LEOEdge: A Satellite-Ground Cooperation Platform for the AI Inference in Large LEO Constellation

Su Yao, Yiying Lin, Mu Wang, Ke Xu, *Fellow, IEEE*, Mingwei Xu, *Senior Member, IEEE*, Changqiao Xu, *Senior Member, IEEE*, and Hongke Zhang, *Fellow, IEEE*

***Abstract*—With the rapid growth of low earth orbit (LEO) satellites, enabling LEO AI inference becomes a fast-increasing trend. However, due to resource heterogeneity, scheduling complexity, and fast movement, how to decide the place of executing each AI inference task is nontrivial in LEO systems. In this paper, we propose LEOEdge, an edge-assisted AI inference system for LEO satellites. We first introduce the adaptive modeling technologies that automatically generate the model for each satellite according to its computation resources. We then propose a layered scheduling optimization scheme to schedule the AI inference task in a distributed manner. LEOEdge also designs a seamless data transmission scheme to avoid transmission failure due to the LEO satellite movement. We conduct a series of simulation tests to validate the performance of the proposed LEOEdge, in terms of the neural network searching efficiency, average time execution latency, and delivery latency.**

***Index Terms*—Cloud-edge-device computing, task offloading, multi-layer optimization, LEO satellite.**

I. Introduction

THE recent rapid advances in low earth orbit (LEO) satellite [1] technologies have shown great success in launching Global Internet access services. For example, Starlink [2] with more than 4000 LEO satellites provides Internet services to more than 2.3 million users within the infrastructure-less area. The purpose of better supporting various Internet applications with artificial intelligence (AI) use cases, computer vision (CV) [3], natural language processing (NLP), and recommendations, raises a growing demand trend on enabling computing on LEO platforms. However, the onboard resources of LEO satellites can be limited since most precious resources are consumed by data forwarding and transmission. To overcome the resource shortage in LEO constellations, a promising way is to build a space-ground collaborative computation platform [4] that allows the LEO satellites to offload parts of computation tasks to high-performance ground computation centers. As shown in Fig. 1, LEO satellites collect the images and text information, running the AI inference model during periods of light workload. When in heavy workload, they forward the data to the ground computing servers. The servers run the AI model and output the inference results. The results are then delivered to the end users via LEO constellations.

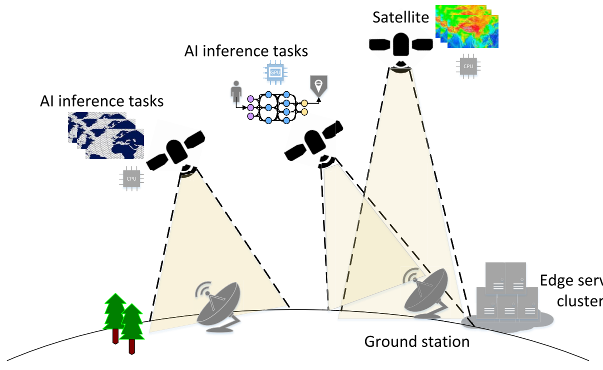


Fig. 1. LEOEdge operational architecture.

Efforts have been made to enable highly efficient task offloading within the space-ground collaboration paradigm. Several studies [5], [6] attempt to apply machine learning technologies such as deep Q-learning and Actor-critic reinforcement learning to solve the complex offloading problem when LEO satellites are involved. Studies such as [7], [8] and [9] leverage the edge/cloud resource to boost satellite computing by formulating the problem as an optimization problem such as game theory. These studies aim to solve the

Received 11 April 2024; revised 26 June 2024; accepted 5 August 2024. Date of publication 26 September 2024; date of current version 18 December 2024. This work was supported in part by the National Key Research and Development Program of China under Grant 2022YFB3102301; in part by the Science Fund for Creative Research Groups of the National Natural Science Foundation of China under Grant 62221003; in part by the National Natural Science Foundation of China under Grant 62472240, Grant 62471066, Grant 62394322, Grant 61932016, Grant 62132011, Grant 62132009, Grant 62101301, Grant 62394323, Grant 92167204, and Grant U22B2031; in part by the National Science Foundation for Distinguished Young Scholars of China under Grant 61825204; in part by Beijing Outstanding Young Scientist Program under Grant BJJWZYJH01201910003011; in part by the Ant Group through CCF-Ant Innovative Research Program under Grant CCF-AFSGRF20210023; and in part by Beijing National Research Center for Information Science and Technology under Grant BNR2023TD03006. *(Corresponding author: Mu Wang.)*

Su Yao is with Beijing National Research Center for Information Science and Technology (BNRist), Tsinghua University, Beijing 100190, China (e-mail: yaosu@tsinghua.edu.cn).

