sustainability

MDPI

Article

Edge-Distributed IoT Services Assist the Economic Sustainability of LEO Satellite Constellation Construction

Meng Zhang, Hongjian Shi and Ruhui Ma *

The School of Electronic Information and Electrical Engineering, Shanghai Jiao Tong University, Shanghai 200240, China; meng-meng@sjtu.edu.cn (M.Z.); shhjwu5@sjtu.edu.cn (H.S.)
* Correspondence: ruhuima@sjtu.edu.cn

Abstract: There are thousands or even tens of thousands of satellites in Low Earth Orbit (LEO). How to ensure the economic sustainability of LEO satellite constellation construction is an important issue currently. In this article, we envision integrating the popular and promising Internet of Things (IoT) technology with LEO satellite constellations to indirectly provide economic support for LEO satellite construction through paid IoT services. Of course, this can also bring benefits to the development of IoT. LEO Satellites can provide networks for IoT products in areas with difficult conditions, such as deserts, oceans, etc., and Satellite Edge Computing (SEC) can help to reduce the service latency of IoT. Many IoT products rely on Convolutional Neural Networks (CNNs) to provide services, and it is difficult to perform CNN inference on an edge server solely. Therefore, in this article, we use edge-distributed inference to enable the IoT services in the SEC scenario. How to perform edge-distributed inference to shorten inference time is a challenge. To shorten the inference latency of CNN, we propose a framework based on a joint partition, named EDIJP. We use a joint partition method combining data partition and model partition for distributed partition. We model the data partition as a Linear Programming (LP) problem. To address the challenge of trading off computation latency and communication latency, we designed an iterative algorithm to obtain the final partitioning result. By maintaining the original structure and parameters, our framework ensures that the inference accuracy will not be affected. We simulated the SEC environment, based on two popular CNN models, VGG16 and AlexNet, the performance of our method is varified. Compared with local inference, EdgeFlow, and CoEdge, the inference latency by using EDIJP is shorter.

Keywords: sustainability; Low Earth Orbit satellite; edge-distributed inference; Internet of Things; joint partition; linear programming

Citation: Zhang, M.; Shi, H.; Ma, R. Edge-Distributed IoT Services Assist the Economic Sustainability of LEO Satellite Constellation Construction. *Sustainability* **2024**, *16*, 1599. https://doi.org/10.3390/su16041599

Academic Editors: Ray E. Sheriff, Ayman Radwan and Weiwei Jiang

Received: 30 December 2023
Revised: 4 February 2024
Accepted: 11 February 2024
Published: 14 February 2024

1. Introduction

Nowadays, there are thousands or even tens of thousands of satellites in Low Earth Orbit (LEO) [1,2]. LEO satellites operate at altitudes of 160 to 2000 km [3], and a large number of LEO satellites can form a constellation of an LEO satellite network, providing global coverage services for terrestrial users [4,5]. Some people suggest using it as a key infrastructure for the upcoming 6G network and beyond [6–9]. However, the cost of constructing an LEO satellite constellation is enormous. The cost of an LEO satellite is approximately USD 65,000; although it is nearly 10,000 times lower than the cost of a costly "exquisite" satellite (e.g., $650,000,000) [10], LEO satellite constellations based on a large number of LEO satellites still face huge economic costs. For example, one of the state-of-the-art LEO satellite networks, SpaceX's Starlink, has over 2000 satellites in different LEO groups currently. Furthermore, the Federal Communications Commission (FCC) has approved Starlink to bring that number up to 12,000 [4]. Moreover, there are many competitors in the industry for LEO satellite constellation construction; OneWeb and O3b are another two leading enterprises in this industry [11,12]. Intense competition drives companies to create better satellite products, which will inevitably exacerbate the

already significant economic pressure.To ensure the economic sustainability of LEO satellite constellation construction, people must find an effective solution to balance the revenue and expenditure of the satellite industry.

In today's era, the Internet of Things (IoT) technology [13] has experienced rapid development and popularization with broad market prospects. By 2025, the number of connected IoT terminal equipment is expected to reach 27 billion [14]. As a promising emerging technology, IoT has been widely applied in many scenarios, such as healthcare, transportation, smart cities, smart homes, etc., and it has achieved excellent application performance [15–18]. While people enjoy the convenient services of IoT, they are also paying for IoT products. IoT offers a wide range of products, covers a wide range of fields, has a large user base, and can provide paid services. Therefore, integrating LEO satellites with IoT services can provide economic support for LEO satellite construction and achieve the economic sustainability of LEO satellite constellations.

Not only will the addition of the IoT technology provide economic support for LEO satellite construction, but the global coverage of LEO satellite constellations also contributes to the popularization of IoT technology and products. LEO satellite networks can cover areas with difficult conditions, such as deserts, oceans, etc., which traditional terrestrial networks cannot achieve [19,20]. IoT terminals in many industries such as transportation (maritime, highway, railway, aviation), maritime monitoring, and farming are located in remote areas without cellular connections [21]; the assistance of LEO satellites can help them solve communication problems. In addition, the infrastructure of terrestrial networks is susceptible to natural disasters such as earthquakes and hurricanes, which can cause communication interruptions in severe cases [11]. Therefore, LEO satellite networks help to eliminate regional restrictions on the use of IoT technology and mitigate the impact of natural disasters on communication.

IoT products can provide economic support for LEO satellite construction, and LEO satellite networks can expand the coverage of IoT products. In addition to the complementary relationship between the IoT and LEO satellite constructions mentioned above, the integration of IoT and LEO satellite constellations is also a trend. In traditional ways, due to the powerful computing and storage capabilities, cloud servers are people's preferred choice for providing IoT services. To complete IoT services, cloud servers must receive raw data from terminal devices and then return services. However, the distance between cloud servers and terminal IoT equipment is relatively long and inevitably brings high latency, which may be unbearable for some latency-sensitive IoT applications. In addition, the growth in data volume brought about by the development of the Internet has also posed new challenges to the capabilities of the communication network, which has brought more significant pressure to the cloud server form [22,23]. Recently, edge computing has provided a solution to the above dilemma. Edge computing migrates services from remote cloud to network edge closer to users. Therefore, applying the edge-computing paradigm to IoT can achieve shorter communication distances and faster services [24–26]. Edge servers are closer to users than cloud servers, so edge computing is expected to solve the high-latency challenge brought by cloud-based service provision [27,28].

Satellite Edge Computing (SEC) is proposed as a new promising computing platform. It uses LEO satellites as edge servers, and it has a lot of technical and theoretical support for its feasibility [5,10,29,30]. For example, in [19], the author proposes a multi-purpose satellite, iSat, that demonstrates the feasibility of configuring computing and storage resources on the LEO satellite. Moreover, the latency from terrestrial stations to visible LEO satellites can be reduced to 1–4 ms [30], which is quite friendly for latency-sensitive IoT applications.

Based on the above discussion, we imagine a way for the IoT to help the economic sustainability of LEO satellite constellations: that is, IoT products rely on LEO satellites to provide services. In this mode, users pay for IoT services, and IoT product providers pay for LEO satellite service providers; the economic sustainability of LEO satellite constellation construction will be guaranteed. However, there is a problem that must be considered, which is that both edge servers on the terrestrial and satellites serving as edge servers

are resource-limited, with limited computing capabilities, making it difficult to handle complex tasks. Many IoT applications use Deep Neural Networks (DNNs), especially Convolutional Neural Networks (CNNs) to provide Artificial Intelligence (AI) services, such as image classification, object detection, text processing, etc. [31–33]. Complex CNN faces significant demands for computing resources, and it generally cannot be executed by an edge server independently. Addressing the challenges, edge-distributed inference is a popular solution, and how to implement distributed inference in the edge environment to shorten the inference latency is a research hotspot.

As far as we know, this article is the first work to use edge-distributed computing in the SEC scenario. However, in the traditional edge computing scenario, there are many research studies on distributed inference, which can provide reference for our research [34–44]. To shorten the inference latency, a lot of schemes of edge-distributed inference have been proposed in recent years. In these works, using model compression techniques, such as model pruning, to achieve distributed inference disrupts the structure of the original model and can have unpredictable impacts on the accuracy of model inference [34–36]. Building a new deep model suitable for distributed inference requires retraining the model, increasing resource and time costs [44]. In contrast, the method of directly performing distributed partitioning and redeployment on the original model has been widely welcomed, as it does not change the parameters and structure of the original model nor does it require retraining [37–43].

Motivation. We have noticed that most existing works on distributed partitioning CNN models have not considered how to trade off communication latency and computing latency, but it is important to shorten the inference latency, and how to make the trade-off is tricky. In Section 2.3, we will give a more detailed introduction to the trade-off.

Contributions. To shorten the latency of edge-distributed inference, in this article, we propose EDIJP, which is an edge-distributed inference framework based on joint partitioning. Regarding our framework, joint partitioning integrates model partitioning and data partitioning. We used an iterative algorithm to trade off communication latency and computing latency. The main contributions of this article are summarized as follows:

- We propose a joint partitioning scheme that effectively reduces the latency of edge inference.
- We shorten the inference latency depending on the trade-off between communication and computing in an edge-distributed inference scene.
- To trade off the communication latency and the computing latency, we design an iterative algorithm to gradually obtain distributed partitioning and deployment results.
- We validated the method's effectiveness using the most general CNN models, VGG16 [45] and Alexnet [46], and the CloudSim [47] simulation platform.

