

## Article

# A Collaborative Inference Algorithm in Low-Earth-Orbit Satellite Network for Unmanned Aerial Vehicle

Zhengqian Xu <sup>1</sup>, Peiying Zhang <sup>2,3,4,5</sup> , Chengcheng Li <sup>1,\*</sup>, Hailong Zhu <sup>6</sup>, Guanjun Xu <sup>7</sup>  and Chenhua Sun <sup>1</sup>

- <sup>1</sup> The 54th Research Institute of CETC, Shijiazhuang 050081, China; zqxu\_0@stu.xidian.edu.cn (Z.X.); zhuansuny@bupt.edu.cn (C.S.)
- <sup>2</sup> Qingdao Institute of Software, College of Computer Science and Technology, China University of Petroleum (East China), Qingdao 266580, China; zhangpeiy@upc.edu.cn
- <sup>3</sup> Shandong Provincial Key Laboratory of Computer Networks, Shandong Computer Science Center (National Supercomputer Center in Jinan), Qilu University of Technology (Shandong Academy of Sciences), Jinan 250013, China
- <sup>4</sup> State Key Laboratory of Integrated Services Networks, Xidian University, Xi'an 710071, China
- <sup>5</sup> National Research Center for Information Science and Technology, Tsinghua University, Beijing 100084, China
- <sup>6</sup> State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing 100876, China; zhuh@bupt.edu.cn
- <sup>7</sup> Space Information Research Institute, Hangzhou Dianzi University, Hangzhou 310018, China; gjxu@ee.ecnu.edu.cn
- \* Correspondence: lengcangche@bupt.cn

**Abstract:** In recent years, the low-Earth-orbit (LEO) satellite network has achieved considerable development. Moreover, it is necessary to introduce edge computing into LEO networks, which can provide high-quality services, such as worldwide seamless low-delay computation offloading for unmanned aerial vehicles (UAVs) or user terminals and nearby remote-sensing data processing for UAVs or satellites. However, because the computation resource of the satellite is relatively scarce compared to the ground server, it is hard for a single satellite to complete massive deep neural network (DNN) inference tasks in a short time. Consequently, in this paper, we focus on the multi-satellite collaborative inference problem and propose a novel COllaborative INference algorithm for LEO edge computing called COIN-LEO. COIN-LEO manages to split the complete DNN model into several submodels consisting of some consecutive layers and deploy these submodels to several satellites for inference. We innovatively leverage deep reinforcement learning (DRL) to efficiently split the model and use a neural network (NN) to predict the time required for inference tasks of a specific submodel on a specific satellite. By implementing COIN-LEO and evaluating its performance in a highly realistic satellite-network-emulation platform, we find that our COIN-LEO outperforms baseline algorithms in terms of inference throughput, time consumed and network traffic overhead.



**Citation:** Xu, Z.; Zhang, P.; Li, C.; Zhu, H.; Xu, G.; Sun, C. A Collaborative Inference Algorithm in Low-Earth-Orbit Satellite Network for Unmanned Aerial Vehicle. *Drones* **2023**, *7*, 575. <https://doi.org/10.3390/drones7090575>

Academic Editor: Emmanouel T. Michailidis

Received: 11 August 2023

Revised: 6 September 2023

Accepted: 9 September 2023

Published: 11 September 2023



**Copyright:** © 2023 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

## 1. Introduction

A satellite network is a type of communication network that can provide beyond-line-of-sight communication services [1]. It consists of a space segment, a ground segment and a user segment. The space segment is mainly composed of a satellite (s) that has communication functions, such as signal processing and routing. The ground segment is mainly composed of gateway stations that can interconnect with terrestrial networks, such as the 4G/5G mobile network. The user segment mainly contains satellite terminals used by end users. It is widely accepted that the 6G network is a space-ground integrated network. As such, the satellite network is an important part of 6G [2].

Compared with traditional geosynchronous (GEO) satellite networks, the low-Earth-orbit (LEO) network has some intrinsic advantages, such as relatively low propagation latency between the terminal and the LEO satellite, seamless global coverage and so

on [3]. Consequently, in recent years, the LEO satellite network has achieved considerable development. Many companies, such as SpaceX, OneWeb, Amazon and Telesat, have deployed their own LEO satellite networks [4]. These networks are expected to meet the growing demand for reliable and accessible Internet connection worldwide.

Despite the fact that the LEO network has many advantages and has attracted much attention in recent years, it cannot provide high-quality services in some scenarios:

- Computation offloading: If a UAV or a user terminal wants to offload computation-intensive tasks to the network, it has to offload the task to the cloud on the ground. However, the end-to-end delay between the user and the cloud is usually very large because the distances between user links (between satellite terminal and the access satellite) and inter-satellite links (ISLs) are very long, which means that the signal propagation delay is very large;
- Remote-sensing data processing: Traditionally, after the remote-sensing payload on the UAV or after the satellite obtains the original data, the UAV or the satellite has to transmit these original/preprocessed data to the cloud on the ground where the data are processed and useful information is obtained [5]. However, these original/preprocessed data are usually very large. For example, the size of a single image taken by a Gaofen-1 satellite is up to several gigabits, and the size of satellite-based synthetic-aperture radar (SAR) echo data for a single image is up to 50 Gbits [6]. As remote-sensing technology evolves, this size becomes larger and larger. This type of remote-sensing data transmission presses hard upon the ISLs and feeder links.

Therefore, we believe that it is necessary to introduce edge computing into LEO networks, which conforms to the trends in the integration of computation and the networking of the 6G system [7]. By deploying computation and storage resource on the LEO satellite, it is possible to build a collaborative space-based edge-computing system that is composed of many satellite-based computing platforms.

Introducing edge computing into the LEO network can yield obvious benefits:

- Low-delay computation offloading: By offloading computation tasks to the LEO satellite (instead of the cloud on the ground), which is the edge of the LEO network, it is possible to support users' latency-sensitive, computationally intensive applications;
- Data analysis nearby the source: By processing the remote-sensing data nearby the data sources (the UAV-based or satellite-based remote-sensing payload), the bandwidth occupation for transmitting huge amounts of original/preprocessed data is saved. Moreover, by processing/analyzing data using an artificial intelligence (AI) algorithm via the satellite-based computing platform and sending only the useful information that has a much smaller data size to the user, the user can obtain the desired information much faster;
- More rapid remote-sensing information dissemination: Edge computing in the LEO network can lead to improved capabilities in near-real-time remote sensing, in which data are processed and transmitted relatively quickly. Some remote-sensing satellites are designed for specific applications, such as disaster monitoring, in which more rapid information dissemination is critical. By leveraging edge computing, it is possible to reduce the time needed for useful information dissemination obtained by satellite remote sensing.

The inference process of deep learning (DL) is an important type of algorithm that is run on the LEO satellite-based edge-computing platform. However, the computing power of an LEO satellite is less than that of an edge-computing node in the terrestrial network, so it is necessary to use multiple satellites' computing resource in order to complete massive inference tasks in a short time [8,9]. For example, when there are many users who are offloading many inference tasks to the access satellite or when a remote-sensing satellite has many images (or video frames) requiring processing, it is necessary for several nearby satellites to compute collaboratively to avoid overloading the access satellite [10].

Therefore, in this paper, we focus on the collaborative inference problem in the LEO edge-computing system and propose a COllaborative INference algorithm in LEO edge computing called COIN-LEO, in which several satellites can collaboratively complete many inference tasks in a finite time. Specifically, the COIN-LEO algorithm first appropriately selects several satellites to participate in the inference process in the ‘neighborhood’ of the data source satellite. Then, COIN-LEO splits the deep neural network (DNN) model into several parts, in which a part consists of several consecutive layers of the model. After obtaining the model-splitting result, COIN-LEO assigns these model parts to selected satellites, and each satellite accomplishes the inference task that corresponds to the assigned model part. In this way, several satellites collaboratively accomplish the compete inference process in a pipeline fashion, which can improve the amount of accomplished inference tasks within a given time.

Our main contributions are summarized as follows:

- We propose a simple method to select appropriate satellites to participate in the collaborative inference process, which is easy to implement in engineering. The core idea is to take into the entire topology of the LEO network in consideration and select several satellites to build a circle-like topology;
- We propose an efficient deep reinforcement learning (DRL)-based algorithm for DNN model splitting. To support the model-splitting algorithm, we use a neural network (NN) to predict the inference time of a specific submodel deployed on a specific satellite and conduct many experiments to obtain enough training data;
- We conduct the collaborative inference experiments in a testbed that highly realistically simulates the LEO network and evaluates several performance metrics, such as inference throughput and time.

The following article is organized as follows. Section 2 provides a comprehensive overview of related works in the literature. The system model is described in Section 3. In Section 4, we explain the details of the COIN-LEO algorithm. Performance evaluation is presented in Section 5. Section 6 concludes this article.

## 2. Related Work

### 2.1. Collaborative Inference

In order to improve the inference performance, there are some works that focus on collaborative inference in the literature.

Shao et al. study edge inference in resource-constrained devices and focus on device edge co-inference, in which the inference is assisted by an edge-computing server [11]. They propose a general three-step framework for the inference: model split-point selection to determine the on-device model; communication-aware model compression to reduce the on-device computation and the resulting communication overhead; and task-oriented encoding of the intermediate feature to further reduce the communication overhead. They only consider splitting the model into two parts. However, in order to utilize many satellites' resources, we limit ourselves to only using one satellite's computing resource.