Yiying Lin, Mu Wang, and Changqiao Xu are with the State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing 100876, China (e-mail: linyiying@bupt.edu.cn; muwang@bupt.edu.cn; cqxu@bupt.edu.cn).

Ke Xu and Mingwei Xu are with the Department of Computer Science and Technology, Tsinghua University, Beijing 100190, China, and also with the Zhongguancun Laboratory, Beijing 100081, China (e-mail: xuke@tsinghua.edu.cn; xumw@tsinghua.edu.cn).

Hongke Zhang is with the School of Electronic and Information Engineering, Beijing Jiaotong University, Beijing 100044, China (e-mail: hkzhang@bjtu.edu.cn).

Digital Object Identifier 10.1109/JSAC.2024.3460083

problem of providing efficient computation resource sharing and task scheduling among satellites and ground computing centers, issues coming from coordination among huge-scale constellations, seamless data transmission between space-ground, and heterogeneous computation capabilities [10] are yet to be solved. There are three main challenges to providing highly efficient computation offloading to large-scale LEO constellations:

1) Increasing constellation scales. The scale of LEO constellations has been continuously expanding, and relying solely on a centralized control for task scheduling will encounter issues related to complexity and scalability in the future. On the other hand, a fully distributed system also poses significant challenges for LEO constellations. The instability of inter-satellite and satellite-to-ground links, along with limited bandwidth, makes it difficult for different satellites and between satellites and ground stations to frequently interact to determine the global optimum. This limitation restricts the performance of fully distributed decision-making.
2) Unstable ground-space links. LEO satellites move very fast to maintain their orbit, and namely link session between the satellite and the ground center can be very short. For example, a 400km height LEO satellite visibility time with a single ground station (GS) could range from a few minutes to several minutes. Maintaining dedicated links in such an environment for information exchange that is required by online offloading decision-making is economically impractical. Besides, directly delivering raw data for AI model training/inference can be also costly.
3) Computation capability differences among LEO satellites. Satellites are highly diverse in computation capability, i.e., early LEO satellites can only support simple computation tasks based on CPU, and recently launched satellites can support machine learning by including GPU/NPU units [11]. The computational overhead and inference performance of the same model can vary significantly across different satellite platforms. Therefore, it is necessary to configure model parameters and network structures according to different satellite platforms, which is yet to be considered in current studies.

We solve the above challenges by proposing LEOEdge, a decentralized computation offloading scheme for supporting AI applications over large LEO constellations with satellite-ground collaboration. For the first challenge, LEOEdge provides a decentralized task offloading strategy that enables each satellite to decide the place to execute the AI model individually. For the second challenge, LEOEdge estimates the workloads based on historical information without requiring frequent data exchange between satellite and ground. Besides, LEOEdge also applies a seamless data transmission scheme to dynamically select the relay to forward data when satellites moves out the communication range of the GS. For the third challenge, we include a neural architecture searching (NAS) algorithm to maximize resource utilization by generating a personalized AI model for each satellite based on its computation capability. Our contributions are as follows:

1) We design a decentralized task offloading framework based on stochastic optimization for LEO constellations, which decides the best place in terms of the accuracy and delay to perform the AI inference.
2) We apply NAS and seamless data transmission to enable improving the computation utilization within the satellite-ground computing collaboration. The approach also aims to minimize the interference in data transmission between space and ground, which caused by the high-speed movement of LEO satellites.
3) We conduct a series of evaluations based the real-world traces to show how the proposed LEOEdge provides an efficient AI inference for the LEO constellations environment.

The rest of the paper is organized as follows: Sec. II briefly introduces the background and related work; Sec. III presents the system overview; Sec. IV describes the model components of the system; Sec. V presents the three-layer decomposition problem in LEO satellites; Sec. VI illustrates the layer optimization of task scheduling in LEOEdge; Sec. VII gives the seamless data transmission over the LEOEdge; Sec. VIII gives the detailed design of adaptive modeling in LEOEdge; Sec. IX provides the evaluation results and Sec. X concludes the paper.