2. Background and Motivation

In this section, we introduce two classic distributed partitioning methods for CNNs, model partitioning and data partitioning. In addition, we have explained the motivation of our idea, which is to trade off the computation latency and communication latency.

2.1. Model Partitition

A DNN model consists of many network layers. After implementing model partitioning on the original DNN, the original model can be split into multiple sub-models, representing different parts of the original model and maintaining the original weight information. As shown in Figure 1, we give a DNN sample that consists of five network layers, and the two red dashed lines have split it into three new sub-models: we named the three new sub-models sm_1, sm_2, and sm_3. The sm_1 sub-model is near the input end, and sm_3 is near the output end. We give the same input data to the original model and sm_1. For the original model, it would generate an output directly. For sm_1, it would give the output feature to sm_2 as input data. Next, the output of sm_2 will also be used by sm_3 as input data. Since the new sub-models still maintain the same weight as the original model, the output result of sm_3 will be no different from the output of the original model.

So, the model partition would not change the inference result if we retained the correct data transmission flow.

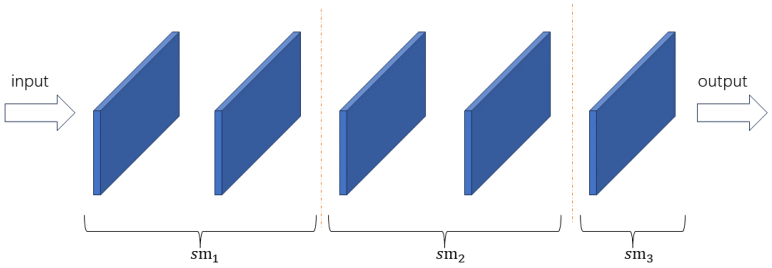
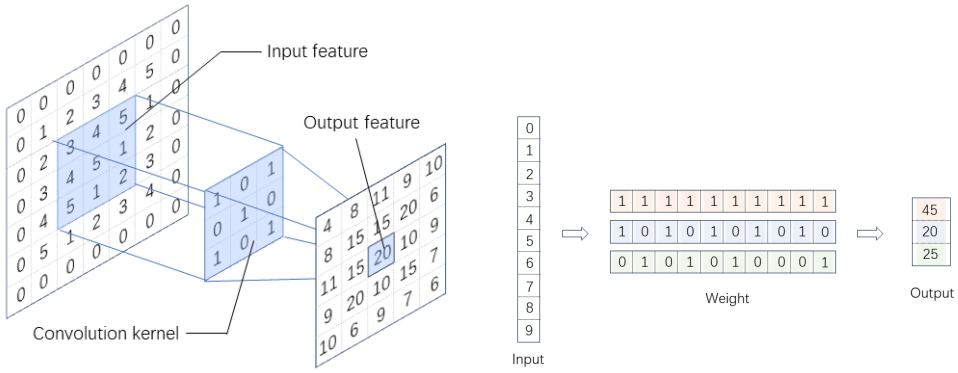


Figure 1. Model partition.

2.2. Data Partition

Based on the dependency relationship between the output features and input features of a network layer, we can divide the CNN network layers into two categories: partial dependency layers and full dependency layers. The convolution layers and pooling layers belong to partial dependency layers. We use convolution layers as an example to illustrate the partial dependency. As shown in Figure 2a, we demonstrated the dependency performance of the convolution layer. We pay attention to the blue area, input features and convolution kernels perform convolution operations to obtain output features, so the relationship between the input feature map and the output feature map is partially dependent. As shown in Figure 2b, we described the dependency behavior of full connection layers. The full connection layer in Figure 2b has parameters containing three sets of vectors, each containing ten elements. Taking a vector containing ten elements as the input, the full connection layer will dot product each set of vector parameters with the input to obtain a three-dimensional vector.



(**a**) convolution layer (**b**) full connection layer

Figure 2. Convolution layer and full connection layer.

The data partitioning of a network layer involves dividing its original input into several sub-inputs. These sub-inputs are arranged to be calculated on different devices to obtain outputs, and all outputs belonging to these devices can be seamlessly concatenated into the original output without redundancy. Each network layer has three dimensions for input or output: height (H), width (W), and number of channels (C). In this article, we divide data into network layers along the height dimension. As shown in Figure 3, we give a convolution layer data partition sample, and the number of input and output channels is simplified to 1. Without data partitioning, the operation of this convolution layer is shown in Figure 2a, which executes in a single device. In Figure 3, the convolution kernel is deployed on two devices: the original input is divided into two parts, which are used as inputs for two devices to obtain the convolution result. We record the two output results as $output_1$ and $output_2$, respectively. Additionally, we record the original output

result as $output_0$. So, the relationship between the original output and the output after data partitioning can be expressed as follows.

$$output_1 \cap output_2 = \emptyset \tag{1}$$

$$output_1 \cup output_2 = output_0 \tag{2}$$

Of course, partitioning the input data can also be transformed into partitioning the output data, as there is a correspondence between them. Taking convolution operation as an example, the kernel size, padding, and stride of convolution operation are denoted as k, p, and s, respectively. If we record the input height range as $[1, H_1]$ and the output height range as $[1, H_2]$, then the input $[in_s, in_e]$ and output $[out_s, out_e]$ of convolution operation in a device satisfy the following relationships.

$$1 \leq in_s \leq in_e \leq H_1 \tag{3}$$

$$1 \leq out_s \leq out_e \leq H_2 \tag{4}$$

$$in_s = (out_s - 1) * s - p + 1 \tag{5}$$

$$in_e = (out_e - 1) * s + k - p \tag{6}$$

In this article, our data partitioning acts on the output data, which can better ensure the relationships explained in Formulas (1) and (2) and avoid unnecessary duplicate calculations, saving computing resources.

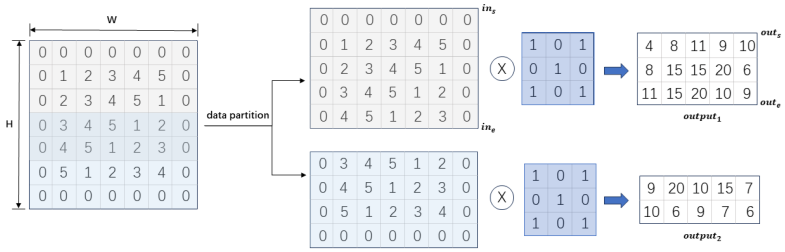


Figure 3. Data partition.

Data partitioning in partial dependency layers can achieve parallel execution, effectively reducing the execution time. However, if data partitioning is used in the full dependency layer, similar parallel computing effects cannot be achieved because even if partial output is obtained, the full dependency layer must use the complete original input. The volume range of the full dependency layer depends on the parameter volume and is independent of the input volume.

2.3. Trade-Off between Computation Latency and Communication Latency

As a type of DNN, CNN consists of many interconnection network layers. As shown in Figure 4, we give a CNN sample that includes two convolution layers, a pooling layer, and two full connection layers. From the perspective of the role played by the combination of network layers in the model, we believe that CNN consists of two parts: the feature extraction part and the classification part. The feature extraction part is responsible for extracting the main features of the input data, while the classification part generates the final inference result based on the extracted features. Generally, the feature extraction part contains convolution layers and pooling layers, while the classification part contains full connection layers. The activation layer is generally located after the three kinds of layers and is responsible for adjusting the value scale of the intermediate data using an activation function. In the description of this article, we classify the activation layer as the network layer it serves rather than treating it as a separate layer. Since the activation function acts on a single value, it does not have an impact on the structure and computation.

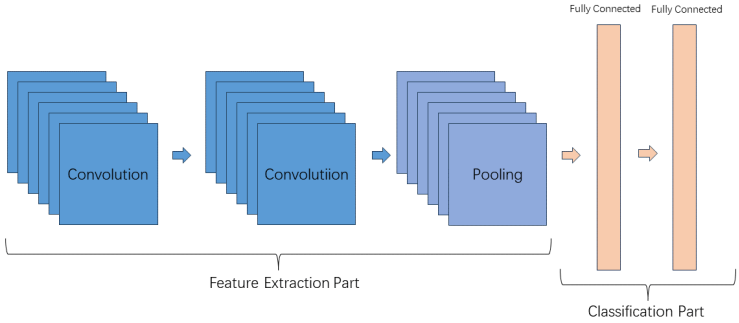


Figure 4. A CNN sample consists of two convolution layers, a pooling layer, and two full connection layers.

Using the model partition between the feature extraction part and the classification part, the original CNN model can be divided into a feature extraction sub-model and a classification sub-model. The convolution layer and pooling layer are both partial dependency layers; the feature extraction can thus use data partition further. Moreover, there is also a significant difference in the computing workload between the two sub-models: the feature extraction sub-network and the classification sub-network. We measured the computational load distribution of two classical CNNs, VGG16 and AlexNet, as shown in Table 1. The data in Table 1 show that the computing workload of CNN is mainly distributed in the feature extraction sub-network. So, the feature extraction sub-model requires parallel computing to alleviate the computing pressure of a single device.