Lin et al. explore the distributed DNN deployment problem on the next-generation network, which includes multi-level hierarchical computing units from the UE to the cloud by leveraging FC and MEC [12]. The computing units in the lower level (i.e., nearer to the UE) usually have weaker computing and bandwidth capacity (e.g., 10 Gflops and 1 Gbps), whereas the ones in the higher level (i.e., nearer to the cloud) are more powerful (e.g., 1 Tflops and 10 Gbps). They solve the deployment problem considering both response time and inference throughput. They study the distributed DNN deployment problem considering multi-level hierarchical computing units, which is different from this paper.

Khan et al. introduce methods for distributed inference over IoT devices and edge servers [13]. They propose two distinct algorithms to split the deep neural network layers' computation between the IoT device and an edge server: the early split strategy (ESS) for battery-powered IoT devices and the late split strategy (LSS) for IoT devices connected to a regular power source, considering several factors, such as the available bandwidth of

ISLs, the available computing resource of satellites, the computing power needed for the inference process as part of the DNN model, etc. They study the distributed inference by the IoT device and the edge server, which is different to our problem.

Kim et al. present experimental results of DNN inference offloading in a real-world 5G MEC testbed to meet requirements for high detection accuracy and low end-to-end latency for object detection [14]. They implement the DNN offloading by applying task pipeline parallelism and a DNN task-decoupling scheme from the edge to the MEC server. However, they offload the whole inference task to the MEC server, which is different from our approach, which splits the inference task into multiple parts.

Yu et al. study the cloud-based model serving problem, in which users deploy models as serverless functions and let the platform handle provisioning and scaling [15]. They observe that serverless functions have constrained resources in CPU and memory, making them inefficient or infeasible to serve the whole large neural networks. As such, they present Gillis, a serverless-based model serving system that automatically partitions a large model across multiple serverless functions for faster inference and reduced memory footprint per function. In their scenario, the DNN model partitions are deployed on different serverless functions between which are low-delay links with large bandwidth, which is different from the satellites' collaborative inference scenario in which ISLs have high delay.

## 2.2. Satellite-Based Edge Computing

In recent years, more and more researchers have begun to recognize the importance of satellite-based edge computing.

Wang et al. introduce a novel satellite–terrestrial network with double edge computing to reap the benefits of providing computing services to remote areas [16]. A strategy is designed to solve the problem of efficiently scheduling the edge servers distributed in the satellite–terrestrial networks to provide more powerful edge-computing services. For the purpose of allocating satellite edge-computing resources efficiently in the strategy, a double-edge-computation offloading algorithm is proposed to optimize energy consumption and reduce latency by assigning tasks to edge servers with minimal cost.

Wang et al. propose a game-theoretic approach to the optimization of computation offloading strategy in satellite edge computing [17]. The system model for computation offloading in satellite edge computing is established, considering the intermittent terrestrial–satellite communication caused by satellites orbiting. They conduct a computation offloading game framework and compute the response time and energy consumption of a task based on the queuing theory as metrics of optimizing performance.

Zhang et al. believe that orbital edge-computing (OEC) technology that deploys edge-computing servers on LEO satellite constellations can meet the growing demand for the real time reliability of various applications [18]. Based on the above motivations, they propose an OEC task allocation (OECTA) algorithm based on the greedy strategy in LEO satellite networks for the Walker Delta satellite constellation, which fully utilizes satellite computing resources to provide services to ground users. They also analyze the performance of their proposed algorithm in terms of computational cost.

Cheng et al. observe that, different from terrestrial edge computing, the computing capacity in LEO satellites is usually unstable due to the limited and consistently changing energy supply of fast-orbiting LEO satellites [19]. They propose a dynamic offloading strategy to minimize the overall delay of tasks from terrestrial users in a satellite edge-computing system, subject to the energy and computing capacity constraints of the LEO satellite. Based on Lyapunov optimization theory, they convert a long-term stochastic problem with a time-varying energy constraint into multiple deterministic one-slot problems parameterized by the current system state, in which task-offloading decisions, computing resource allocation and transmit power control are jointly optimized.

Israel Leyva-Mayorga and his colleagues investigated a framework for optimizing image distribution and parameter compression in LEO edge computing [20]. Their approach increases the number of supported images by the system while reducing energy

consumption. Their research primarily focuses on energy efficiency. Zhu and his team studied task-offloading algorithms based on deep reinforcement learning, achieving better offloading cost effectiveness [21]. Their approach is suitable for satellite–terrestrial networks with fast fading channels, and their computational tasks lean more towards task offloading rather than multi-satellite collaborative inference.

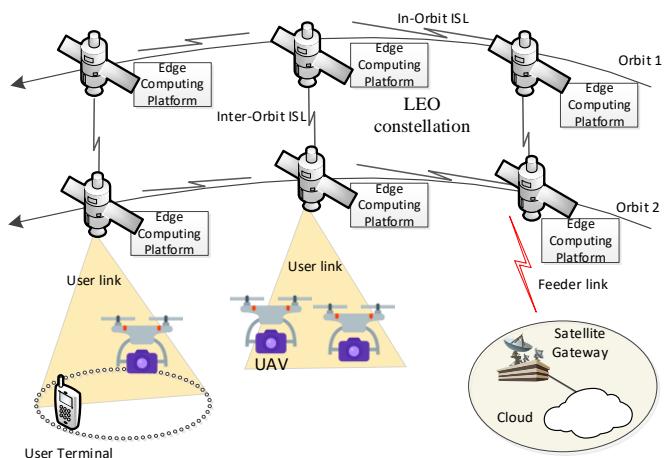
As we can see from these related works, present research in the field of LEO edge computing predominantly focuses on task offloading, emphasizing the transfer of computational tasks from users to satellites to enhance computational efficiency in the LEO satellite network. These works encompass a wide range of topics, including deep reinforcement learning, distributed task scheduling and model optimization, with the aim of addressing resource constraints and energy limitations. However, research on collaborative inference among satellites remains relatively scarce, leaving a critical gap in the current body of knowledge. Research into multi-satellite collaborative inference investigates how to harness the resources of multiple LEO satellites to execute deep-learning inference tasks in a distributed manner across multiple satellites, thereby improving inference performance. Our research is dedicated to filling this research gap.

### 3. System Model

#### 3.1. LEO Edge Computing

The LEO satellite network consists of a constellation of LEO satellites positioned in low Earth orbits. These LEO satellites act as access points and routers. These satellites work together to ensure continuous coverage and connectivity to all the regions of the world. The low orbital altitude of LEO satellites results in reduced signal latency, making it suitable for relatively low-delay communication. The constellation's global coverage allows it to reach even remote and sparsely populated areas, providing Internet access to a wider population.

LEO satellite networks typically use mesh-like topology, in which there are several paths between every satellite and another satellite. The main advantages of this topology are flexibility and fault tolerance: data can be transmitted through different paths, ensuring system reliability and continuity even if a satellite fails. Figure 1 illustrates a typical edge-computing scenario in an LEO satellite network. UAVs or user terminals offload inference tasks to the LEO network via a user link, and these tasks are distributed among several satellites within the LEO network. The satellites are interconnected in full-mesh topology, which provides path redundancy and seamless coverage. For the sake of visual clarity, not all inter-satellite links are shown in the figure.



**Figure 1.** Edge computing in the LEO satellite network.

Due to the satellites being distributed around the Earth's orbit, there may be significant differences in communication delay between different satellites. The communication delay between satellites depends on the hop count, ISL bandwidth and signal propagation distance.

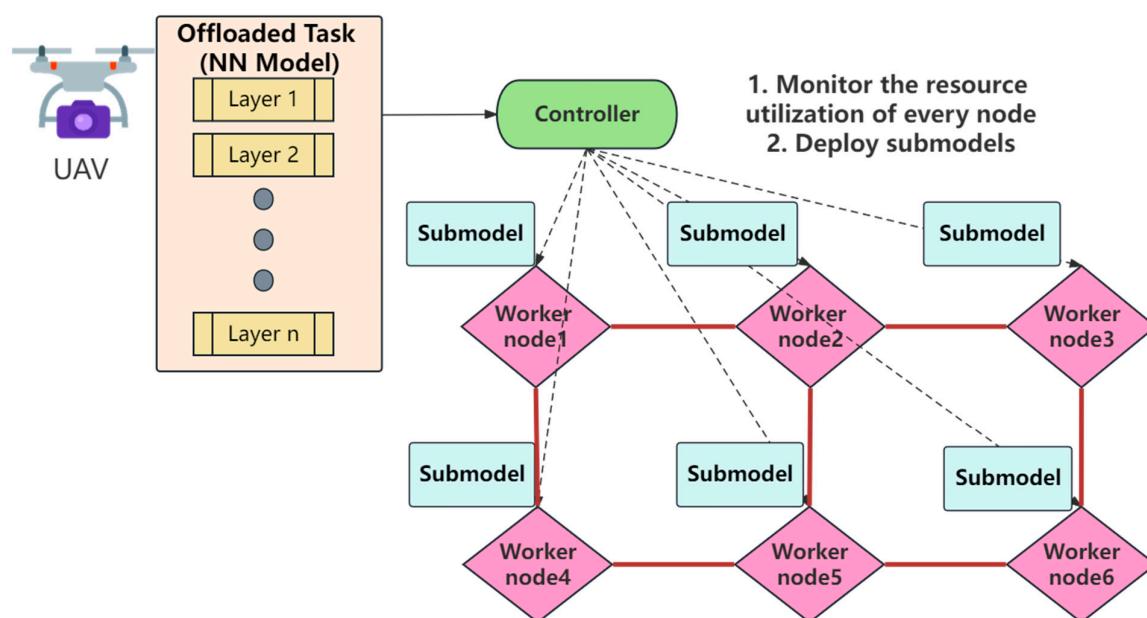
In order to provide a global edge-computing service, an important idea is to integrate edge computing into the LEO network [22]. Specifically, the edge-computing platform should be deployed on the LEO satellite. Additionally, the edge-computing platforms on every satellite should work collaboratively to form a distributed satellite-based edge cloud with abundant computing resources.