II. Background and Related Work

A. Background of Edge Assisted LEO Systems

As the tasks in space become increasingly complex and the demand for intelligence grows, LEO satellites are now capable of supporting a multitude of complex AI tasks. These tasks range from identifying targets in remote sensing images, image segmentation, cooperative monitoring, and perception, etc.

A typical use case is Earth observation missions [12]. These missions involve a large number of sensors on satellites to capture Earth images and process them through nodes on the satellite. Because directly transmitting raw data would consume a substantial amount of bandwidth, the limited and sometimes unstable link between the satellite and Earth makes it difficult to ensure that these image data can be timely transmitted to the ground for processing. At this point, the processing units on the satellite, such as CPUs, GPUs, etc., can be utilized to process the data, and the processed data or results can be directly transmitted to the ground, thereby effectively reducing the processing latency of AI tasks.

Performing AI tasks on satellites holds potential. Nonetheless, this approach encounters challenges due to the heterogeneous computational capacities of satellites. This variance is due to differences in satellites' functions, production times, and payloads, causing some to struggle with complex tasks. Rapid updates in AI and limited computing power on satellites also pose problems, limiting the size of deployable models and their accuracy.

To effectively accommodate a wide range of AI applications within LEO satellite constellations, integrating edge computing becomes imperative. This approach involves the strategic deployment of computational servers at ground stations, specifically designated to handle tasks relayed from satellites. Within such an edge-enhanced LEO ecosystem, it is the prerogative of the LEO satellite to ascertain whether specific tasks should be offloaded to these GS servers. Upon completion of these tasks, the outcomes are promptly relayed back either directly to the satellite or to the end-users on the ground, ensuring a seamless flow of information and optimized task execution. In the following sections, we will introduce related work addressing these challenges.

B. Related Work

Enabling AI inference on satellite platforms has become increasingly attractive recently. AsyncFLEO [13] introduces an innovative asynchronous FL framework that strategically leverages the accessibility of satellite local models opportunistically, eliminating the need to wait for the complete availability of all models. In [14], the challenge of deploying AI applications is conceptualized as a dual problem involving both model propagation and sink satellite scheduling. The former extends the traditional FL star topology by introducing horizontal intra-plane communication pathways. The latter leverages the predictability inherent in satellite orbiting patterns. Giuffrida et al. [15] propose a tailored convolutional neural network (CNN) implemented on a satellite platform to enhance the capabilities of remote sensing for AI applications. Razmi et al. [16] implement on-board FL orchestrated by an out-of-constellation parameter server (PS). Considering orbital movements, a communication scheme is proposed by specifically tailoring to running a synchronous FL procedure within a satellite mega-constellation orchestrated by an out-of-constellation PS.

Executing the AI inference over the can be difficult due to the resource limitation on satellite, and recent advanced attempts to offload satellite inference tasks to the edge computing on the ground.

Studies apply the ML-based offloading methods to satellites. In [17], a three-tier computation offloading architecture based on multi-agent deep reinforcement learning is given, which mainly considers offloading location selection and transmission energy allocation issues. In [5], Qiu et al. formulate the resource allocation problem as a joint optimization problem and use a novel deep Q-learning approach to learn the optimal resource allocation strategies. In [6], a state-action-reward-state-action based actor-critic reinforcement learning resource allocation strategy is proposed. This approach facilitates optimal resource allocation and IoRT data scheduling by leveraging causal information within LEO satellites. However, ML-based methods require feeding the model with a large volume of operation information and sufficient parameters to ensure the offloading performance, which is at the cost of extra computation resources. Specifically, the scheduling in these methods requires additional CPU and storage resources to maintain the ML model for continuously calculating the place to offload tasks, making deploying the scheduling on resource-limited satellites infeasible. If we deploy this model on the ground, frequent data exchange between the satellite and the ground is required to ensure the model's accuracy, which consumes precious communication resources.