Table 1. Computing workload comparison.

Models	CL [a] of FE [b] (MACs)	CL of CF [c] (MACs)
VGG16	15.38 G	123.65 M
AlexNet	656.91 M	58.64 M

[a] Computing Workload; [b] Feature Extraction Part; [c] Classification Part.

Only using data partition, there are two ways to partition the feature extraction sub-model: one is to partition the entire sub-model output, and the other is to partition each network layer output separately. Suppose model partitioning is added to describe the above two schemes. In that case, it can be seen as two extreme cases: performing model partitioning at the last layer of the feature extraction sub-model and after each layer of the feature extraction sub-model. The total inference latency contains computing latency and communication latency. In the first case, there will be a significant amount of computational redundancy, as obtaining the required input from the partitioned output goes through multiple network layers. In Figure 3, we can see that there are two overlapping lines in the input; if another layer is calculated upwards, the overlapping input of these two lines will become overlapping output, which means the intermediate output will no longer follow the rules of Formulas (1) and (2). The deeper the CNN, the greater the computational redundancy. In the second case, frequent data transmission may occur between devices, as this approach does not consider the mutual influence of states after layer partitioning.

A large amount of computational redundancy increases computation latency, while frequent data transmission increases communication latency. To minimize the total inference latency, it is necessary to find a trade-off between the two cases—that is, to find a trade-off between communication latency and computation latency. Overall, to distribute the execution of CNN in an SEC environment, we used a joint partitioning scheme that integrates model partitioning and data partitioning. Moreover, we designed an iterative algorithm to progressively obtain joint partitioning results.

2.4. Category

Table 2 illustrates the comparison between EDIJP and two other state-of-the-art edge-distributed inference frameworks. CoEdge [37] achieves parallel computing by partitioning data for each network layer. After the execution of each network layer, a synchronization wait is performed, and then each device transmits padding data to one adjacent device to ensure the correctness of the computation. EdgeFlow [48] also performs data partitioning on each network layer to achieve parallel computing. Intending to shorten inference latency, it determines the data partitioning scheme layer by layer based on the partitioning results of the previous network layer. Both of these frameworks do not use model partitioning, and after each network layer; there is data interaction between devices. Additionally, CoEdge also performs synchronous waiting. Although our solution has a small amount of overlapping computing, it greatly avoids frequent data transmission.

Table 2. Comparision of different frameworks.

	EDIJP	**CoEdge**	**EdgeFlow**
Data partition	✓	✓ [a]	✓
Model partition	✓	✗ [b]	✗
Synchronous waiting	✗	✓	✗
Frequent communication	✗	✓	✓
Overlapping computing	✓	✗	✗

[a] involved; [b] not involved.

3. EDIJP Framework

In this section, we introduce the overview of the EDIJP framework and the data transmission rule that is used with our framework.

3.1. EDIJP Overview

We divide the LEO satellites involved in distributed inference into two categories: requester and workers. The requester is the satellite that generates the inference request, it will give the input data, and the final inference result will also be returned to it. Workers are all the edge satellites that are considered for distributed inference; these are responsible for completing the tasks the requester assigns, including executing inference and data transmission. It is worth noting that an edge satellite can be both a requester and a worker. In once-distributed inference, there is only one requester, but there can be more than one worker.

Distributed task partitioning and deployment algorithms should be written into the resources of each satellite and integrated into a Joint Partition Module (JPM). We will provide detailed explanations of the JPM module in Section 4. As shown in Figure 5, we describe the EDIJP framework as the preparation phase and the execution phase. The partitioning and deployment decisions will be formulated in the preparation phase through the JPM. The JPM is integrated into every worker, and the worker who generates inference requests is also called a requester, which will generate deployment decisions using JPM based on the situations of computing resources and bandwidth resources. In the preparation phase, model partitioning acts on the original model, generating several sub-models that are executed sequentially. Data partitioning acts on the output data of each sub-model, allowing the inference task of the sub-models to be completed in parallel by multiple edge satellites.

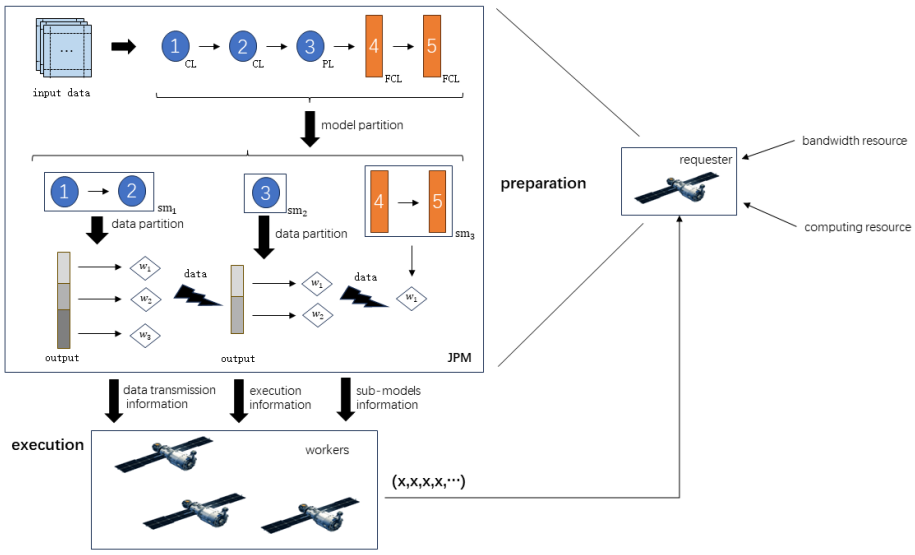


Figure 5. The EDIJP overview.

As shown in Figure 5, there are three workers in the system, names w_1, w_2, and w_3. The CNN model contains five network layers, which are two convolution layers (CLs), one pooling layer (PL), and two fully connection layers (FCLs). After the model partition, the original model is divided into three sub-models. The sub-model sm_3 that contains the two FCLs is the classification part in the original model, which will be directly deployed to a worker without any other partition operation. The other two sub-models are further processed by data partition, arranging the inference task to multiple workers. After the preparation phase, the JMP generates data transmission, execution, and sub-model information, which helps to arrange tasks for all the workers. For the convenience of description, we have separately described the results of model partitioning and data partitioning, but this does not mean that there is a clear order between model partitioning and data partitioning during JPM execution.

After the requester generates a partitioning and deployment plan, the inference tasks are distributed to the workers. At this point, the preparation phase ends and the execution phase begins. Each worker receives and executes the assigned task, and one worker may be assigned multiple tasks. After receiving all the required input data, the worker performs sub-model inference and then transmits the generated intermediate data to the output destination workers; we will describe the data transmission rule in Section 3.2.

3.2. Data Transmission between Workers

There are two workers w_i and w_j. In the original model, the execution of the sub-model in w_i requires the output of the sub-model in w_j as input. Six situations show the relationships between the input and output ranges of the two sub-models on two workers, which are contained in Figure 6. We assume that the output feature range of worker w_j is $[out_s, out_e$, the input feature range of worker w_i is $[in_s, in_e]$, and w_j needs to transmit data to w_i. In Figure 6, the two dashed lines represent the range of input data that w_i wishes to obtain. If the range indicated by the dashed lines exceeds the output range of w_j, such as Figure 6e,f, it indicates that w_j does not need to transmit data to w_i. The data range sent from w_j to w_i follows the following equations.

$$trans_s = \max\{out_s, in_s\}, \tag{7}$$

$$trans_e = \max\{out_e, in_e\}, \tag{8}$$

$[trans_s, trans_e]$ is the data transmission scale from w_j output to w_i. If $trans_e < trans_s$, there is no data transmission between the two workers.

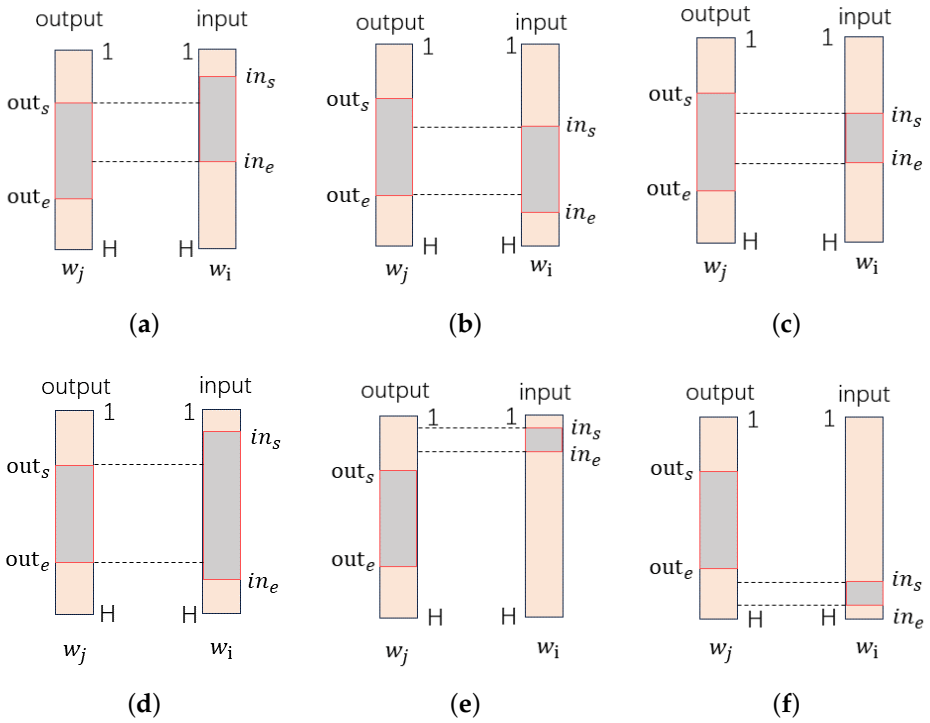


Figure 6. Data transmission situations. The sub-figures correspond to six relationships between the w_j output range and w_i input range. In sub-figures (**a**–**d**), w_j needs to transmit the output data between two dashed lines to w_i; in sub-figures (**e**,**f**), w_j need not to transmit output data to w_i.