DNN inference is an important type of algorithm that will be run on the LEO satellite-based edge-computing platform. However, the computing power of an LEO satellite is less than that of an edge-computing node in the terrestrial network, so it is necessary to use multiple satellites' computing resources in order to complete massive inference tasks in a short time.

### 3.2. System Architecture

In order to implement multi-satellite collaborative inference, we adopt a simple collaborative inference system architecture consisting of a controller and several worker nodes [23].

As shown in Figure 2, the controller plays a crucial role in the collaborative inference system. It is a software runtime that can be deployed on a specific satellite, which is responsible for coordinating and managing the system's operations. It has a global view and can monitor the status and workload of each worker node, assigning tasks and allocating resources accordingly. The controller collects information, such as CPU and memory occupation from each worker node and based on this information, it intelligently schedules and distributes tasks to achieve faster inference and load balance.

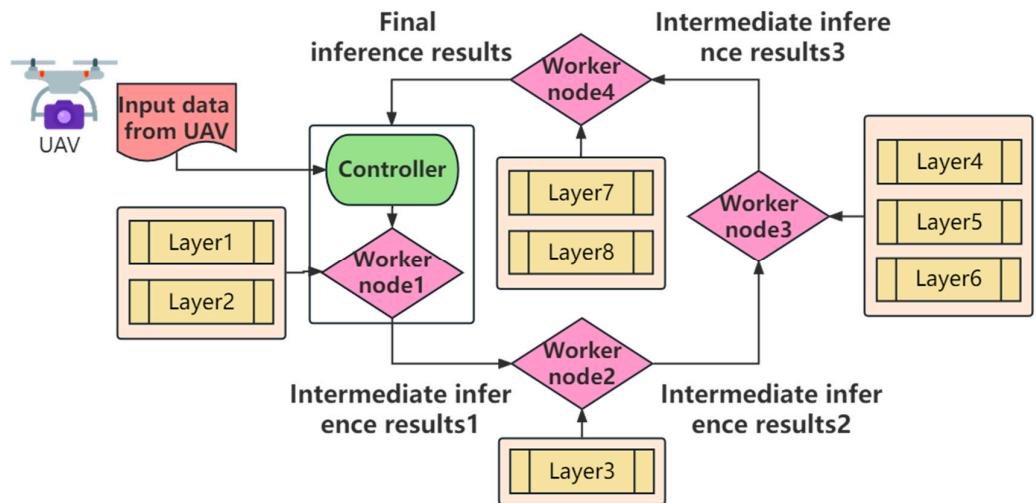


**Figure 2.** Collaborative inference system architecture.

The worker node is the LEO satellite-based edge-computing platform that carries out specific computing tasks, e.g., submodel inference. Each worker node has CPU and memory capacity. They can communicate and collaborate with each other to jointly accomplish complex DL inference tasks. The worker nodes are connected through high-speed satellite ISLs and switches, enabling intermediate result data transfer and DNN model parameter sharing. Such a distributed computing architecture can fully leverage the resources of multiple LEO satellites, improving overall computational efficiency and inference speed.

Before inference tasks begin, the controller intelligently splits the model into submodels suitable for different worker nodes based on their resource occupation and the submodel inference task requirements. These submodels are then assigned to the corresponding worker nodes for inference as shown in Figure 3. By well-designed model

partitioning and task assigning, we can balance the load of each worker node and enhance the overall performance and efficiency of the system.



**Figure 3.** A collaborative inference example.

Our model's operational process can be summarized in the following steps:

1. Traverse the directed acyclic graph (DAG) structure of the NN model to find all possible splitting points. These points usually represent branches or parallel computation nodes in the NN model that can be independently computed;
2. Based on the results from the found possible splitting point, perform the finest-grained splitting of the model and save the divided parts. This divides the model into multiple smallest parts, each of which can be independently computed on different computing nodes;
3. The controller collects information about the current CPU and memory occupation of each worker node. This information helps evaluate the availability and workload of each worker node;
4. Based on the collected information, use our algorithm to calculate the optimal NN model-splitting method. This algorithm takes into account factors such as resource occupation, load balancing and transmission delay to determine the most efficient way to split the NN model;
5. Based on the NN model-splitting result from the above step, perform the model splitting and assign submodels to worker nodes sequentially, in which each computing node is responsible for processing its assigned part of the model;
6. Send the submodels to the respective worker nodes and start the computation threads. Each worker node can independently carry out its own computation tasks using the original data or intermediate inference result output by another worker node;
7. The central controller sends data to the worker node that runs the inference of the first submodel. Intermediate/final result data flows between worker nodes, and each node performs computations in the specified order, thereby accomplishing the NN model's inference task in a pipelined fashion.

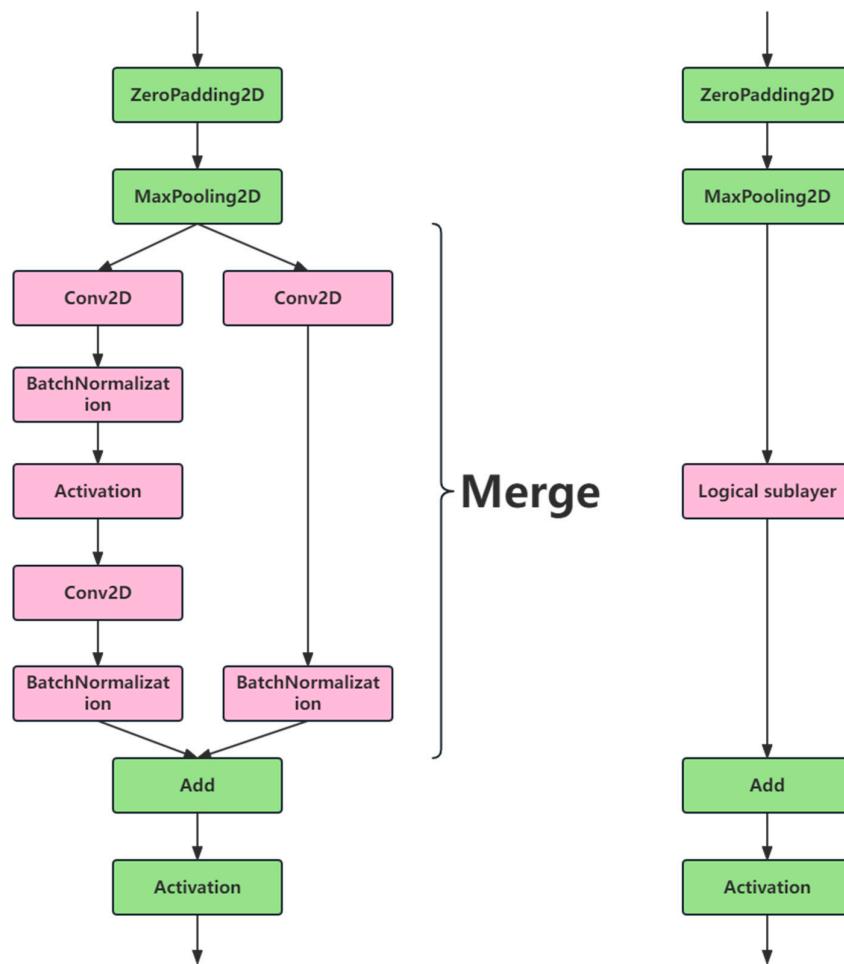
Through these steps, our collaborative inference system achieves efficient distributed DL inference tasks. By dividing the model and utilizing parallel and pipelined computations, we fully leverage the resources of multiple satellites, thereby improving the speed of model inference.

### 3.3. Analysis of Inference Process

Next, we explain how to split complex DNN models whose structures are not strictly linear.

Modern DNNs are not simple linear structures but rather DAGs. A DAG is a representation of a graph composed of a set of vertices and directed edges, with no cycles. In a DAG, each directed edge has a direction, pointing from one vertex to another, and there are no paths that start at a vertex, traverse through several edges and return to the same vertex. The structure of a DNN model can be modeled as a DAG. In a DNN, each node represents a neuron or computing units, and directed edges represent the connections between neurons.

To achieve model partitioning and parallel computation, we first use DAG traversal to find nodes that can be split. DAG traversal involves sequentially traversing nodes in a computational graph to determine the execution order and the dependency relationships of the computations. In DL, DAG traversal is commonly used in the forward- and backward-propagation processes of the model to compute the output values and gradients of each node. As shown in Figure 4, we adopt the current mainstream and intuitive approach, using traversal to find model split points and treating the layers between two split points as a logical layer. This approach works well for models with residual blocks, such as ResNet or PANet.



**Figure 4.** Merge branches into logical sub-layers.

Currently, some researchers consider not transforming the DAG into a chain but decomposing it into a set of execution units [24]. In that way, edge-computing nodes are responsible only for executing specific operations, and results are sent to the corresponding targets through forwarding tables. However, this scheduling method would introduce significant network overhead, and for our LEO satellite network scenario, such network overhead may even exceed the DNN inference time, which is unacceptable. Therefore, we did not use this method and instead opted for partitioning based on logical layers.

Moreover, this logical layer-based partitioning method simplifies the complexity of our subsequent algorithms.

