyberShake_100 tasks, respectively. The experimental results demonstrate that none of the nodes in the cluster experienced overload during the task execution time.

Figure 10. Cybershake_100 Task Node CPU Load

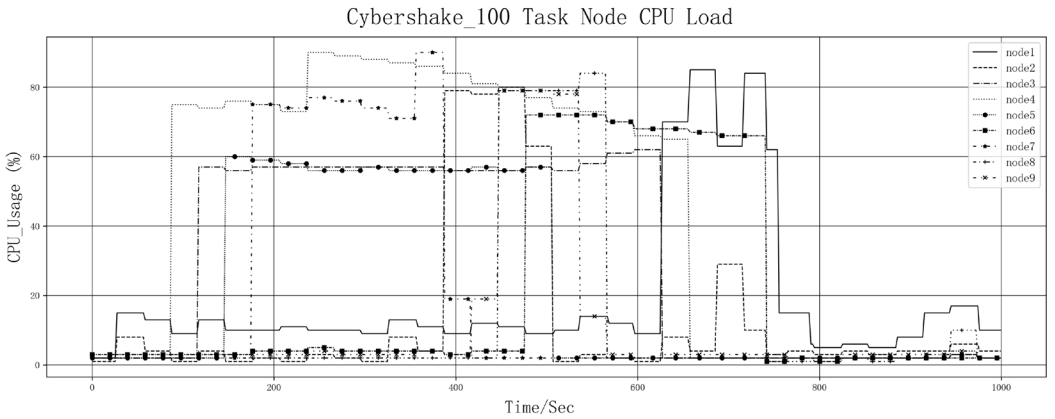
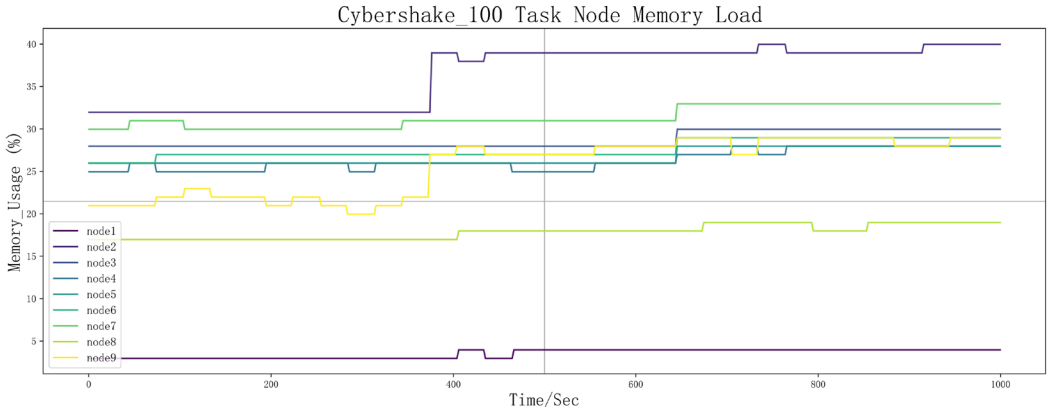


Figure 11. Cybershake_100 Task Node Memory Load



Simulator Environment and Parameter Configuration

We conducted simulations of dynamic network connections in a cluster using the open-source scheduling framework COSCO-Workflow (Tuli et al., 2021) on a 64-bit Windows 10 operating system. The node configurations mirrored those within the cluster. Specifically, we represented five terminal devices with settings of 2vCPU and 4GB of memory, while three edge devices were configured with 4vCPU and 8GB of memory, and one cloud service center node was configured with 8vCPU and 32GB of memory. The bandwidth between terminal devices and the transmission bandwidth from terminal devices to edge or cloud center nodes were set within the range of [0,30Mbps]. The bandwidth between edge devices and the transmission bandwidth from edge devices to terminal devices or the cloud center were set within the range of [50,100Mbps], while the bandwidth from the cloud center node to edge devices or terminal devices was set within the range of [500Mbps, 1Gbps]. These bandwidth settings were controlled by the function Time_varying_bandwidth implemented through scripts, which executed once every 60 intervals, randomly sampling values within the specified ranges. The hyperparameter settings of the algorithm and the selection of comparative algorithms remained consistent with those employed in the cluster environment testing.