4. Joint Workload Partition Moudle

In this section, we introduced the inherent logic of JPM completing distributed partitioning and deployment, including linear programming modeling for data partitioning problems, the initialization algorithm, and the iterative partition algorithm.

4.1. Problem Formulation

Under the paradigm of SEC, we assume that the distributed environment consists of M workers, which can be represented as $\mathbb{W} = \{w_1, w_2, \ldots, w_M\}$. We use $\mathbb{P} = \{p_1, p_2, \ldots, p_M\}$ to record the computing power of all workers, where the computing power of worker w_i is numerically represented as p_i, $i \in [1, M]$. The out-bandwidth of workers is represented by a set $\mathbb{B} = \{b_1, b_2, \ldots, b_M\}$, where b_i represents the out-bandwidth of worker w_i, $i \in [1, M]$.

We focus on a single sub-model that consists of N partial dependency layers, such as convolution layers and pooling layers. The network layers in the sub-model can be represented as $\mathbb{L} = \{l_1, l_2, \ldots, l_N\}$. The set $\mathbb{WL} = \{wl_1, wl_2, \ldots, wl_N\}$ contains the computing workload of all the network layers, where wl_n represents the computing workload of the layer l_n. We obtain the $\mathbb{WL}$ using the ptflops tool. As stipulated in Section 3.2, we use H to represent the original input feature height of each network layer, and $[trans_s, trans_e]$ can describe the data transmission situation from w_j to w_i.

We use a data partition to split the output data of this sub-model. The data partitioning is represented by $X = \{x_1, x_2, \ldots, x_M\}$; x_m is the proportion of the output data of worker w_m to the original output data of the sub-model. To ensure that X is an effective data partition, X must comply with the following constraints, Formulas (9) and (10), ensuring that the sub-model output can be concatenated into the original output by all workers.

$$0 \leq x_i \leq 1, \quad i = 1, 2, \ldots, M \tag{9}$$

$$\sum_{i=1}^{M} x_i = 1, \quad i = 1, 2, \ldots, M \tag{10}$$

We note that the inference completion time for this sub-model on the worker w_i is T_i, and T_i can be calculated using the following Equation (11).

$$T_i = T_{trans(j,i)} + T^i_{comp}, \tag{11}$$

where $T_{trans(j,i)}$ represents the transmission time of input data from the requester w_j to worker w_i, and T^i_{comp} represents the execution time at worker w_i. The inference completion time of the sub-model on the worker is equal to the time it takes for the worker to obtain the required input data plus the worker's execution time.

The transmission time from device j to device i can be computed as (12), where $D_{j,i}$ is the volume of data transmitted from w_j to w_i. In a 64-bit system, a float32 number occupies 4 Bytes. Assuming the input feature's width and number of channels are α and μ, respectively, the calculation of $D_{j,i}$ is shown in Equation (13).

$$t_{trans(j,i)} = \frac{D_{j,i}}{\mathbb{B}_{j,i}}, \tag{12}$$

$$D_{j,i} = (trans_e - trans_s) * \alpha * \mu * 4, \tag{13}$$

The computation time of worker w_i for the sub-model is denoted by t^i_{comp}; we obtain it using (14).

$$t^i_{comp} = \frac{x_i \times \sum_{j=1}^{N} wl_j}{p_i} \tag{14}$$

Our goal is to minimize the sub-model's inference latency, which is equal to the latest time for all workers to complete the sub-model. Therefore, we build an optimization objective as (15).

$$\begin{aligned} \min_{x} \quad & \lambda \\ s.t. \quad & T_i \leq \lambda, i = 1, 2, \ldots, M; \\ & Constraints \quad (9) \quad and \quad (10). \end{aligned} \tag{15}$$

The optimization problem (15) is a Linear Programming (LP) problem, and it is an NP-complete problem [49]. In this article, we use the Mosek tool to solve the linear programming problem in the Python environment.

4.2. Algorithm Design

To obtain a distributed partitioning and deployment solution, we adopted the initialization algorithm and the iterative partition algorithm. Using the initialization algorithm, we obtain the original partition for the feature extraction part and the execution worker for the classification part. Using the iterative partition algorithm, we further implement the joint partition strategy on the feature extraction part, striving for a shorter inference latency.

4.2.1. Initialization

When the original CNN is divided into multiple sub-models by the model, each sub-model is parallelly executed on multiple workers due to data partitioning. The inference completion time of a sub-model on any one worker w_i is calculated as shown in Formula (16),

$$T_i = \max_{j=1}^{M} \{T_j + T_{trans(j,i)}\} + T^i_{comp}, \tag{16}$$

where $T_{trans(j,i)}$ represents the transmission time of input data from worker w_j to worker w_i, and T^i_{comp} represents the execution time at worker w_i. Because all workers may transmit data to w_i as input data for the sub-model in w_i, we take the final time when all workers transmit data to w_i in this round as the starting time for w_i to perform sub-model inference.

As the CNN model can be divided into the feature extraction part and classification part, we consider the feature extraction sub-model, denoted by $\mathbb{EM}$, as the initial model that needs to be further partitioned, and the classification sub-model, denoted by $\mathbb{CM}$, needs a worker w_k to execute it. As shown in Algorithm 1, we first solve the optimization objective (15) for $\mathbb{EM}$ and obtain the data partition for the output feature of $\mathbb{EM}$. Then, we use a traversal method to obtain k, which is the deploy worker number for the classification sub-model.

Algorithm 1: Initialization Algorithm

Input: $\mathbb{M}$: CNN model;
WL: computing workload set;
M: the number of workers. $\mathbb{B}$: out-bandwidth set;
$\mathbb{P}$: worker computing power set $\mathbb{P} = \{p_1, p_2, \ldots p_M\}$;
Output: $\mathbb{X} = \{\dot{x}\}$: the initial partition set;
k: the number of workers that execute the classification sub-model.

Split M into feature extraction sub-model $\mathbb{EM}$ and classification sub-model $\mathbb{CM}$;
Solve the problem (15) for $\mathbb{EM}$, obtain the data partition result $\dot{x}$ and the inferred completion time of $\mathbb{EM}$ on each worker;
Initial total inference latency of $\mathbb{M}$ as $\hat{\mathrm{T}}$ = Infinity;
for *i in (1,M)* **do**
    $T_i = \max_{j=1}^{M}\{T_j + T_{trans(j,i)}\} + \frac{wl_{\mathbb{CM}}}{p_i}$;
    if T_i<$\hat{T}$ **then**
        $\hat{\mathrm{T}} = T_i$;
        k = i;
    end
end
return $\mathbb{X}, k, \hat{\mathrm{T}}$.

4.2.2. Iterative Partition

After Algorithm 1, we obtain the initial data partition $\dot{x}$ of $\mathbb{EM}$. We further implement joint partitioning on $\mathbb{EM}$ based on $\mathbb{X}$ to obtain a shorter inference latency. We adopt an iterative Algorithm 2 to gradually obtain the final partition result.

Based on $\mathbb{X}$ generated from Algorithm 1, we can give the $\mathbb{EM}$ an initial partition. We assume that the network layers number of $\mathbb{EM}$ is $\check{\mathrm{N}}$. As shown in Algorithm 2, we traverse the network layers from layer $\check{\mathrm{N}}$ to layer l_1, treating layer l_1 to the current layer, which we named as l_c, as a sub-model for data partitioning, solving the optimization objective (15), and obtaining a new partition x and a new total inference time T using Equation (16). If the new inference time T is smaller than $\hat{\mathrm{T}}$, x will be retained, and the layer number l_c will also be recorded. Otherwise, no operation is performed, and we continue to traverse forward. Perform the above steps until reaching layer l_1; then, the iteration is reached, and the final partition result $\mathbb{X}$ is obtained. We can deploy all the sub-models by $\mathbb{X}$, $\mathbb{Y}$, and k.