#### 4. COIN-LEO Algorithm

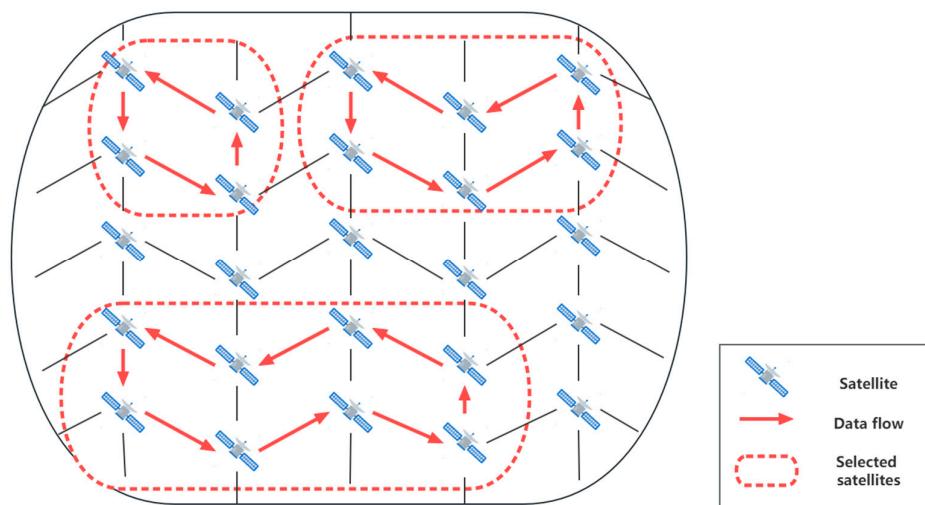
Each satellite can be viewed as an independent worker node. We aim to design an algorithm that can split the model into several submodels and deploy different submodels to different worker nodes. These submodels are deployed to several worker nodes for distributed inference, forming a pipelined inference cluster.

The proposed COIN-LEO algorithm mainly consists of two parts. The first part is to select appropriate satellites to participate in collaborative inference. The second part is to determine the specific model-splitting positions in order to deploy submodels to the above-selected satellites. To predict the inference time for different submodels of varying sizes under different resource-occupation conditions of each satellite, we design a DNN that can make accurate predictions based on the satellite's resource occupation.

##### 4.1. Selecting Satellite for Collaborative Inference

As mentioned above, the LEO satellite network adopts a mesh-like structure. The one-way propagation delay for each hop of LEO satellites is approximately 7–15 ms. As for DNN inference tasks, the inference time can vary from tens of milliseconds to several seconds, depending on the model's size. For some relatively simple inference tasks, this propagation delay cannot be ignored [25]. Therefore, our method of selecting satellites tends to reduce the propagation delay in the collaborative inference, specifically the number of satellites that data/intermediate results have to travel through.

To achieve the desired inference delay and throughput performance, we can choose a structure in which multiple satellites are sequentially connected. By forming a circular structure in which satellites are connected end-to-end, the sum of all ISL propagation delays is reduced, as shown in Figure 5. When one satellite completes a part of an inference task, the intermediate result can be directly transmitted to the next satellite node without requiring additional transmission paths. This effectively controls the transmission delay of intermediate results and model parameters, improving the system's response speed and inference efficiency. Additionally, this circular structure can better adapt to the dynamic changes in the satellite network. When new satellite nodes join or leave the system, the topology structure can be adjusted and adapted relatively flexibly.



**Figure 5.** Examples of selecting satellites for collaborative inference.

We choose to connect four satellites end-to-end as our structure. This decision was made considering that as the number of satellite nodes increases, the corresponding transmission delays also increase. Considering the model size and inference speed, using too

many satellite nodes may have a negative impact on throughput and other performance metrics. To strike a balance, we opt to use four satellites for distributed deep-learning inference. On one hand, the one-way propagation delay of the ISL ranges from 7 to 15 ms and the transmission delay increases linearly as the number of satellites increases. In the scenario with four satellites, the transmission delay for intermediate results to traverse each satellite and return to the initial satellite is approximately 28–60 ms, which is within an acceptable range. On the other hand, if the number of satellites is too low, it would hinder the advantages of collaborative inference and make it challenging to improve the system's throughput. If the number of satellites is even fewer than four, it would not achieve the expected benefits of collaborative inference. Considering these factors collectively, this configuration allows us to maintain a reasonable trade-off between the number of nodes and the overall performance of the system.

#### 4.2. Deep Reinforcement Learning Using the PPO Algorithm

Reinforcement learning (RL) is an important branch of machine learning, whose goal is to enable an agent to learn how to make decisions in a specific environment through interaction with the environment in order to maximize cumulative rewards. Reinforcement learning is a method of learning the optimal strategy through trial and error, similar to how humans try different actions during the learning process to achieve the best results. In reinforcement learning, the agent interacts with the environment, observes the current state and takes actions based on the current state. After taking an action, the agent receives an immediate reward or punishment to evaluate the goodness of that action. The goal of the agent is to learn the policy that maximizes the cumulative reward, which means selecting the optimal action in each state to achieve the best decision-making outcome.

Deep reinforcement learning (DRL) is a specific subset of reinforcement learning (RL). It involves using deep neural networks as function approximators to learn complex policies or value functions from high-dimensional state spaces. The use of deep neural networks allows DRL to handle large and complex environments, making it suitable for tasks that involve raw sensory input.

The key components of deep reinforcement learning include:

1. Environment: The environment is the context in which the agent operates. It can be a simulated environment, a real-world scenario, or even a virtual game environment;
2. State: The state represents the current condition of the environment, which the agent observes to make decisions;
3. Action: The action is the decision made by the agent based on the observed state. It determines the agent's interaction with the environment;
4. Reward: The reward is a scalar value that provides feedback to the agent after taking an action. It serves as a measure of how well the agent is performing and guides its learning process;
5. Policy: The policy is the strategy or behavior that the agent uses to select actions based on the observed state;
6. Value Function: The value function estimates the expected long-term return or reward from a particular state.

For the modeling of this problem, we provide the following symbol interpretations:

$S$ : The set of states in the reinforcement learning model.  $S : \{\vec{c}, \vec{m}, \vec{l^l}, \vec{b}, p, i\}$ ;

$\vec{c}$ : The remaining CPU resources of each satellite in the cluster;

$\vec{m}$ : The remaining memory resources of each satellite in the cluster;

$\vec{l^l}$ : The signal propagation delay of each inter-satellite link in the cluster;

$\vec{b}$ : The remaining link rate of each inter-satellite link;

$p$ : The layer number before the last splitting point (the last action) in the neural network model;

$i$ : The step number (also the satellite number to place the model);

A: The set of actions in the reinforcement learning model. A:  $\{a \in [p + 1, L]\}$ ;  
 a: Splitting the model after layer a;  
 L: The total number of layers in the neural network model;  
 $dp_{p+1,a}^i$ : Putting the layer from p + 1 to a on the i th satellite, the processing latency of the section;  
 $dt_{i,a}$ : Latency of transmitting the intermediate results of layer a from the i th satellite to the next satellite;  
 $d^{stream}$ : Time for the entire inference process;  
 $d_{max}^{stream}$ : Configurable upper time limit for the reasoning process.

#### 4.2.1. State

In the reinforcement learning model, the state set S is composed of multiple state variables, which are used to describe the current situation of the environment. Specifically, the state set S contains the following state variables:  $S : \{\vec{c}, \vec{m}, \vec{l}, \vec{b}, p, i\}$ . Among these state variables,  $\vec{c}$  and  $\vec{m}$  describe the resource status of each satellite in the cluster, specifically;  $\vec{c}$  and  $\vec{m}$  represent the remaining CPU and memory resources on each satellite;  $\vec{l}$  and  $\vec{b}$  describe the transmission status of inter-satellite links; and p and i are used to track information about the previous splitting point and the current step, respectively. These state variables together form the state set S of the environment. In reinforcement learning, the agent observes the current state S to make decisions and choose the optimal action.

#### 4.2.2. Action

The action set A is used to describe all possible actions that the agent can take in a specific state. Specifically, the action set A is composed of an integer set, representing the action of splitting the neural network model at layer a to partition it into different parts. Here, a represents the action of splitting the model after layer a, where  $a \in [p + 1, L]$ , and p is the layer number before the last splitting point (the last action), while L is the total number of layers in the neural network model.

This designed action set A allows the agent to decide at which layer to perform the splitting in each state in order to further partition the model into appropriate submodels. The choice of actions directly impacts the model's structure and inference performance. During the training process of reinforcement learning, the agent will select an action a from the action set A based on the current state S, thus generating a sequence of actions to complete the model partitioning.

#### 4.2.3. Reward

The reward function R(S, A) is used to evaluate the actions A and the state S taken by the agent in reinforcement learning and provide corresponding rewards or penalties. This reward function considers two factors: data flow delay and model inference time.

Specifically, the reward function R(S, A) is defined as follows:

$$R(S, A) = \begin{cases} \frac{1}{dp_{p+1,a}^i + dt_{i,a}} & \text{if } i < 3 \\ \frac{1}{dp_{p+1,a}^i + dt_{i,a} + dp_{a+1,L}^i} & \text{if } i = 3 \text{ and } d^{stream} \leq d_{max}^{stream} \\ -d^{stream} & \text{if } i = 3 \text{ and } d^{stream} > d_{max}^{stream} \end{cases} \quad (1)$$

In the first case ( $i < 3$ ), the reward function considers a combination of data flow delay and model inference time to encourage early completion of the inference task and reduce latency. In the last splitting case ( $i = 3$ ), two submodels are generated, and the second case takes this scenario into account. In the third case ( $i = 3$ ), if  $d^{stream} > d_{max}^{stream}$ , the reward function provides a negative reward, i.e., a penalty, to avoid exceeding the maximum allowable data flow delay and ensure the effectiveness and real-time nature of the inference task.