Table 3. Evaluation of Makespan Using Montage Workflow

Tasks	Greedy(sec)	HEFT(sec)	SDPLS(sec)	GAT-A2C(sec)
25	222.07	**99.15**	126.34	117.96
50	288.49	268.12	245.76	**191.63**
100	624.92	495.25	411.54	**382.43**
1000	4877.76	3685.01	3579.62	**3203.24**

Table 4. Evaluation of Makespan Using Cybershake Workflow

Tasks	Greedy(sec)	HEFT(sec)	SDPLS(sec)	GAT-A2C(sec)
30	388	212.16	264.23	**191.30**
50	749.31	524.26	539.12	**468.63**
100	1684.28	1589.65	1204.35	**1049.29**

On the basis of the complexity analysis of the algorithm, the complexity of the graph encoder segment is denoted as $O(k \cdot E \cdot d_{in} \cdot d_{out})$. The complexity of the first step, the graph convolution operation (DAGConv), depends on the number of edges E (i.e., the number of dependencies between tasks or connections between resource nodes) and the dimensions of the input and output features. Taking resource nodes as an example, the input dimension d_{in} is 4, which includes the node's current computational capacity $u_{m.cpu}$, total computational capacity $u_{m.cpu.total}$, current memory resources $u_{m.mem}$, and total memory resources $u_{m.mem.total}$. The output dimension d_{out} is set to 128. The feature dimension of the edges is 1, representing the network bandwidth between two computing nodes, with a bandwidth of 0 if the connection is interrupted. In the second step, the process iterates k times, where k is set to 2. The third step involves aggregating all node embeddings into a single graph embedding. In the graph decoder segment, the complexity of the algorithm arises from the dot product operations used to compute attention scores, denoted as $O(N^2 \cdot d_{key})$, where N is the number of nodes and d_{key} is the dimension of the keys, which is set to 128. As the scale of tasks and network increases, with a corresponding increase in the number of resource nodes and edges, the complexity in the graph encoder part exhibits linear growth, while in the graph decoder part, the complexity shows quadratic growth.

The GAT-A2C algorithm demonstrates significant temporal advantages over other algorithms. For the Montage_1000 workflow, GAT-A2C is on average 38.40% faster than the greedy algorithm, 11.35% faster than the HEFT algorithm, and 11.56% faster than the SDPLS algorithm. For the Cybershake_100 workflow, GAT-A2C averages 41.95% faster than greedy, 18.14% faster than HEFT, and 17.85% faster than SDPLS. These results indicate that as task sizes increase, the complexity growth of the GAT-A2C algorithm has a minimal negative impact on task completion times. Compared to other commonly used algorithms, it significantly reduces the time required to complete tasks, especially when there are many tasks, in which case its efficiency advantage becomes even more pronounced.

CONCLUSION

This paper proposes a generic A2C task scheduling algorithm utilizing a graph attention mechanism to achieve efficient task scheduling and device load balancing in a space-air-ground integrated edge cloud environment. We employed GCNs to extract information from task and resource graphs, with a focus on improving the matching mechanism between tasks and computing resources in dynamic network environments. We also conducted tests on typical workflow processing applications for task completion time and device load balancing metrics. The test results show that compared to three benchmark algorithms, our method achieves an improvement of over 10% in task completion

time, while also performing well in terms of device load balancing, demonstrating the algorithm's robustness in environments with time-variant bandwidth. This method is suitable for task scheduling scenarios in highly dynamic network environments and offers valuable insights for addressing task scheduling issues in remote terrestrial areas or specific regions with weak connectivity.

LIMITATIONS AND FUTURE PROSPECTS

This article still has inadequacies in representing the computational resource model. IoT applications requiring artificial intelligence need substantial computational power, largely for tasks involving model training or inference, which are typically executed on computing nodes equipped with GPUs. In future work, we plan to incorporate GPU computational resources into the computing node model to enhance the practicality of our algorithm. Additionally, we need to focus on the relationship between the algorithm's complexity and the scale of tasks and network. Testing the convergence speed and response time of the algorithm will also be part of our future work. Moreover, the integration of edge computing environments with streaming computation frameworks is becoming a trend for real-time data processing in IoT (Babar et al., 2023). We intend to implement the integration of our algorithm with the streaming data processing framework Apache Flink, targeting IoT artificial intelligence applications such as object detection in actual environments. Through Flink functions, we will implement the parallelization of task workflows in a DAG structure and package the applications into Docker images to be deployed on edge nodes. Subsequently, our task scheduling algorithm will be applied to optimize the completion time of tasks and the load on device nodes within the edge cloud environment, thereby enhancing the speed of streaming data processing and optimizing resource utilization efficiency in edge cloud settings.