Algorithm 2: Iterative Partition Algorithm

Input: $\mathbb{M}$: CNN model;
$\check{N}$: the number of $\mathbb{EM}$;
WL: computation workload set;
$\mathbb{B}$: out-bandwidth set;
$\mathbb{P}$: computing power set;
$\mathbb{X}$: the initial partition set from Algorithm 1;
$\mathbb{Y}$={$\check{N}$}: the initial layer set used to contain all the end layer numbers of sub-models;
$\hat{T}$: current inference latency;
T: record for new inference latency.
Output: $\mathbb{X}$, $\mathbb{Y}$

```
for c in (Ň,1) do
    Solve problem (15) for sub-model containing layers [l_1, l_c] to obtain x;
    consider x, compute new inference latency T using Equation (16);
    if T<T̂ then
        T̂ = T
        x insert to 𝕏;
        c insert to 𝕐;
    end
end
return 𝕏, k, T̂.
```

4.2.3. Complexity

The LP problem is an NP-complete problem [49]; there are a lot of algorithms to solve the LP problem [50]. In the worse case of solving the LP problem, the time complexity of using Input Sparsity Time algorithms [51] is $O(n^{2.5}L)$, and that value would be $O((n+d)^{1.5}nL)$ using Vaidya's 89 algorithm [52], where d is the number of constraints, n is the number of variables, and L is the number of bits. In the 64-bit operating system, using the float32 form, the value of L is 32. For our optimization objective (15), the number of variables is the workers' number M, and the number of constraints is 2+M. So, we can view the time complexity of solving the LP problem (15) as $O(M^{2.5})$ approximately. Generally, the time complexity of solving the maximum problem is $O(logn)$, where n is the scale of the problem. The time complexities of the other operations in our algorithms, such as split, assignment, and insert, are all O(1). Therefore, we can calculate the time complexity of the two algorithms in this article, the time complexity of Algorithm 1 is $O(M^{2.5}+MlogM)$, and the time complexity of Algorithm 2 is O($\check{N}(logM+M^{2.5})$), where $\check{N}$ is the number of network layers in the feature extraction part, and M is the number of workers. The time complexity values of Algorithm 1 and Algorithm 2 can be simplified to $O(M^{2.5})$ and O($\check{N}M^{2.5}$), respectively. Overall, the time complexity of the method proposed in this article is O($\check{N}M^{2.5}$).

5. Evaluation

In this section, we explain the experimental setup and the experimental results to demonstrate the effectiveness of our EDIJP framework.

5.1. Experimental Setup

5.1.1. Prototype

In this experiment, we simulated the SEC environment using CloudSim [47], which is an open-source simulation platform for the cloud computing environment. We have made some modifications to CloudSim from the source code to make it suitable for the research environment of this article.

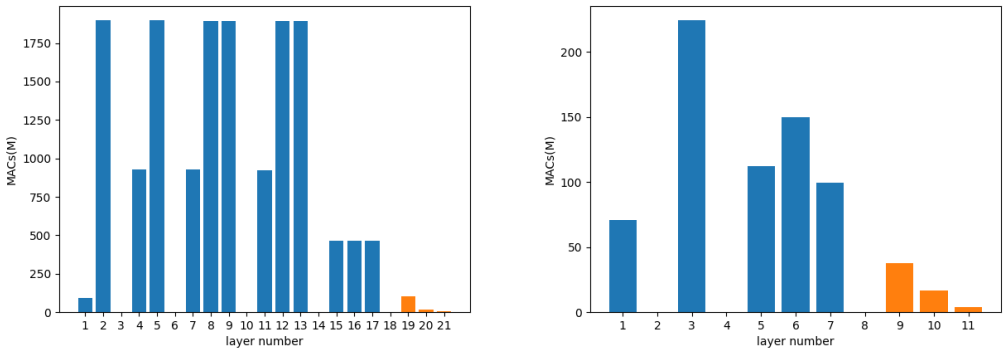
We consider an SEC network with 60 LEO satellites, and it can provide 5 LEO satellites as workers for edge-distributed inference one time. Given the limited size, each LEO satellite is attached with lightweight accelerators, such as NVIDIA Jeston Orin NX [21,53]. We obtained the computing power of an NVIDIA Jeston Orin NX using the method mentioned in [37]. Considering the inconsistent computing power of edge servers, we use one-fifth of the obtained value as the computing power of a CPU core in edge servers and control the CPU core number of each worker to an integer between 2 and 10 randomly. We control the out-bandwidth for each worker between 1000 and 2000 MB/S [54]. We selected five out-bandwidth values, 1000 MB/S, 1250 MB/S, 1500 MB/S, 1750 MB/S, and 2000 MB/S, and randomly assigned them to the five workers. Therefore, by fixing a worker as the requester, it can avoid randomness and make the environment configuration more realistic. Table 3 shows the parameters of our simulation environment.

Table 3. Environment parameters.

Workers	Cores Number	Out-Bandwidth (MB/S)
worker 1	3	1500
worker 2	9	1750
worker 3	7	1250
worker 4	5	2000
worker 5	7	1000

5.1.2. Methodology

Based on the initialization Algorithm 1 and iteration Algorithm 2, we obtain the distributed deployment plan using Python 3.6 and then obtain simulation results on the CloudSim platform. We adopt two pre-trained PyTorch models from the ImageNet database, VGG16 and Alexnet, as the CNN models of our experiments. The ptflops tool is used to obtain their MACs, which is not only for the whole model but also for each network layer. The computation workload distribution is shown in Figure 7. We set the workload as the image classification task on one ImageNet [55] image. The average inference latency, computation intensity, and transmission intensity of one hundred runs are taken as the results.



(**a**) VGG16 (**b**) AlexNet

Figure 7. This is the computing workload distribution of VGG16 and AlexNet. We use two colors to differentiate the network layers that belong to the feature extraction part and the classification part, blue and yellow. The computing workload of the feature extraction part is significantly higher than that of the classification part.

5.1.3. Baseline Methods

In the evaluation, we use two typical state-of-the-art frameworks and a local approach as our baselines. (1) CoEdge [37]. CoEdge achieves parallel computing by partitioning input data for each network layer. After the execution of each network layer, a synchronization

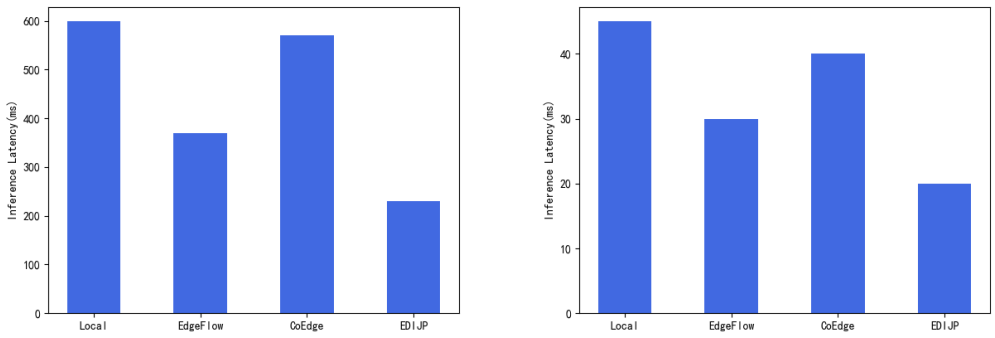
wait is performed, and then each device transmits padding data to one adjacent device to ensure the correctness of the computation. (2) EdgeFlow [48]. EdgeFlow also performs data partitioning on each network layer to achieve parallel computing. To shorten the inference latency, it determines the data partitioning scheme layer by layer based on the partitioning results of the previous network layer. (3) Local. The local approach executes the whole inference task at the requester.

5.2. Evaluation Result

5.2.1. Inference Latency Comparison

Our research goal is to shorten inference latency using edge-distributed inference, so inference latency is the most reflective indicator of our method's performance. Based on the image classification service, we obtained inference latencies of VGG16 and AlexNet using our method and three benchmark methods.

As shown in Figure 8, compared to the other three benchmark algorithms, we can see that our method EDIJP can obtain a shorter inference latency. For the execution of AlexNet, compared with the local approach, the inference latency of EDIJP has decreased by 56%, and compared to EdgeFlow [48] and CoEdge [37], the inference latency of EDIJP has decreased by 33% and 50%, respectively.



(**a**) VGG16 (**b**) AlexNet

Figure 8. The comparison result of inference latency.

5.2.2. Workload Deploy and Data Transmission

Due to the heterogeneity of satellite hardware resource allocation, different workers' computing power and out-bandwidth vary. Generally speaking, workers with stronger computing power should bear more of a computing workload. In contrast, workers with higher out-bandwidth should transmit greater data volumes to synchronize inference processes among workers and minimize the idle time of each worker as much as possible. To observe and analyze the relationship between computing workload deployment and workers' computing powers, as well as the relationship between data transmission volume and worker out-bandwidth, we separately calculated the computing workload distribution and data transmission situation for VGG16 and AlexNet using our method, EdgeFlow, and CoEdge.

When analyzing the relationship between computing workload distribution and worker computing power, we normalize the environmental configuration; the detailed calculation process is shown as (18). It is worth noting that when calculating the workload, we use the input height H_{in} of the network layer; when calculating the data transmission volume, we use the output height H_{out} of the network layer.