Through the design of the reward function, the agent can choose the optimal actions in different states to efficiently complete the inference task and maximize cumulative rewards, which aims to optimize inference performance and latency effects. In this way, the agent can gradually learn and optimize strategies during the training process of reinforcement learning to adapt to different environments and task requirements.

#### 4.2.4. Proximal Policy Optimization (PPO) Algorithm

We use the proximal policy optimization (PPO) algorithm for reinforcement learning [26]. PPO is a commonly used reinforcement learning algorithm designed to address the policy optimization problem. The main feature of the PPO algorithm is to update the policy through proximal clipping, ensuring that policy improvements are not too drastic and avoiding instability. The goal of the PPO algorithm is to improve the performance of the current policy as much as possible while maintaining the similarity of the policy and avoiding significant changes during policy updates.

The objective of PPO is to maximize the expected cumulative reward, denoted by  $J(\theta)$ , where  $\theta$  represents the policy parameters. This can be formulated as follows:

$$J(\theta) = E[R(t)], \quad (2)$$

where  $R(t)$  represents the cumulative reward from time step 0 to time step  $t$ .

To optimize the objective function  $J(\theta)$ , we use the PPO algorithm for policy updates. The core idea of PPO is to constrain the policy updates using proximal clipping to enhance algorithm stability.

Specifically, the PPO algorithm uses two different objective functions for policy updates, namely the clipped surrogate objective and PPO-clip objective.

##### 1. Clipped Surrogate Objective

The Clipped Surrogate Objective is the core objective function of the PPO algorithm, used for computing the policy gradient. When updating the policy, the PPO algorithm calculates the policy gradient by comparing the probability ratio between the new policy and the old policy, then constrains it within a certain range, which is called proximal clipping. The purpose of this is to ensure that the improvement of the new policy relative to the old policy is not too large, thereby enhancing the stability of the algorithm.

The clipped surrogate objective is used to calculate the policy gradient and is defined as:

$$L^\theta(s, a) = \min\left(\frac{\pi'(a|s)}{\pi(a|s)} \cdot A^\pi(s, a), \text{clip}\left(\frac{\pi'(a|s)}{\pi(a|s)}, 1 - \epsilon, 1 + \epsilon\right) \cdot A^\pi(s, a)\right) \quad (3)$$

where  $\pi'(a|s)$  represents the probability of selecting action  $a$  under the old policy in state  $s$ ,  $\pi(a|s)$  represents the probability of selecting action  $a$  under the new policy in state  $s$  and  $A^\pi(s, a)$  represents the advantage of selecting action  $a$  in state  $s$ . “clip” refers to the operation of bounding or limiting a value within a specified range. The purpose of using clipping in the PPO algorithm is to control the magnitude of policy updates to enhance the stability of the learning process.

##### 2. PPO-Clip Objective

The PPO-clip objective is a variant of the clipped surrogate objective, used to compute the optimization objective for policy updates. The PPO-clip objective is the minimum value between the clipped surrogate objective and the KL divergence. It constrains the policy update using proximal clipping and KL divergence to ensure that the magnitude of policy updates remains within a reasonable range, thereby preventing the algorithm from over-optimizing.

$$L^\theta(s) = \left[ \min\left(L^\theta(s, a), \text{clip}\left(\frac{\pi'(a|s)}{\pi(a|s)}, 1 - \epsilon, 1 + \epsilon\right)\right) \right] \quad (4)$$

where  $L^\theta(s)$  represents the PPO-clip objective in state  $s$ ,  $\pi'(a|s)$  represents the probability of selecting action  $a$  under the old policy in state  $s$  and  $\pi(a|s)$  represents the probability of selecting action  $a$  under the new policy in state  $s$ .

With the policy updates using proximal clipping, the PPO algorithm can achieve good performance and stability in large-scale and complex reinforcement learning tasks. This makes PPO one of the widely used algorithms in the field of reinforcement learning.

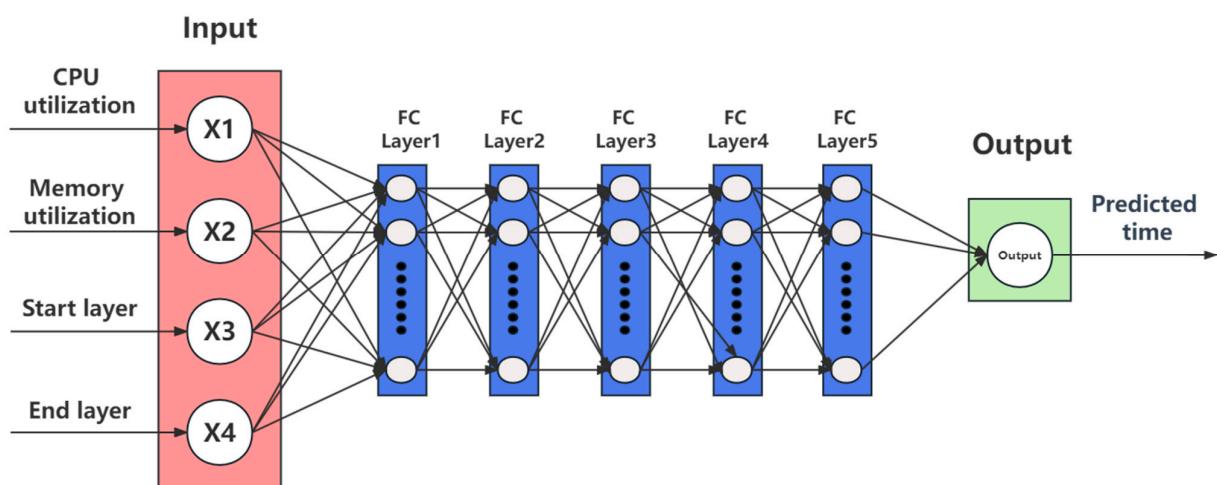
Due to the policy update method of proximal clipping, the PPO algorithm exhibits good stability and is suitable for large-scale and complex reinforcement learning tasks. The PPO algorithm performs well in tasks with discrete action spaces, which aligns with the discrete action space in our training scenario. The PPO algorithm also has a relatively fast training speed and can achieve good performance in a fewer number of sample iterations.

Overall, the PPO algorithm is a very practical and effective reinforcement learning algorithm that has made significant progress in various application domains. Its excellent performance and stability make it a favorable choice for our reinforcement learning training. Indeed, the algorithm has shown promising results in our context.

#### 4.3. DNN Model Used to Predict Submodel Inference Times

Traditional time prediction methods simply use the data size divided by the theoretical CPU computing power. However, this method is overly idealized and fails to reflect the complex impact of real-world scenarios and CPU task scheduling on inference time, making it difficult to estimate accurately. Therefore, we trained a deep neural network to obtain more accurate inference time predictions. The DNN model takes into account various factors and features related to the hardware and environment, enabling us to achieve more precise and reliable inference time estimates compared to the simplistic traditional approach.

We built a deep neural network model to predict the time required for inference tasks on low-Earth-orbit satellites. The model is based on Keras and is a sequential model composed of multiple dense layers. Firstly, the model includes an input layer with a feature dimension of 4, representing the CPU occupation, memory occupation, starting layer for model partitioning and ending layer for model partitioning on the satellite. Next, as shown in Figure 6, we have four hidden layers, each containing 256 neurons with the rectified linear unit (ReLU) activation function, which increases the depth of the model and helps extract higher-level feature representations. Finally, the model outputs the predicted inference time based on the regression problem being solved.



**Figure 6.** DNN model used to predict submodel inference times. (FC Layer: fully connected layer).

We used our collected data as the training data for the deep neural network model. We designed an algorithm that can occupy the machine's CPU and memory. We partitioned the model into the smallest units and then combined them into different-sized models to

perform inference under different CPU and memory occupation rates, collecting actual execution time data for training purposes.

We observed that when the CPU occupation increases by more than 20%, there is a significant increase in the inference execution time, which gradually returns to normal afterward. This may be caused by system scheduling. However, when our model is in the pipeline, this one-time increase can be averaged over a large amount of data flow, so it can be considered to have a relatively small impact on the actual inference process. But this kind of anomalous data can affect our model training results, so we remove such data before training.

During the model training process, we used mean-squared Error (MSE) as the loss function to measure the difference between the model's predicted values and the true values. After training, the model's MSE on the validation set was around 50. This MSE value indicates the average squared error between the model's predicted results and the true inference time, which means the actual error is within  $\pm 50$  milliseconds. Although there is still some error, considering the special environment of the low-Earth-orbit satellite Internet, with limited resources and communication delays, our model has achieved good prediction results, which are sufficient for reinforcement learning training.

## 5. Performance Evaluation

### 5.1. Experiment Setup

In this section, we set up the experimental test environment to evaluate our approach. We built a small-scale cluster with four nodes on a cloud platform to simulate the scenario of four satellites. Our cloud platform is a self-developed satellite network simulation platform based on cloud computing. It is highly efficient and capable of simulating link connectivity, latency and bandwidth variations. With this platform, we can create realistic scenarios and accurately evaluate the performance of satellite networks under various conditions, allowing us to make informed decisions and optimize the network design and operations. To simulate a realistic low-Earth-orbit satellite environment, we set the inter-satellite link delay to 15 ms, and the inter-satellite link rate is 50 MB/s. The software environment includes Python 3.10, TensorFlow 2.10.0, Keras 2.10.0 and Gymnasium 0.28.1. With this cluster in place, we varied the CPU and memory occupation rates of the environment to create different scenarios.