Several studies attempt to facilitate computing resource allocation by the optimization methods. To ensure the computing offloading time of LEO satellites, the study in [18] introduces the dynamic offloading strategy that minimizes the overall delay of tasks from terrestrial users in satellite edge computing network with the energy and computing capacity constraints of the LEO satellite. In [19], a multi-layer multi-access edge computing system is given out for LEO satellites edge-assisted, This system addresses a joint optimization problem involving the allocation of communication and computing resources, ensuring satisfaction of users' computing latency requirements. In [20], Xie et al. consider the problem of multi-layer edge computing architecture design and heterogeneous edge computing resource co-scheduling and propose a satellite–ground edge computing framework that enables users to obtain computing services anywhere in the world and reduce redundant network traffic. In [21], a novel orbital edge computation task allocation algorithm grounded in a greedy strategy is introduced for LEO satellite networks. This algorithm extends the computational capabilities to the satellite network, thereby furnishing users with diverse and multi-tiered computing resources. Considering the large scale of the LEO constellation and the fact that most nodes may not be able to receive control commands from the ground in real-time, a decentralized lightweight offloading scheme is required to allow each satellite to decide where to execute the model.

Similar to our work, studies [22] and [23] design decentralized computation offloading schemes in LEO satellites. In [22], Vasisht et al. propose a low latency distributed downlink. In [23], StarFront introduces a content distribution framework that collaboratively exploits cache nodes within both LEO satellites and terrestrial clouds, aiming to efficiently optimize content access latency at a cost-effective level. Although these studies also focus on distributed computing offloading, our work is different in the following perspectives: (1) Different from these studies that use distribution architecture to minimize latency, LEOEdge also introduces a layered optimization for better scheduling of different compute resources; (2) We employ seamless data transmission techniques and NAS to enhance communication efficiency and optimize computational utilization in the collaborative computing framework between satellites and ground stations; (3) To accommodate the dynamic and complex environment in space, each LEO satellite applied the DL-based method to address the issue rather than using conventional optimization approaches.

III. System Overview

We build LEOEdge, a satellite-ground cooperation platform designed to augment AI applications operating over LEO satellites. Fig. 2 shows the structure of the proposed LEOEdge, which mainly includes several modules deployed over edge computing and satellites. Both the satellite and ground side deploy the computing container to execute the AI inference model. On the satellite side, LEOEdge deploys a seamless data transmission module that prevents communication breakdowns

TABLE I
COMPARISON OF RELATED STUDIES

Model Name	Support AI application	Decentralized scheduling	Machine learning-based	Heterogeneous computing	Communication optimization
AsyncFLEO	✓	✗	✓	✗	✓
FedLEO	✓	✓	✓	✗	✓
CloudScout	✓	✗	✓	✗	✗
FedISL	✓	✗	✓	✗	✓
MADRL	✗	✓	✓	✗	✓
Q-learning	✗	✗	✓	✗	✗
SACRL	✗	✗	✓	✗	✓
Dynamic Offloading	✗	✗	✗	✗	✓
Multi-Layer Offloading	✗	✗	✗	✗	✓
STECN	✗	✗	✗	✓	✓
OEC-TA	✗	✗	✗	✗	✓
L2D2	✗	✓	✗	✗	✓
StarFront	✗	✓	✗	✗	✓

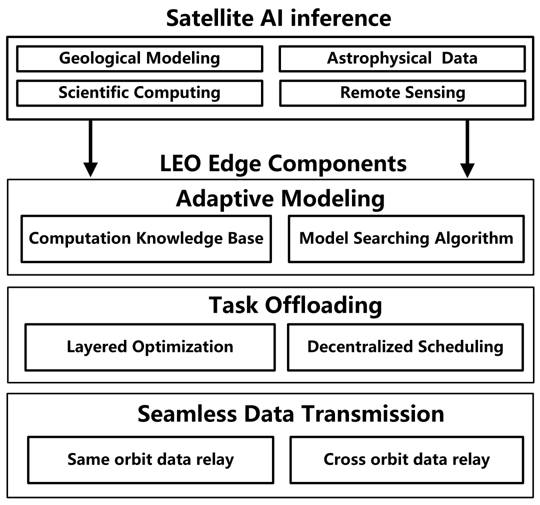


Fig. 2. System overview of LEOEdge.

in model inference, which improves efficient data transmission. Task offloading module to orchestrate the distribution of computation tasks across available units. It dynamically schedules computation tasks, taking into consideration the real-time status of the network and the computational demands of the model. On the ground side, the edge computing server applies the adaptive modeling scheme to select the models for the computation units including edge servers and satellites according to their computation capabilities. Based on the On-satellite and On-ground design, LEOEdge provides a high-performance model inference capability, specifically tailored to the unique demands of AI applications in LEO satellites.