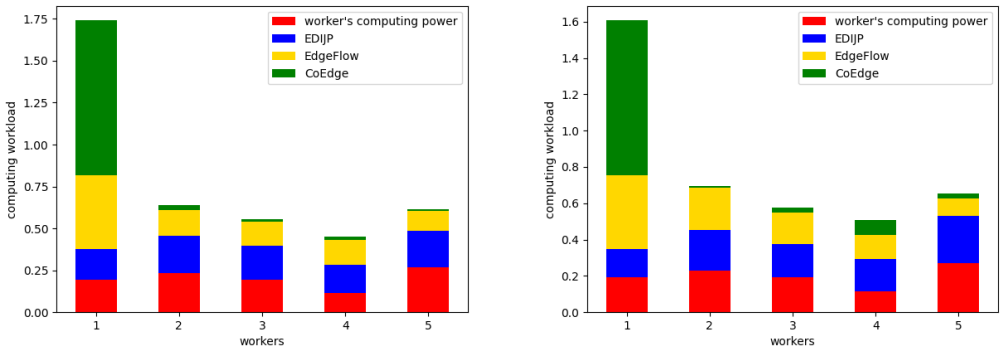
$$\zeta_i = \frac{\sum_{j=1}^{N} \frac{h_i^j}{H_{in}}}{N}, \tag{17}$$

$$\bar{\zeta}_i = \frac{\zeta_i}{\sum_{j=1}^{M} \zeta_j}, \tag{18}$$

In (17), ζ_i is the average proportion of computing workload allocated to worker w_i at each layer, while h_i^j is the input height of w_i in layer l_j for executing a sub-model. According to $\zeta_i, i = 1, \ldots, M$, a vector $\zeta = \{\zeta_1, \ldots, \zeta_M\}$ containing the average computing workload proportion of each worker per layer can be obtained. We use (18) to normalize the vector ζ. Similarly, we can standardize the computing power of each worker using (19); $\bar{\eta}_i^p$ is the standardized worker's computing power.

$$\bar{\eta}_i^p = \frac{p_i}{\sum_{i=1}^{M} p_i}, \tag{19}$$

As shown in Figure 9, it is evident that the computation workload distribution obtained by our method is more in line with the computing power distribution of the worker.



(**a**) VGG16 (**b**) AlexNet

Figure 9. The computing workload distribution and workers' computing power distribution.

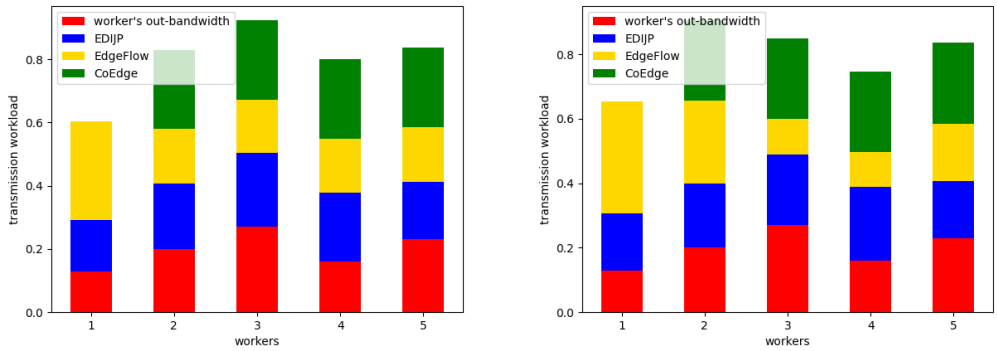
Like our logic of analyzing computation workload distribution and worker computing power, we analyze the relationship between data transmission volume and worker out-bandwidth. In (20), ξ_i is the average ratio of the volume of data output from the worker w_i to other workers at each layer to the total output data volume. h_i^j is the height at which the worker outputs to other workers in layer l_j, as the data transmission from the worker to itself does not involve bandwidth and can be ignored. (21) and (22) are standardized operations, just like standardizing the distribution of computation workload. $\bar{\xi}_i$ (21) is the standardized value of the worker's output data volume, and $\bar{\vartheta}_i^b$ in (22) is the standardized value of the worker's out-bandwidth.

$$\xi_i = \frac{\sum_{j=1}^{N} \frac{h_i^j}{H_{out}}}{N}, \tag{20}$$

$$\bar{\xi}_i = \frac{\xi_i}{\sum_{j=1}^{M} \xi_j}, \tag{21}$$

$$\bar{\vartheta}_i^b = \frac{b_i}{\sum_{i=1}^{M} b_i}, \tag{22}$$

As shown in Figure 10, we have explained the relationship between the output data volume of the worker and its out-bandwidth. Based on our method, we can see that the volume of data output generated by workers is most closely aligned with its out-bandwidth.

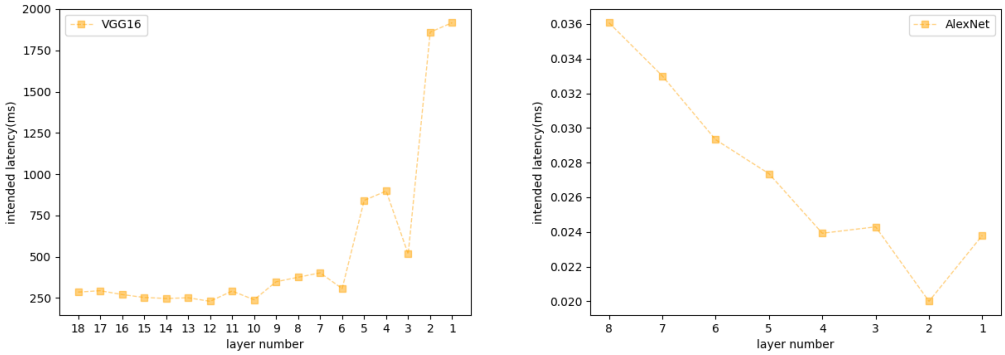


(**a**) vgg16 (**b**) alexnet

Figure 10. The data transmission distribution and the workers' out-bandwidth distribution.

5.2.3. Latency Iteration

We use an iterative algorithm to gradually obtain an effective partitioning and deployment strategy to shorten a CNN model's inference latency. To study the decline curve of inference latency during the iteration process, we record the inference latency values during one iteration process, as shown in Figure 11. From Figure 11a, it can be seen that for VGG16, compared to the network layers located at the front, our method believes that the network layers located near the end are more suitable as cutting points for model partitioning. From Figure 11b, we can see that our method does not have the same preference for cutting points for AlexNet as for VGG16.



(**a**) VGG16 (**b**) AlexNet

Figure 11. The latency iterations of VGG16 and AlexNet.

6. Related Work

In this section, we conducted a theoretical analysis of the existing research on accelerating the DNN inference of edge-distributed inference. To the best of our knowledge, there is no research on edge-distributed inference in the SEC, so we analyzed the related works in terrestrial edge computing.

To accelerate the inference of DNN, there is a work to achieve the distributed DNN inference using pruning [44]. In [44], the authors use a class-aware pruning scheme [56] to trim the original DNN so that the new model small DNN (SNN) can only cover a portion of the original output categories. Using this principle, the author obtains several SNNs for distinguishing output categories based on the original DNN, which are deployed on multiple edge servers and collaborate to obtain inference results that can fully cover the category range of the original output. However, changing the parameters and structure of the original model can easily have unpredictable impacts on the inference accuracy.

There is also a method that proposes a loosely coupled CNN structure to fundamentally solve the problem of distributed deployment [42]. In [42], the authors designed a new loosely coupled structure (LCS) to adapt the distributed CNN inference. However, this approach faces the need for retraining new models, requiring different parameters for different purposes and corresponding to different datasets, which is a huge consumption of computing resources and time.

The method that distributed partition DNNs effectively avoids the challenges brought by the above two methods. Distributed partitioning and the deployment of DNNs can take place without changing the structure and parameters of the original model, and it does not need to retrain the model [38,41,57–59]. In [41], the authors deploy workload in a distributed manner based on network layer types. They use an input partition for convolution layers and a weight partition for full connection layers. BiasedOne-Dimensional Partition (BODP), Modified Spectral Co-Clustering (MSCC), and Modified Spectral Co-Clustering (MSCC) are proposed as adjuncts. In [38], the authors propose an Fused Tile Partitioning (FTP) method to parallelize the convolution operation; it can divide the feature maps of each layer into small tiles in a grid fashion. Both [57,58] rely on Deep Reinforcement Learning (DRL). In [57], the authors search for an optimal partition location for each layer volume using the Layer Configuration-based Partition Scheme Search (LC-PSS). For a layer volume splitter, they model the split process as a Markov Decision Process (MDP) and use DRL to make optimal split decisions. In [58], the authors use DRL techniques to assist in task allocation in edge-distributed inference. Modeling the partition process as MDP, DRL agents use inference latency and layer configuration as the rewards and states, and then the optimal segmentation decisions for each layer volume are made one by one. In [59], the authors use a host edge server to configure multiple secondary edge servers, and the overlapping zone of sub-tasks on the secondary edge servers is executed on the host edge server.