Because there are no other algorithms of collaborative inference for LEO edge computing in the literature, we choose two baseline algorithms for comparison. The first baseline algorithm is called the equally splitting strategy, in which the complete model is split into four submodels that have the same number of layers. The second baseline algorithm is called the randomly splitting strategy, in which the layers of the complete model are randomly split. The strategy for selecting satellites for collaborative inference is the same as COIN-LEO. Through these comparisons, we sought to evaluate the robustness and adaptability of our approach in varying resource-occupation conditions. The goal was to identify how well our model could handle fluctuations in CPU and memory usage and whether it could consistently make efficient decisions for model partitioning in diverse settings.

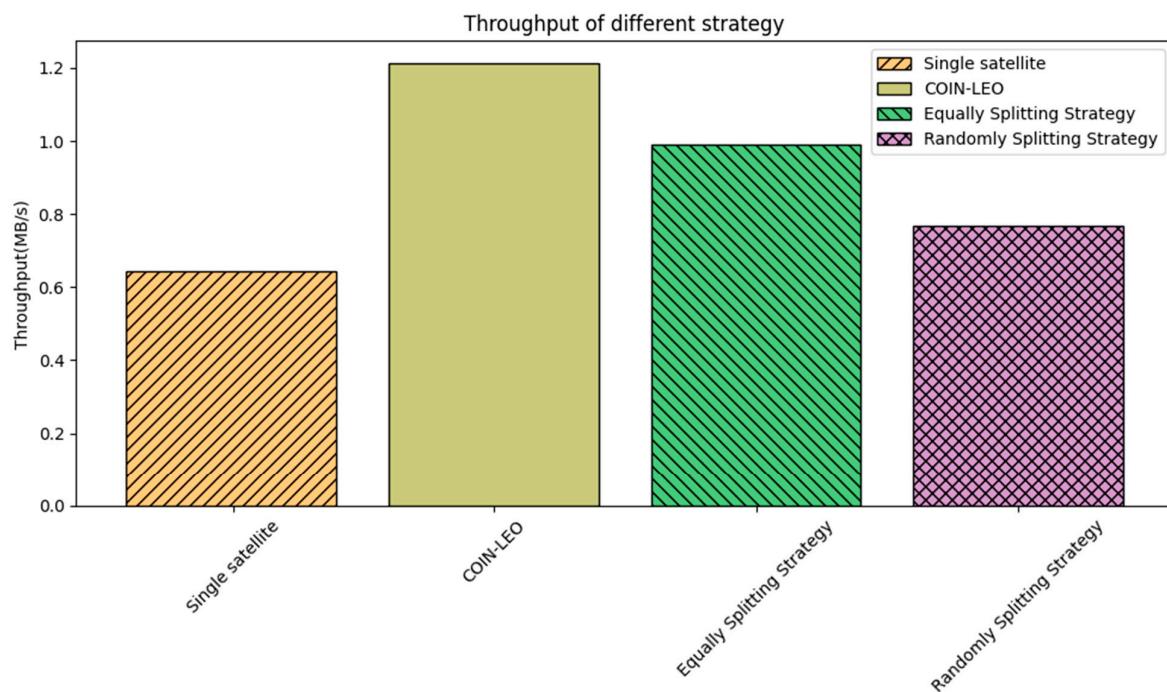
We conducted tests in four different scenarios. In the first scenario, each node's CPU occupation was around 1%, representing an idle state, while the memory occupation was approximately 34%. In the second scenario, the CPU occupation was about 25%, and the memory occupation was around 45%. Moving on to the third scenario, the CPU occupation increased to approximately 50%, while the memory occupation remained at 45%.

The fourth scenario was unique; we kept two nodes idle, and for the other two nodes, we increased the CPU occupation to 50% and the memory occupation to 45%. This setup allowed us to form a meaningful comparison against the equal distribution strategy.

### 5.2. Results

Firstly, we compare the inference throughput of collaborative inference and inference by a single node. The inference throughput is calculated as the ratio between the data

size of all the images inferred and the total time consumed. It is measured by completing 1000 inference tasks continuously and recording the total time. Furthermore, the inference performance by a single satellite (not collaborative inference) is presented. The inference throughput of every strategy is shown in Figure 7. As shown in Figure 7, the inference throughput was only 0.6423 MB/s when all inference was performed on a single node. In contrast, when using the random distribution strategy for collaborative inference, the throughput reached 0.7506 MB/s. When using the equally splitting strategy for collaborative inference, the inference throughput reached 0.9738 MB/s. And when using our COIN-LEO for collaborative inference, the inference throughput reached 1.1935 MB/s, which is the highest among all the model-splitting strategies. Compared to the single-node inference performance, the improvement in inference throughput is as high as 85.80% with our COIN-LEO strategy. This indicates a significant improvement in inference throughput using distributed collaborative DNN inference, which is because collaborative inference can utilize several satellites' computing resources.

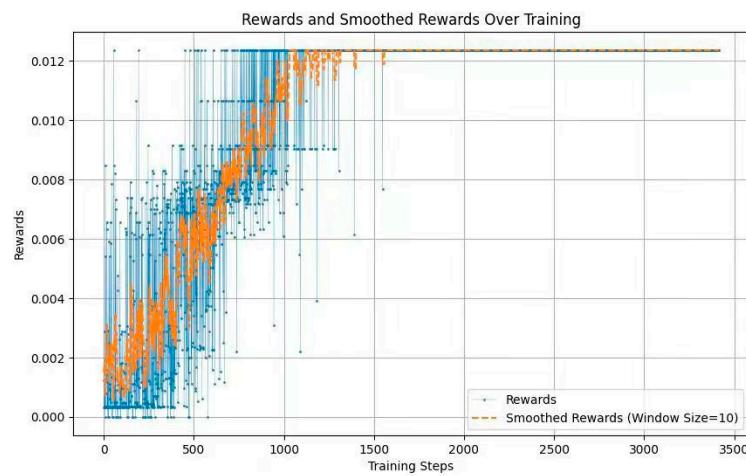


**Figure 7.** Throughput of different strategy.

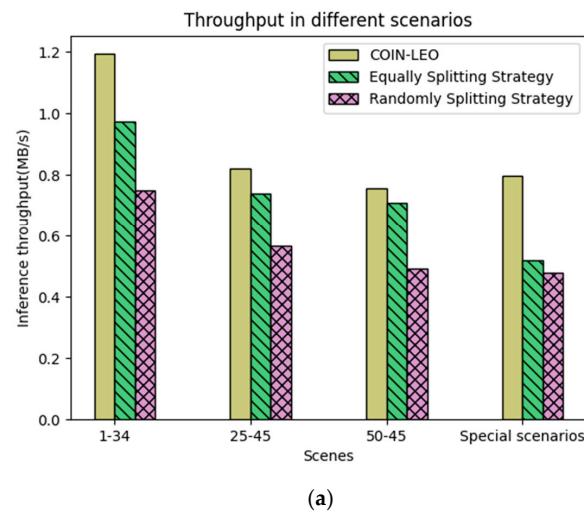
Figure 8 illustrates the evolution of rewards during the training process of our algorithm. We conducted 100,000 training iterations with sampling performed every 10 iterations. We used a 'done' return value of true in DRL as an indicator of the end of an episode step. As depicted in Figure 8, the blue line represents the rewards, while the orange line represents the smoothed reward with a window size of 10. From Figure 8, it is evident that COIN-LEO's reward exhibits impressive increase and convergence.

Then, we compare the performance of COIN-LEO with the other baseline strategies. The inference throughput and time of every strategy are shown in Figure 9a,b, respectively. As shown in Figure 9a, the inference throughput of COIN-LEO is the highest among all the strategies in every scenario, which clearly demonstrates the effectiveness of our strategy in improving throughput. Compared to the equally splitting algorithm, our proposed COIN-LEO showed an average improvement of 17.285% in inference time. Furthermore, compared to the random allocation algorithm, our improvement was even higher, with an average increase of 35.787%. In the special scenario we set, the algorithm's improvement was most pronounced, showing a 35.665% increase compared to the equal allocation algorithm. These results highlight the superior performance of our algorithm in relatively complex operational environments. Our approach outperformed the equal distribution strategy,

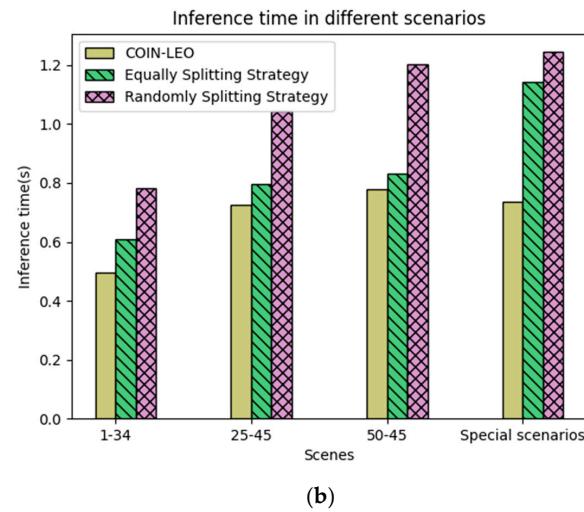
showcasing the benefits of our optimized resource allocation and model-partitioning techniques. With these improvements, we achieved higher throughput, indicating more efficient and faster model inference in the distributed environment.



**Figure 8.** Rewards and smoothed rewards when training COIN-LEO.



(a)



(b)

**Figure 9.** Throughput and inference time for different scenarios. (a) Throughput for different scenarios; (b) inference time for different scenarios.