A. Task Offloading

LEOEdge introduces layered optimization and decentralized scheduling that realizes the offloading of computation tasks on a case-by-case basis. When the framework implements optimization among different layers, the policy decisions of the upper layers need to take into account the resource status and arithmetic constraints of the lower layers, and the scheduling decisions of the lower layers need to be fed back to the upper layers to guide future policy adjustments. Complementary to layered optimization is the distributed scheduling mechanism. Distributed scheduling refers to the process of allocating and scheduling computational tasks among multiple computing nodes. This scheduling mechanism allows individual nodes to dynamically adjust task offloading decisions based on real-time network conditions, resource availability, and task priorities and requirements.

B. Seamless Data Transmission

The line-of-sight contact time between a particular LEO satellite and a ground base station may relatively short. A single transit of a satellite may last from 7 to 30 minutes [24]. During this limited communication time, individual LEO satellites are often unable to fully take on the reasoning of large AI tasks due to their own workload and limited computational resources. LEOEdge introduces to avoid the failure of inference results return to the satellite due to the satellite out of the communication range of GS. By leveraging the predicted arrival time of the satellite and incorporating relay point forwarding mechanisms, LEOEdge enhances the continuity of data exchange. This approach significantly bolsters the reliability and efficiency of satellite-borne AI services, ensuring more robust and continuous AI data processing capabilities in space-based operations.

C. Adaptive Modeling

To minimize the impact of heterogeneous computing in LEO satellites, it is very crucial to choose the appropriate neural network for different computing units such as CPU, GPU, and NPU. Thus, the adaptive modeling technology is proposed on computation knowledge base. When the LEO satellite is prepared to run the related AI application, the ground will use the NAS to select the best-adapted model based on the satellite's device information to fully utilize the satellite's computing.

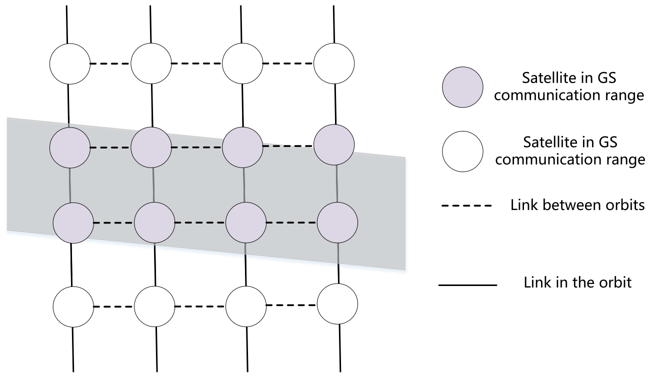

Satellite in GS communication range
Satellite in GS communication range
Link between orbits
Link in the orbit

Fig. 3. Network model of satellite constellation.

Based on the above modules, the process of AI inference over the LEOEdge can be described as follows: The AI inference task subscriber submits the ML scripts containing the AI model information to the edge computing server on the ground. The adaptive modeling module parses the AI model information and generates the models for each computation unit based on their computation compatibility, i.e., model execution time. The AI inference models are distributed and deployed on both the ground edge servers and the satellites. Once the satellite generates inference tasks, it will assist in the offloading of computational tasks through seamless data transmission. These data first go to the task scheduler to determine whether it can be processed on local or on remote ground edge servers. If the scheduler decides to process the inference model on locally, the features are forwarded to the local model to output the inference results. Otherwise, the features are forwarded to the models on the ground edge server, and the edge server returns the inference results to the satellite. In the next three sections, we will describe the designate of the main component task scheduler, seamless data transmission and adaptive modeling.

IV. System Modeling

A. Network Model

As illustrated in Fig. 3, suppose there are N satellites in the satellite constellation, each orbiting the Earth along a fixed trajectory. Assume that the constellation comprises M orbits, with N_m satellites in each orbit, where $m = 1, 2, \ldots, M$. Each satellite in an orbit maintains a link with its adjacent satellites in the same orbit. Considering that satellites in different orbits have varying speeds, stable links cannot be maintained between satellites on different orbits. Instead, data exchange is facilitated through opportunistic communication. Suppose there is a GS S connected to a server cluster with J servers, capable of providing computational support for tasks on the satellite. Due to the limited communication range of the GS, satellites can offload inference tasks only when they pass within the range of the GS. Since the speed and orbit of the satellites are fixed, based on the current position and orbital parameters of each satellite, it is possible to calculate the link time between satellites in adjacent orbits, as well as the communication time between the satellite and the GS. Assume the data rate between n_i and S is $l(n_i, S)$. Considering the extreme communication environment between the satellite and the ground link, due to the underlying communication errors and packet loss caused by long-distance transmission, it is inevitable. Assuming the packet loss rate is p. For n_i, we let t_{n_i} denote raw data and result need to be exchanged between n_i and S, the minimum data transmission delay can be estimated by