Considering that these distributed partitioning studies did not consider the trade-off between communication latency and computation latency, which is important for shortening inference latency and trick, we propose the EDIJP framework. Our proposed framework is based on joint partitioning for distributed deployment. We model the data partition problem as an LP problem and design an iterative algorithm to achieve the trade-off of communication latency and computation latency so that the inference latency can be shortened as much as possible.

7. Conclusions

To ensure the economic sustainability of the LEO satellite constellation building, we propose to integrate the IoT and LEO satellites, using the payment services of IoT to indirectly provide economic support for the LEO satellite construction building. Many IoT products require CNN to provide services. To solve the distributed deployment problem of CNNs in SEC and shorten the inference latency, we propose the EDIJP framework. We propose a joint partition approach that combines model partition and data partition. We model the data partition problem as an LP problem and propose an iterative algorithm to trade off communication latency and computation latency, achieving the goal of shortening inference latency. To comprehensively analyze and observe the performance of EDIJP, we designed a series of comparative experiments. However, our method still has limitations that cannot be extended to all types of DNNs, which is the improvement direction we need to focus on in the next step. In addition, we should further explore the characteristics of SEC and push the development of IoT in satellite constellations based on edge-distributed inference.

Author Contributions: Conceptualization, M.Z., H.S. and R.M.; methodology, M.Z. and H.S.; validation, M.Z.; investigation, M.Z.; resources, R.M.; writing—original draft preparation, M.Z.; writing—review and editing, M.Z., H.S. and R.M.; supervision, R.M.; project administration, H.S. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Data are contained within the article.

Acknowledgments: This work was supported in part by the Shanghai Key Laboratory of Scalable Computing and Systems.

Conflicts of Interest: The authors declare no conflicts of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript; or in the decision to publish the results.

Abbreviations

The following abbreviation is used in this manuscript:

LEO	Low Earth Orbit
SEC	Satellite Edge Computing
IoT	Internet of Things
LP	Linear Program
AI	Artificial Intelligence
CNN	Convolutional Neural Network
JPM	Joint Partition Module
DNN	Deep Neural Network
SNN	Small DNN
DRL	Deep Reinforcement Learning
MDP	Markov Decision Process

References

1. McDowell, J.C. The Low Earth Orbit Satellite Population and Impacts of the SpaceX Starlink Constellation. *Astrophys. J. Lett.* **2020**, *892*, L36. [CrossRef]
2. Hainaut, O.R.; Williams, A.P. Impact of satellite constellations on astronomical observations with ESO telescopes in the visible and infrared domains. *Astron. Astrophys.* **2020**, *636*, A121. [CrossRef]
3. Lu, Y.; Shao, Q.; Yue, H.; Yang, F. A Review of the Space Environment Effects on Spacecraft in Different Orbits. *IEEE Access* **2019**, *7*, 93473–93488. [CrossRef]
4. Ma, S.; Chou, Y.C.; Zhao, H.; Chen, L.; Ma, X.; Liu, J. Network Characteristics of LEO Satellite Constellations: A Starlink-Based Measurement from End Users. In Proceedings of the IEEE INFOCOM 2023—IEEE Conference on Computer Communications, New York City, NY, USA, 17–20 May 2023; pp. 1–10. [CrossRef]
5. Cassarà, P.; Gotta, A.; Marchese, M.; Patrone, F. Orbital Edge Offloading on Mega-LEO Satellite Constellations for Equal Access to Computing. *IEEE Commun. Mag.* **2022**, *60*, 32–36. [CrossRef]
6. Lin, X.; Cioni, S.; Charbit, G.; Chuberre, N.; Hellsten, S.; Boutillon, J. On the Path to 6G: Embracing the Next Wave of Low Earth Orbit Satellite Access. *IEEE Commun. Mag.* **2021**, *59*, 36–42. [CrossRef]
7. Wei, X.; Tang, C.; Fan, J.; Subramaniam, S. Joint Optimization of Energy Consumption and Delay in Cloud-to-Thing Continuum. *IEEE Internet Things J.* **2019**, *6*, 2325–2337. [CrossRef]
8. Yue, P.; An, J.; Zhang, J.; Ye, J.; Pan, G.; Wang, S.; Xiao, P.; Hanzo, L. Low Earth Orbit Satellite Security and Reliability: Issues, Solutions, and the Road Ahead. *IEEE Commun. Surv. Tutor.* **2023**, *25*, 1604–1652. [CrossRef]
9. Lin, Z.; Niu, H.; An, K.; Wang, Y.; Zheng, G.; Chatzinotas, S.; Hu, Y. Refracting RIS-Aided Hybrid Satellite-Terrestrial Relay Networks: Joint Beamforming Design and Optimization. *IEEE Trans. Aerosp. Electron. Syst.* **2022**, *58*, 3717–3724. [CrossRef]
10. Denby, B.; Lucia, B. *Orbital Edge Computing: Nanosatellite Constellations as a New Class of Computer System*; ASPLOS '20; Association for Computing Machinery: New York, NY, USA, 2020; pp. 939–954. [CrossRef]
11. Deng, P.; Gong, X.; Que, X. A bandwidth-aware service migration method in LEO satellite edge computing network. *Comput. Commun.* **2023**, *200*, 104–112. [CrossRef]
12. Zhang, Z.; Zhang, W.; Tseng, F. Satellite Mobile Edge Computing: Improving QoS of High-Speed Satellite-Terrestrial Networks Using Edge Computing Techniques. *IEEE Netw.* **2018**, *33*, 70–76. [CrossRef]
13. Jing, B.; Xue, H. IoT Fog Computing Optimization Method Based on Improved Convolutional Neural Network. *IEEE Access* **2024**, *12*, 2398–2408. [CrossRef]
14. Qu, Z.; Zhang, G.; Cao, H.; Xie, J. LEO Satellite Constellation for Internet of Things. *IEEE Access* **2017**, *5*, 18391–18401. [CrossRef]