AUTHOR NOTE

The data used to support the findings of this study are included within the article.

CONFLICTS OF INTEREST

The authors declare no conflicts of interest.

FUNDING INFORMATION

No funding was received for this work.

This research received no specific grant from any funding agency in the public, commercial, or not-for-profit sectors.

PROCESS DATES

Received: March 25, 2024, Revision: April 28, 2024, Accepted: April 28, 2024

CORRESPONDING AUTHOR

Correspondence concerning this article should be addressed to Corresponding Yunke Jiang and Xiaojuan Sun, Aerospace Information Research Institute, Chinese Academy of Sciences, Beijing, China; Key Laboratory of Technology in Geo-Spatial Information Processing and Application System, Beijing, China; School of Electronic, Electrical and Communication Engineering, University of Chinese Academy of Sciences, Beijing, China.

Email:, xjsun@mail.ie.ac.cn.

REFERENCES

Ai, L., Tan, B., Zhang, J., Wang, R., & Wu, J. (2023). Dynamic offloading strategy for delay-sensitive task in mobile-edge computing networks. *IEEE Internet of Things Journal*, *10*(1), 526–538. 10.1109/JIOT.2022.3202797

Ajmal, M. S., Iqbal, Z., Khan, F. Z., Ahmad, M., Ahmad, I., & Gupta, B. B. (2021). Hybrid ant genetic algorithm for efficient task scheduling in cloud data centers. *Computers & Electrical Engineering*, *95*, 107419. 10.1016/j.compeleceng.2021.107419

Alakbarov, R. (2022). An optimization model for task scheduling in mobile cloud computing. *International Journal of Cloud Applications and Computing*, *12*(1), 1–17. 10.4018/IJCAC.297102

Babar, M., Jan, M. A., He, X., Tariq, M. U., Mastorakis, S., & Alturki, R. (2023). An optimized IoT-enabled big data analytics architecture for edge–cloud computing. *IEEE Internet of Things Journal*, *10*(5), 3995–4005. 10.1109/JIOT.2022.315755238046398

Baburao, D., Pavankumar, T., & Prabhu, C. S. R. (2021). Load balancing in the fog nodes using particle swarm optimization-based enhanced dynamic resource allocation method. *Applied Nanoscience*, *13*(3), 1–10.

Bisht, J., & Vampugani, V. S. (2021). Load and cost-aware min-min workflow scheduling algorithm for heterogeneous resources in fog, cloud, and edge scenarios. *International Journal of Cloud Applications and Computing*, *12*(1), 1–20. 10.4018/IJCAC.2022010105

Cao, B., Zhang, J., Liu, X., Sun, Z., Cao, W., Nowak, R., & Lv, Z. (2022). Edge–cloud resource scheduling in space–air–ground-integrated networks for internet of vehicles. *IEEE Internet of Things Journal*, *9*(8), 5765–5772. 10.1109/JIOT.2021.3065583

Chai, F., Zhang, Q., Yao, H., Xin, X., Gao, R., & Guizani, M. (2023). Joint multi-task offloading and resource allocation for mobile edge computing systems in satellite IoT. *IEEE Transactions on Vehicular Technology*, *72*(6), 7783–7795. 10.1109/TVT.2023.3238771

Chen, X., Lin, C., & Lin, B. (2024). An intelligent workflow scheduling scheme for complex network robustness in fuzzy edge-cloud environments. *IEEE Transactions on Network Science and Engineering*, *11*(1), 1–18. 10.1109/TNSE.2023.3321089

Chen, X., Wu, Y., & Xiao, S. (2023). Particle swarm–grey wolf cooperation algorithm based on microservice container scheduling problem. *IEEE Access : Practical Innovations, Open Solutions*, *11*, 16667–16682. 10.1109/ACCESS.2023.3244881

Chen, Z., Hu, J., & Min, G. (2019). Learning-Based resource allocation in cloud data center using advantage actor-critic. In *ICC 2019—2019 IEEE International Conference on Communications (ICC)* (pp. 1–6). IEEE. 10.1109/ICC.2019.8761309