We believe that one of the reasons for such a performance is that COIN-LEO can allocate tasks more efficiently based on the remaining resources on each satellite, as shown in Table 1. By considering the CPU and memory occupation of each satellite, COIN-LEO intelligently distributes the inference tasks to maximize the utilization of available resources. This allows for a more balanced and optimized workload distribution among the satellites, leading to improved inference efficiency and overall system performance. In Table 1, we show the average and median durations for each satellite to perform submodel inference in scenarios with CPU and memory occupation rates of 12–38, 50–45, 2–32 and 50–43, respectively. In the last column, we calculate the percentages relative to the results using the random distribution strategy as the baseline for other strategies. Since the second and fourth satellites in our scenarios have relatively high CPU and memory occupancy rates, logically they should be assigned relatively lighter inference tasks. The COIN-LEO algorithm indeed met our expectations. As observed, the COIN-LEO algorithm assigned the heaviest inference task to the most resource-abundant satellite, which was the third one, while the other satellites were responsible for inference tasks on the scale of tens of milliseconds. This allocation strategy fully utilized the remaining resources. In contrast, the equal distribution strategy performed poorly in this case. The random distribution strategy even assigned the heaviest task to the heavily loaded fourth satellite, which undoubtedly affected the final inference speed.

**Table 1.** Time statistics under different strategies on different satellites.

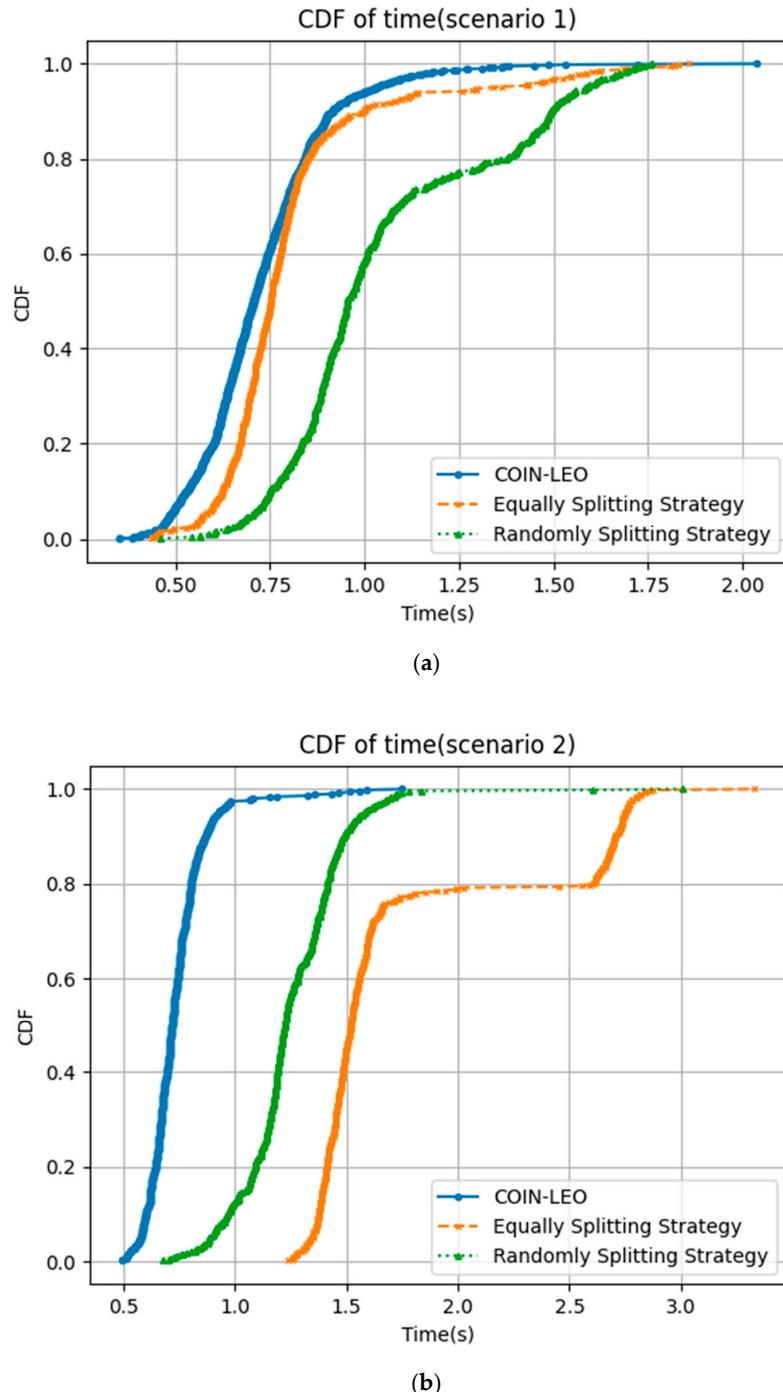
Scenarios	Satellite	Strategy	Mean Value	Median Value	Percentage
CPU-memory occupation: 12–38, 50–45, 2–32, 50–43	1	COIN-LEO	63.524	59.0	49.58%
		Equally splitting	252.695	251.0	210.92%
		Randomly splitting	123.876	119.0	100%
	2	COIN-LEO	41.512	33.0	9.40%
		Equally splitting	1585.0	1600.0	455.84%
		Randomly splitting	417.2	351.0	100%
	3	COIN-LEO	1939.063	1940.0	715.86%
		Equally splitting	1408.962	1557.0	574.53%
		Randomly splitting	279.388	271.0	100%
	4	COIN-LEO	49.706	49.0	2.97%
		Equally splitting	366.084	363.0	22.04%
		Randomly splitting	1644.699	1647.0	100%

Next, Figure 10 shows the cumulative distribution function (CDF) of time consumed for every task plot for different strategies under various scenarios. These CDF plots provide a visual representation of how the strategies perform in terms of inference time distribution across different satellites. By comparing the CDF plots for each strategy, we can clearly observe the differences in their performance and assess the effectiveness of COIN-LEO in improving inference efficiency across different resource-occupation scenarios.

From the CDF plot, we can observe that when using our COIN-LEO strategy, the inference process of every task is faster compared to other strategies. The curve representing our strategy generally starts at a higher point on the *x*-axis, indicating that a larger proportion of inference times are lower compared to the other strategies.

This means that our approach efficiently distributes the inference workload, resulting in more instances of faster inference times. We believe that these results can be attributed to our algorithm for two main reasons. Firstly, our algorithm performs reasonable allocation by deploying heavier tasks on satellites with more available resources, thus reducing the frequency of CPU thread switches and ultimately improving inference speed. Secondly,

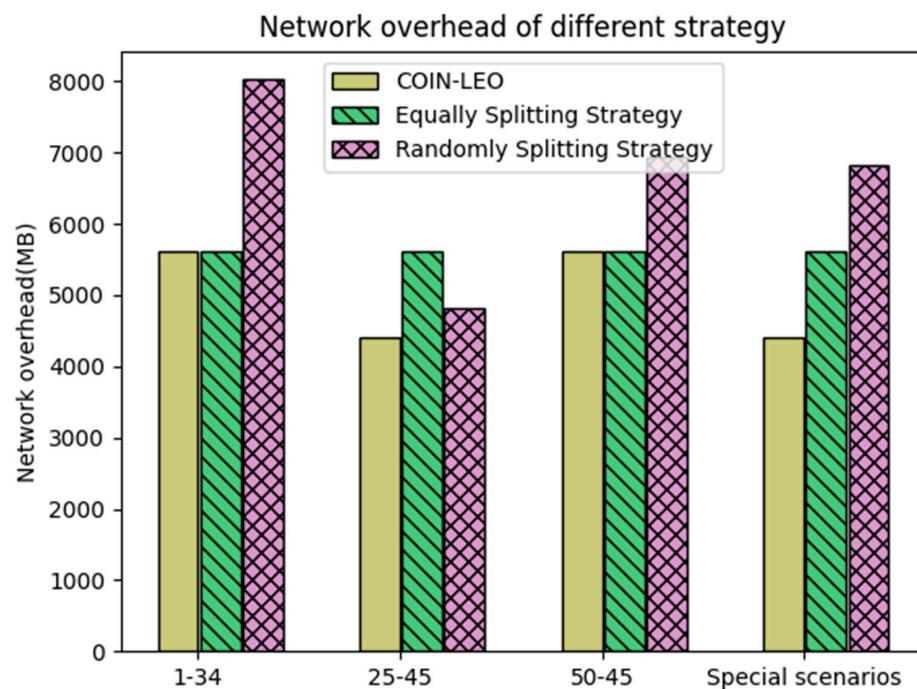
our algorithm generates relatively smaller traffic sizes. This also reduces data-transmission time requirements, enhancing system throughput. As a result, the CDF plot confirms that our strategy is more effective in minimizing the overall inference time and improving the performance of the distributed deep-learning inference system.



**Figure 10.** CDF diagrams for different scenarios. (a) CPU memory occupation: 36–46, 26–45, 25–46, 26–46; (b) CPU memory occupation: 12–38, 50–45, 2–32, 50–43.

As shown in Figure 11, we also conduct a comprehensive comparison of traffic sizes under different strategies. The flow of data in our experimental scenarios is derived from the intermediate results of inference generated by each satellite. We carefully analyzed and recorded the traffic sizes for each strategy in all four scenarios. Our COIN-LEO strategy demonstrated its advantage in optimizing traffic size by efficiently distributing

computational tasks across the satellite network. This is because our designed reward function tries to reduce the time for complete collaborative inference, which leads to reduced transmission delay and traffic size. These results underscore the effectiveness of our proposed algorithm in minimizing the overall traffic burden and enhancing the overall performance of the distributed deep-learning system.