15. Padmaja, B.; Narasimha Rao, P.V.; Madhu Bala, M.; Rao Patro, E.K. A Novel Design of Autonomous Cars using IoT and Visual Features. In Proceedings of the 2018 2nd International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC)I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC), Palladam, India, 30–31 August 2018; pp. 18–21. [CrossRef]
16. Hassan, R.; Sagar, A.K.; Banda, L. Future Internet of Things: A Framework for Next Generation Smart Cities. In Proceedings of the 2021 IEEE 6th International Conference on Computing, Communication and Automation (ICCCA), Arad, Romania, 17–19 December 2021; pp. 106–112. [CrossRef]
17. Derawi, M.O.; Dalveren, Y.; Cheikh, F.A. Internet-of-Things-Based Smart Transportation Systems for Safer Roads. In Proceedings of the 6th IEEE World Forum on Internet of Things, WF-IoT 2020, New Orleans, LA, USA, 2–16 June 2020; pp. 1–4. [CrossRef]
18. Lin, Z.; Niu, H.; An, K.; Hu, Y.; Li, D.; Wang, J.; Al-Dhahir, N. Pain Without Gain: Destructive Beamforming From a Malicious RIS Perspective in IoT Networks. *IEEE Internet Things J.* **2023**, 1. [CrossRef]
19. Wang, Y.; Yang, J.; Guo, X.; Qu, Z. Satellite Edge Computing for the Internet of Things in Aerospace. *Sensors* **2019**, *19*, 4375. [CrossRef] [PubMed]
20. Dong, Q.; Xu, X.; Han, S.; Liu, R.; Zhang, X. DDPG-Based Task Offloading in Satellite-Terrestrial Collaborative Edge Computing Networks. In Proceedings of the IEEE International Conference on Communications, ICC 2023–Workshops, Rome, Italy, 28 May–1 June 2023; pp. 1541–1546. [CrossRef]
21. Xu, Z.; Xu, G.; Wang, H.; Liang, W.; Xia, Q.; Wang, S. Enabling Streaming Analytics in Satellite Edge Computing via Timely Evaluation of Big Data Queries. *IEEE Trans. Parallel Distrib. Syst.* **2024**, *35*, 105–122. [CrossRef]
22. Yang, Y.; Wei, H. Edge-IoT Computing and Networking Resource Allocation for Decomposable Deep Learning Inference. *IEEE Internet Things J.* **2023**, *10*, 5178–5193. [CrossRef]
23. Zhou, Z.; Chen, X.; Li, E.; Zeng, L.; Luo, K.; Zhang, J. Edge Intelligence: Paving the Last Mile of Artificial Intelligence with Edge Computing. *Proc. IEEE* **2019**, *107*, 1738–1762. [CrossRef]
24. Goudarzi, M.; Palaniswami, M.; Buyya, R. Scheduling IoT Applications in Edge and Fog Computing Environments: A Taxonomy and Future Directions. *ACM Comput. Surv.* **2023**, *55*, 152:1–152:41. [CrossRef]
25. Rajagopal, S.M.; Supriya, M.; Buyya, R. FedSDM: Federated learning based smart decision making module for ECG data in IoT integrated Edge-Fog-Cloud computing environments. *Internet Things* **2023**, *22*, 100784. [CrossRef]
26. Jung, S.; Jeong, S.; Kang, J.; Kang, J. Marine IoT Systems with Space-Air-Sea Integrated Networks: Hybrid LEO and UAV Edge Computing. *IEEE Internet Things J.* **2023**, *10*, 20498–20510. [CrossRef]
27. Guo, H.; Liu, J. UAV-Enhanced Intelligent Offloading for Internet of Things at the Edge. *IEEE Trans. Ind. Inform.* **2020**, *16*, 2737–2746. [CrossRef]
28. Kan, T.; Chiang, Y.; Wei, H. Task offloading and resource allocation in mobile-edge computing system. In Proceedings of the 27th Wireless and Optical Communication Conference, WOCC 2018, Hualien, Taiwan, 30 April–1 May 2018; pp. 1–4. [CrossRef]
29. Bhattacherjee, D.; Kassing, S.; Licciardello, M.; Singla, A. In-orbit Computing: An Outlandish thought Experiment? In Proceedings of the HotNets '20: The 19th ACM Workshop on Hot Topics in Networks, Virtual Event, USA, 4–6 November 2020; pp. 197–204. [CrossRef]
30. Abreha, H.G.; Chougrani, H.; Maity, I.; Nguyen, V.; Chatzinotas, S.; Politis, C. Fairness-Aware Dynamic VNF Mapping and Scheduling in SDN/NFV-Enabled Satellite Edge Networks. In Proceedings of the IEEE International Conference on Communications, ICC 2023, Rome, Italy, 28 May–1 June 2023; pp. 4892–4898. [CrossRef]
31. Massimi, F.; Ferrara, P.; Benedetto, F. Deep Learning Methods for Space Situational Awareness in Mega-Constellations Satellite-Based Internet of Things Networks. *Sensors* **2023**, *23*, 124. [CrossRef]
32. Swaminathan, B.; Palani, S.; Vairavasundaram, S.; Kotecha, K.; Kumar, V. IoT-Driven Artificial Intelligence Technique for Fertilizer Recommendation Model. *IEEE Consum. Electron. Mag.* **2023**, *12*, 109–117. [CrossRef]
33. Tripathy, P.K.; Tripathy, A.K.; Agarwal, A.; Mohanty, S.P. MyGreen: An IoT-Enabled Smart Greenhouse for Sustainable Agriculture. *IEEE Consum. Electron. Mag.* **2021**, *10*, 57–62. [CrossRef]
34. He, Y.; Zhang, X.; Sun, J. Channel Pruning for Accelerating Very Deep Neural Networks. In Proceedings of the IEEE International Conference on Computer Vision, ICCV 2017, Venice, Italy, 22–29 October 2017; pp. 1398–1406. [CrossRef]
35. Liang, J.; Zhang, L.; Bu, C.; Cheng, D.; Wu, H.; Song, A. An automatic network structure search via channel pruning for accelerating human activity inference on mobile devices. *Expert Syst. Appl.* **2024**, *238*, 122180. [CrossRef]
36. Yu, F.; Cui, L.; Wang, P.; Han, C.; Huang, R.; Huang, X. EasiEdge: A Novel Global Deep Neural Networks Pruning Method for Efficient Edge Computing. *IEEE Internet Things J.* **2021**, *8*, 1259–1271. [CrossRef]
37. Zeng, L.; Chen, X.; Zhou, Z.; Yang, L.; Zhang, J. CoEdge: Cooperative DNN Inference With Adaptive Workload Partitioning Over Heterogeneous Edge Devices. *IEEE/ACM Trans. Netw.* **2021**, *29*, 595–608. [CrossRef]
38. Zhao, Z.; Barijough, K.M.; Gerstlauer, A. DeepThings: Distributed Adaptive Deep Learning Inference on Resource-Constrained IoT Edge Clusters. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* **2018**, *37*, 2348–2359. [CrossRef]
39. Hadidi, R.; Cao, J.; Woodward, M.; Ryoo, M.S.; Kim, H. Distributed Perception by Collaborative Robots. *IEEE Robot. Autom. Lett.* **2018**, *3*, 3709–3716. [CrossRef]
40. Zhou, L.; Samavatian, M.H.; Bacha, A.; Majumdar, S.; Teodorescu, R. Adaptive parallel execution of deep neural networks on heterogeneous edge devices. In Proceedings of the 4th ACM/IEEE Symposium on Edge Computing, SEC 2019, Arlington, VA, USA, 7–9 November 2019; pp. 195–208. [CrossRef]

41. Mao, J.; Chen, X.; Nixon, K.W.; Krieger, C.D.; Chen, Y. MoDNN: Local distributed mobile computing system for Deep Neural Network. In Proceedings of the Design, Automation & Test in Europe Conference & Exhibition, DATE 2017, Lausanne, Switzerland, 27–31 March 2017; pp. 1396–1401. [CrossRef]
42. Du, J.; Du, Y.; Huang, D.; Lu, Y.; Liao, X. Enhancing Distributed In-Situ CNN Inference in the Internet of Things. *IEEE Internet Things J.* **2022**, *9*, 15511–15524. [CrossRef]
43. Hadidi, R.; Cao, J.; Ryoo, M.S.; Kim, H. Toward Collaborative Inferencing of Deep Neural Networks on Internet-of-Things Devices. *IEEE Internet Things J.* **2020**, *7*, 4950–4960. [CrossRef]
44. Hemmat, M.; Davoodi, A.; Hu, Y.H. Edge^n AI: Distributed Inference with Local Edge Devices and Minimal Latency. In Proceedings of the 27th Asia and South Pacific Design Automation Conference, ASP-DAC 2022, Taipei, Taiwan, 17–20 January 2022; pp. 544–549. [CrossRef]
45. Simonyan, K.; Zisserman, A. Very Deep Convolutional Networks for Large-Scale Image Recognition. In Proceedings of the 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, 7–9 May 2015; Conference Track Proceedings.
46. Krizhevsky, A.; Sutskever, I.; Hinton, G.E. ImageNet classification with deep convolutional neural networks. *Commun. ACM* **2017**, *60*, 84–90. [CrossRef]
47. Calheiros, R.N.; Ranjan, R.; Rose, C.A.F.D.; Buyya, R. CloudSim: A Novel Framework for Modeling and Simulation of Cloud Computing Infrastructures and Services. *arXiv* **2009**, arXiv:0903.2525.
48. 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, 2–5 May 2022; pp. 330–339. [CrossRef]
49. Burks, T.M.; Sakallah, K.A. Min-max linear programming and the timing analysis of digital circuits. In Proceedings of the Proceedings of the 1993 IEEE/ACM International Conference on Computer-Aided Design, Santa Clara, CA, USA, 7–11 November 1993; pp. 152–155. [CrossRef]
50. Wikipedia. Linear Programming. Available online: https://en.wikipedia.org/wiki/Linear_programming/ (accessed on 1 February 2024).
51. Lee, Y.T.; Sidford, A. Efficient Inverse Maintenance and Faster Algorithms for Linear Programming. In Proceedings of the 2015 IEEE 56th Annual Symposium on Foundations of Computer Science, Berkeley, CA, USA, 17–20 October 2015; pp. 230–249. [CrossRef]
52. Vaidya, P. Speeding-up linear programming using fast matrix multiplication. In Proceedings of the 30th Annual Symposium on Foundations of Computer Science, Research Triangle Park, NC, USA, 30 October 1989; pp. 332–337. [CrossRef]
53. NVIDIA. NVIDIA Jeston Orin NX. 2023. Available online: https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-orin/ (accessed on 1 February 2024).
54. 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 IEEE International Conference on Communication Systems, ICCS 2018, Chengdu, China, 19–21 December 2018; pp. 450–455. [CrossRef]
55. Deng, J.; Dong, W.; Socher, R.; Li, L.; Li, K.; Fei-Fei, L. ImageNet: A large-scale hierarchical image database. In Proceedings of the 2009 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2009), Miami, FL, USA, 20–25 June 2009; pp. 248–255. [CrossRef]
56. Hemmat, M.; Miguel, J.S.; Davoodi, A. CAP'NN: Class-Aware Personalized Neural Network Inference. In Proceedings of the 57th ACM/IEEE Design Automation Conference, DAC 2020, San Francisco, CA, USA, 20–24 July 2020; pp. 1–6. [CrossRef]
57. Zhang, S.; Zhang, S.; Qian, Z.; Wu, J.; Jin, Y.; Lu, S. DeepSlicing: Collaborative and Adaptive CNN Inference with Low Latency. *IEEE Trans. Parallel Distrib. Syst.* **2021**, *32*, 2175–2187. [CrossRef]
58. Hou, X.; Guan, Y.; Han, T.; Zhang, N. DistrEdge: Speeding up Convolutional Neural Network Inference on Distributed Edge Devices. In Proceedings of the 2022 IEEE International Parallel and Distributed Processing Symposium, IPDPS 2022, Lyon, France, 30 May–3 June 2022; pp. 1097–1107. [CrossRef]
59. Li, N.; Iosifidis, A.; Zhang, Q. Distributed Deep Learning Inference Acceleration using Seamless Collaboration in Edge Computing. In Proceedings of the IEEE International Conference on Communications, ICC 2022, Seoul, Republic of Korea, 16–20 May 2022; pp. 3667–3672. [CrossRef]