Cui, H., Zhang, J., Geng, Y., Xiao, Z., Sun, T., Zhang, N., Liu, J., Wu, Q., & Cao, X. (2022). Space-air-ground integrated network (SAGIN) for 6G: Requirements, architecture and challenges. *China Communications*, *19*(2), 90–108. 10.23919/JCC.2022.02.008

Deelman, E., Vahi, K., Juve, G., Rynge, M., Callaghan, S., Maechling, P. J., Mayani, R., Chen, W., Ferreira Da Silva, R., Livny, M., & Wenger, K. (2015). Pegasus, a workflow management system for science automation. *Future Generation Computer Systems*, *46*, 17–35. 10.1016/j.future.2014.10.008

Elgendy, I. A., Zhang, W.-Z., He, H., Gupta, B. B., & Abd El-Latif, A. A. (2021). Joint computation offloading and task caching for multi-user and multi-task MEC systems: Reinforcement learning-based algorithms. *Wireless Networks*, *27*(1), 2023–2038. 10.1007/s11276-021-02554-w

Farid, M., Lim, H. S., Lee, C. P., & Latip, R. (2023). Scheduling scientific workflow in multi-cloud: A multi-objective minimum weight optimization decision-making approach. *Symmetry*, *15*(11), 2047. 10.3390/sym15112047

Guo, H., & Liu, J. (2018). Collaborative computation offloading for multiaccess edge computing over fiber–wireless networks. *IEEE Transactions on Vehicular Technology*, *67*(5), 4514–4526. 10.1109/TVT.2018.2790421

Hajvali, M., Adabi, S., Rezaee, A., & Hosseinzadeh, M. (2023). Decentralized and scalable hybrid scheduling-clustering method for real-time applications in volatile and dynamic fog-cloud environments. *Journal of Cloud Computing (Heidelberg, Germany)*, *12*(1), 66. 10.1186/s13677-023-00428-4

Hamidi, H., & Mohammadi, K. (2006). Modeling fault tolerant and secure mobile agent execution in distributed systems. *International Journal of Intelligent Information Technologies*, *2*(1), 21–36. 10.4018/jiit.2006010102

Han, J., Wang, H., Wu, S., Wei, J., & Yan, L. (2020). Task scheduling of high dynamic edge cluster in satellite edge computing. In *2020 IEEE World Congress on Services (SERVICES)* (pp. 287–293). IEEE. 10.1109/SERVICES48979.2020.00063

Huang, X., Jiang, Y., Fan, H., Tang, H., Wang, Y., Jin, J., Wan, H., & Zhao, X. (2021). TATA: Throughput-aware TAsk placement in heterogeneous stream processing with deep reinforcement learning. In *2021 IEEE International Conference on Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking (ISPA/BDCloud/SocialCom/SustainCom)* (pp. 44–54). IEEE. 10.1109/ISPA-BDCloud-SocialCom-SustainCom52081.2021.00021

Kar, B., Yahya, W., Lin, Y.-D., & Ali, A. (2023). Offloading using traditional optimization and machine learning in federated cloud–edge–fog systems: A survey. *IEEE Communications Surveys and Tutorials*, *25*(2), 1199–1226. 10.1109/COMST.2023.3239579

Kipf, T. N., & Welling, M. (2017). Semi-supervised classification with graph convolutional networks. ArXiv. http://arxiv.org/abs/1609.02907

Lee, M., Yu, S., Kwon, K., Lee, M., Lee, J., & Kim, H. (2024). Mixed-integer linear programming model for scheduling missions and communications of multiple satellites. *Aerospace (Basel, Switzerland)*, *11*(1), 83. 10.3390/aerospace11010083

Li, P., Xiao, Z., Wang, X., Huang, K., Huang, Y., & Gao, H. (2024). EPtask: Deep reinforcement learning based energy-efficient and priority-aware task scheduling for dynamic vehicular edge computing. *IEEE Transactions on Intelligent Vehicles*, *9*(1), 1830–1846. 10.1109/TIV.2023.3321679

Li, Y., Guo, X., Meng, Z., Qin, J., Li, X., Ma, X., Ren, S., & Yang, J. (2023). A hierarchical resource scheduling method for satellite control system based on deep reinforcement learning. *Electronics (Basel)*, *12*(19), 3991. 10.3390/electronics12193991