**Figure 11.** Network overhead of different strategy.

## 6. Conclusions

This paper focuses on addressing the problem of collaborative DL inference in LEO satellite edge computing. We propose the COIN-LEO algorithm, specifically designed for efficient model partitioning and task assigning among satellites. To achieve this, we trained a five-layer fully connected NN to accurately predict inference time based on the remaining resources of each satellite. Through a series of experiments, we evaluate the performance of our proposed COIN-LEO algorithm in different resource-occupation scenarios. The results demonstrate that COIN-LEO effectively improves inference efficiency by intelligently assigning tasks based on resource-occupation rate.

In future research, we plan to further enhance our approach by incorporating a broader range of DL models and diverse datasets. This expansion aims to explore the strategy's adaptability and generalization across various real-world scenarios, ensuring robustness and effectiveness in different satellite constellations and dynamic environments.

**Author Contributions:** Formal analysis, H.Z.; Methodology, P.Z., C.L., G.X. and C.S.; Software, Z.X. All authors have read and agreed to the published version of the manuscript.

**Funding:** This work is partially supported by the Natural Science Foundation of Shandong Province under Grant ZR2022LZH015 and ZR2020MF006, partially supported by the Industry-university Research Innovation Foundation of Ministry of Education of China under Grant 2021FNA01001, partially supported by the Open Foundation of State Key Laboratory of Integrated Services Networks (Xidian University) under Grant ISN23-09, partially supported by the Key Funding from National Natural Science Foundation of China under Grant 92067206.

**Data Availability Statement:** Data sharing is not applicable.

**Conflicts of Interest:** The authors declare no conflict of interest.

## References

- Zhu, X.; Jiang, C. Integrated Satellite-Terrestrial Networks Toward 6G: Architectures, Applications, and Challenges. *IEEE Internet Things J.* **2022**, *9*, 437–461. [[CrossRef](#)]
- Xiao, Z.; Yang, J.; Mao, T.; Xu, C.; Zhang, R.; Han, Z.; Xia, X.-G. LEO Satellite Access Network (LEO-SAN) towards 6G: Challenges and Approaches. *IEEE Wirel. Commun.* **2022**, *1*–8. [[CrossRef](#)]
- Zhou, D.; Sheng, M.; Li, J.; Han, Z. Aerospace Integrated Networks Innovation for Empowering 6G: A Survey and Future Challenges. *IEEE Commun. Surv. Tutor.* **2023**, *25*, 975–1019. [[CrossRef](#)]
- Del Portillo, I.; Cameron, B.G.; Crawley, E.F. A Technical Comparison of Three Low Earth Orbit Satellite Constellation Systems to Provide Global Broadband. *Acta Astronaut.* **2019**, *159*, 123–135. [[CrossRef](#)]
- Zhao, L.; Zhang, Q.; Li, Y.; Qi, Y.; Yuan, X.; Liu, J.; Li, H. China’s Gaofen-3 Satellite System and Its Application and Prospect. *IEEE J. Sel. Top. Appl. Earth Obs. Remote Sens.* **2021**, *14*, 11019–11028. [[CrossRef](#)]
- Singh, P.; Diwakar, M.; Shankar, A.; Shree, R.; Kumar, M. A Review on SAR Image and Its Despeckling. *Arch. Comput. Methods Eng.* **2021**, *28*, 4633–4653. [[CrossRef](#)]
- Li, C.; Zhang, Y.; Xie, R.; Hao, X.; Huang, T. Integrating Edge Computing into Low Earth Orbit Satellite Networks: Architecture and Prototype. *IEEE Access* **2021**, *9*, 39126–39137. [[CrossRef](#)]
- Kim, T.; Choi, J.P. Performance Analysis of Satellite Server Mobile Edge Computing Architecture. In Proceedings of the 2020 IEEE 92nd Vehicular Technology Conference (VTC2020-Fall), Virtual, 18 November–16 December 2020; IEEE: Victoria, BC, Canada, 2020; pp. 1–6.
- Zhang, P.; Chen, N.; Shen, S.; Yu, S.; Kumar, N.; Hsu, C.-H. AI-Enabled Space-Air-Ground Integrated Networks: Management and Optimization. *IEEE Netw.* **2023**, *1*–7. [[CrossRef](#)]
- Wu, G.; Xu, Z.; Zhang, H.; Shen, S.; Yu, S. Multi-Agent DRL for Joint Completion Delay and Energy Consumption with Queuing Theory in MEC-Based IIoT. *J. Parallel Distrib. Comput.* **2023**, *176*, 80–94. [[CrossRef](#)]
- Shao, J.; Zhang, J. Communication-Computation Trade-off in Resource-Constrained Edge Inference. *IEEE Commun. Mag.* **2020**, *58*, 20–26. [[CrossRef](#)]
- Lin, C.-Y.; Wang, T.-C.; Chen, K.-C.; Lee, B.-Y.; Kuo, J.-J. Distributed Deep Neural Network Deployment for Smart Devices from the Edge to the Cloud. In Proceedings of the ACM MobiHoc Workshop on Pervasive Systems in the IoT Era, Catania, Italy, 2 July 2019; ACM: Catania, Italy, 2019; pp. 43–48.
- Khan, M.A.; Hamila, R.; Erbad, A.; Gabbouj, M. Distributed Inference in Resource-Constrained IoT for Real-Time Video Surveillance. *IEEE Syst. J.* **2023**, *17*, 1512–1523. [[CrossRef](#)]
- Kim, G.-Y.; Kim, R.; Kim, S.; Nam, K.-D.; Rha, S.-U.; Yoon, J.-H. DNN Inference Offloading for Object Detection in 5G Multi-Access Edge Computing. In Proceedings of the 2021 International Conference on Information and Communication Technology Convergence (ICTC), Jeju Island, Republic of Korea, 20 October 2021; IEEE: Jeju Island, Republic of Korea, 2021; pp. 389–392.
- Yu, M.; Jiang, Z.; Ng, H.C.; Wang, W.; Chen, R.; Li, B. Gillis: Serving Large Neural Networks in Serverless Functions with Automatic Model Partitioning. In Proceedings of the 2021 IEEE 41st International Conference on Distributed Computing Systems (ICDCS), Washington, DC, USA, 7–10 July 2021; IEEE: Washington, DC, USA, 2021; pp. 138–148.
- Wang, Y.; Zhang, J.; Zhang, X.; Wang, P.; Liu, L. A Computation Offloading Strategy in Satellite Terrestrial Networks with Double Edge Computing. In Proceedings of the 2018 IEEE International Conference on Communication Systems (ICCS), Chengdu, China, 19–21 December 2018; IEEE: Chengdu, China, 2018; pp. 450–455.
- Wang, Y.; Yang, J.; Guo, X.; Qu, Z. A Game-Theoretic Approach to Computation Offloading in Satellite Edge Computing. *IEEE Access* **2020**, *8*, 12510–12520. [[CrossRef](#)]
- Zhang, Y.; Chen, C.; Liu, L.; Lan, D.; Jiang, H.; Wan, S. Aerial Edge Computing on Orbit: A Task Offloading and Allocation Scheme. *IEEE Trans. Netw. Sci. Eng.* **2023**, *10*, 275–285. [[CrossRef](#)]
- Cheng, L.; Feng, G.; Sun, Y.; Liu, M.; Qin, S. Dynamic Computation Offloading in Satellite Edge Computing. In Proceedings of the ICC 2022-IEEE International Conference on Communications, Seoul, Republic of Korea, 16 May 2022; IEEE: Seoul, Republic of Korea, 2022; pp. 4721–4726.
- Leyva-Mayorga, I.; Martinez-Gost, M.; Moretti, M.; Pérez-Neira, A.; Vázquez, M.Á.; Popovski, P.; Soret, B. Satellite Edge Computing for Real-Time and Very-High Resolution Earth Observation. *IEEE Trans. Commun.* **2023**, *1*. [[CrossRef](#)]
- Zhu, D.; Liu, H.; Li, T.; Sun, J.; Liang, J.; Zhang, H.; Geng, L.; Liu, Y. Deep Reinforcement Learning-Based Task Offloading in Satellite-Terrestrial Edge Computing Networks. In Proceedings of the 2021 IEEE Wireless Communications and Networking Conference (WCNC), Nanjing, China, 29 March–1 April 2021; pp. 1–7.
- Li, C.; Zhang, Y.; Hao, X.; Huang, T. Jointly Optimized Request Dispatching and Service Placement for MEC in LEO Network. *China Commun.* **2020**, *17*, 199–208. [[CrossRef](#)]
- Parthasarathy, A.; Krishnamachari, B. DEFER: Distributed Edge Inference for Deep Neural Networks. In Proceedings of the 2022 14th International Conference on Communication Systems & NETworkS (COMSNETS), Bangalore, India, 4 January 2022; IEEE: Bangalore, India, 2022; pp. 749–753.
- Hu, C.; Li, B. Distributed Inference with Deep Learning Models across Heterogeneous Edge Devices. In Proceedings of the IEEE INFOCOM 2022—IEEE Conference on Computer Communications, London, UK, 2 May 2022; IEEE: London, UK, 2022; pp. 330–339.

25. Wu, G.; Wang, H.; Zhang, H.; Zhao, Y.; Yu, S.; Shen, S. Computation Offloading Method Using Stochastic Games for Software Defined Network-Based Multi-Agent Mobile Edge Computing. *IEEE Internet Things J.* **2023**, *1*. [[CrossRef](#)]
26. Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; Klimov, O. Proximal Policy Optimization Algorithms. *arXiv* **2017**, arXiv:1707.06347.

**Disclaimer/Publisher's Note:** The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.