Li, Z., & Chen, P. (2023). Risk-aware distributionally robust optimization for mobile edge computation task offloading in the space–air–ground integrated network. *Sensors (Basel)*, *23*(12), 5729. 10.3390/s2312572937420894

Liu, J., Mao, Y., Zhang, J., & Letaief, K. B. (2016). Delay-optimal computation task scheduling for mobile-edge computing systems. In *2016 IEEE International Symposium on Information Theory (ISIT)* (pp. 1451–1455). IEEE. 10.1109/ISIT.2016.7541539

Liu, Z., Huang, L., Gao, Z., Luo, M., Hosseinalipour, S., & Dai, H. (2023). GA-DRL: Graph neural network-augmented deep reinforcement learning for DAG task scheduling over dynamic vehicular clouds. ArXiv. http://arxiv.org/abs/2307.00777

Mao, S., He, S., & Wu, J. (2021). Joint UAV position optimization and resource scheduling in space-air-ground integrated networks with mixed cloud-edge computing. *IEEE Systems Journal*, *15*(3), 3992–4002. 10.1109/JSYST.2020.3041706

Mao, X., Cao, Z., Fan, M., Wu, G., & Pedrycz, W. (2023). DL-DRL: A double-level deep reinforcement learning approach for large-scale task scheduling of multi-UAV. arXiv. https://arxiv.org/abs/2208.02447

Nilchi, A. R. N., Vafaei, A., & Hamidi, H. (2008). Evaluation of security and fault tolerance in mobile agents. In *2008 5th IFIP International Conference on Wireless and Optical Communications Networks (WOCN '08)* (pp. 1–5). IEEE. 10.1109/WOCN.2008.4542509

Niu, L., Chen, X., Zhang, N., Zhu, Y., Yin, R., Wu, C., & Cao, Y. (2023). Multiagent meta-reinforcement learning for optimized task scheduling in heterogeneous edge computing systems. *IEEE Internet of Things Journal*, *10*(12), 10519–10531. 10.1109/JIOT.2023.3241222

Sheng, S., Chen, P., Chen, Z., Wu, L., & Yao, Y. (2021). Deep reinforcement learning-based task scheduling in IoT edge computing. *Sensors (Basel)*, *21*(5), 1666. 10.3390/s2105166633671072

J., Du, J., Wang, J., Wang, J., & Yuan, J. (2020). Priority-aware task offloading in vehicular fog computing based on deep reinforcement learning. *IEEE Transactions on Vehicular Technology*, *69*(12), 16067–16081. 10.1109/TVT.2020.3041929

Shi, W., Cao, J., Zhang, Q., Li, Y., & Xu, L. (2016). Edge computing: Vision and challenges. *IEEE Internet of Things Journal*, *3*(5), 637–646. 10.1109/JIOT.2016.2579198

Soltani, N., Soleimani Neysiani, B., & Barekatain, B. (2017). Heuristic algorithms for task scheduling in cloud computing: A survey. *International Journal of Computer Network and Information Security*, *9*(8), 16–22. 10.5815/ijcnis.2017.08.03

Srikanth, G. U., & Geetha, R. (2023). Effectiveness review of the machine learning algorithms for scheduling in cloud environment. *Archives of Computational Methods in Engineering*, *30*(6), 3769–3789. 10.1007/s11831-023-09921-0

Sun, Z., Mo, Y., & Yu, C. (2023). Graph-reinforcement-learning-based task offloading for multiaccess edge computing. *IEEE Internet of Things Journal*, *10*(4), 3138–3150. 10.1109/JIOT.2021.3123822

Liu, J., Shi, Y., Fadlullah, Z. Md., & Kato, N. 2018. Space-air-ground integrated network:Survey, A. (●●●).. . *IEEE Communications Surveys and Tutorials*, *20*(4), 2714–2741.

Tang, Q., Fei, Z., Li, B., & Han, Z. (2021). Computation offloading in LEO satellite networks with hybrid cloud and edge computing. *IEEE Internet of Things Journal*, *8*(11), 9164–9176. 10.1109/JIOT.2021.3056569

Topcuoglu, H., Hariri, S., & Min-You, W. (2002). Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE Transactions on Parallel and Distributed Systems*, *13*(3), 260–274. 10.1109/71.993206

Tuli, S., Casale, G., & Jennings, N. R. (2021). MCDS: AI augmented workflow scheduling in mobile edge cloud computing systems. arXiv. https://arxiv.org/abs/2112.07269

Wang, C., Li, Y., & Jin, D. (2014). Mobility-assisted opportunistic computation offloading. *IEEE Communications Letters*, *18*(10), 1779–1782. 10.1109/LCOMM.2014.2347272

Wang, C., Peng, J., Cai, L., Peng, H., Liu, W., Gu, X., & Huang, Z. (2023). AI-enabled spatial-temporal mobility awareness service migration for connected vehicles. *IEEE Transactions on Mobile Computing*, *23*(4), 3274–3290. 10.1109/TMC.2023.3271655

Wang, F., Jiang, D., Qi, S., Qiao, C., & Shi, L. (2021). A dynamic resource scheduling scheme in edge computing satellite networks. *Mobile Networks and Applications*, *26*(2), 597–608. 10.1007/s11036-019-01421-5

Wang, Y., Yang, J., Guo, X., & Qu, Z. (2019). Satellite Edge computing for the internet of things in aerospace. *Sensors (Basel)*, *19*(20), 4375. 10.3390/s1920437531658684

Wang, Z., Goudarzi, M., Gong, M., & Buyya, R. (2024). Deep reinforcement learning-based scheduling for optimizing system load and response time in edge and fog computing environments. *Future Generation Computer Systems*, *152*, 55–69. 10.1016/j.future.2023.10.012

Xiu, X., Li, J., Long, Y., & Wu, W. (2023). MRLCC: An adaptive cloud task scheduling method based on meta reinforcement learning. *Journal of Cloud Computing (Heidelberg, Germany)*, *12*(1), 75. 10.1186/s13677-023-00440-8

Yoo, S., Jeong, S., Kim, J., & Kang, J. (2023). Cache-assisted mobile edge computing over space-air-ground integrated networks for extended reality applications. arXiv. https://arxiv.org/abs/2309.03357

Yu, S., Dab, B., Movahedi, Z., Langar, R., & Wang, L. (2020). A socially-aware hybrid computation offloading framework for multi-access edge computing. *IEEE Transactions on Mobile Computing*, *19*(6), 1247–1259. 10.1109/TMC.2019.2908154

Yu, S., Gong, X., Shi, Q., Wang, X., & Chen, X. (2021). EC-SAGINs: Edge computing-enhanced space-air-ground integrated networks for internet of vehicles. *IEEE Internet of Things Journal*, *9*(8), 5742–5754. 10.1109/JIOT.2021.3052542

Zhang, K., Peng, M., & Sun, Y. (2021). Delay-optimized resource allocation in fog-based vehicular networks. *IEEE Internet of Things Journal*, *8*(3), 1347–1357. 10.1109/JIOT.2020.3010861

Shi,

Zhang, X., Tian, S., Liu, Y., & Cao, Z. (2022). User location-aware edge services selection based on generative adversarial network and improved ant colony algorithm. *Applied Intelligence*, *53*(11), 13643–13664. 10.1007/s10489-022-04093-z

Zhang, Z., Zhang, W., & Tseng, F.-H. (2019). Satellite mobile edge computing: Improving QoS of high-speed satellite-terrestrial networks using edge computing techniques. *IEEE Network*, *33*(1), 70–76. 10.1109/MNET.2018.1800172

Zhao, M., Chen, C., Liu, L., Lan, D., & Wan, S. (2022). Orbital collaborative learning in 6G space-air-ground integrated networks. *Neurocomputing*, *497*, 94–109. 10.1016/j.neucom.2022.04.098

Zheng, W., Wang, C., Chen, Z., & Zhang, D. (2022). A priority-based level heuristic approach for scheduling DAG applications with uncertainties. *2022 IEEE 25th International Conference on Computer Supported Cooperative Work in Design (CSCWD)* (pp. 1022–1027). IEEE.

Zhu, X., & Jiang, C. (2023). Delay optimization for cooperative multi-tier computing in integrated satellite-terrestrial networks. *IEEE Journal on Selected Areas in Communications*, *41*(2), 366–380. 10.1109/JSAC.2022.3227083