$$d_{n_i} = \frac{t_{n_i} p}{l(n_i, S)} \tag{1}$$

For the case of executing the AI inference model locally, the transmission delay can be omitted and is equal to zero.

B. Computation Model

In LEOEdge, LEO satellites are equipped with computation modules to support various on-board AI inference tasks, such as CV for remote sensing. Also, when the satellite enters the communication range of the GS, some of the complex computational tasks on the satellite can be offloaded to the GS. However, due to the limited processing capacity of the GS servers, deciding which tasks to offload from the satellite to the GS is crucial for overall execution efficiency. For each n_i, the deployed AI model is m_i with inference accuracy a_i. We denote the execution time of m_i on n_i as $c(m_i)$. The server clusters S within the GS consists of the J servers, and a load balance scheduler (LBS) allocates the arrival inference tasks to each server according to server's instant workload. The workload of each server is measured by the number of tasks allocated, i.e., the CPU occupancy. For simplicity, we assume that all servers deploy the inference model with the same architecture and AI inference tasks offloaded to the server utilize the same model. Therefore, the server's workload can be measured by the number of arriving tasks. Let the $L_j(Q)$ denote the workload of j when the number of allocated tasks is Q. Besides, since the server side is more powerful in computation and a larger model can be deployed, offloading the AI inference tasks can achieve better accuracy, i.e., $a_s \leq a_i$. Furthermore, considering the virtualization technologies are widely used in current edge computing, each server runs multiple computing containers and each container can individually execute AI inference. Assume there are O containers for S and O_j is set of containers over the sever j, the workload of each container can be given by $L_o(Q), o \in O$.

V. Task Offloading Problem

The main objective of the task offloading in LEO satellites is the improve the overall AI inference performance in terms of the inference accuracy and execution time, and to ensure the overall servers are within a reasonable workload. For the first objective, we introduce an offloading advantage operator in the form of

$$F(\alpha\Delta_c(i) + (1-\alpha)\Delta_a(i)) \tag{2}$$

where

$$\Delta_c(i) = c(m_i) - c(m_e) - d_{n_i} \tag{3}$$

$$\Delta_a(i) = a_s - a_i \tag{4}$$

$$\alpha \in [0, 1] \tag{5}$$

Without loss of the generality, $F(.)$ is concave and equals to zero when $\alpha\Delta_c(i)+(1-\alpha)\Delta_a(i) < 0$. This function measures the benefit of offloading tasks to the edge server. The greater the difference in time between executing the task at the edge (including the time for data interaction and the execution time on the edge server) and executing it locally, and the higher the accuracy of the edge model compared to the local model, the greater the value of this function. This indicates that the benefit of offloading to the edge computing server at the GS is higher. To constrain the workload, we introduce the following constraints:

$$L_o(Q_o) < P, \ \forall o \in O \tag{6}$$

where Q_o denotes the current number of tasks running on the container o. Let $x_{i,m}$ denote the decision policy for n_i, where $x_{i,m} = 1$ indicates the n_i offloading its m-th tasks to the edge server and o otherwise. Based on the above analysis, we can formulate the task offloading optimization problem as follows:

$$\begin{aligned} &\max \sum_{i\in N}\sum_{m=1}^{M} F(\alpha\Delta_c(i) + (1-\alpha)\Delta_a(i))x_{i,m} \\ &\mathrm{s.t\ L_o(Q_o) < P,\ \forall o \in O} \end{aligned} \tag{7}$$

In the aforementioned problem, there is only one decision variable $x_{i,m}$, which is the decision made by the satellite on whether to offload the task to the edge server. In fact, to meet the strict constraints, task offloading is a multi-level decision-making process. Firstly, each satellite decides whether to offload each task onboard to the edge. When the raw data of the computational task is transmitted to the GS, the LBS of S decides to which server the task should be scheduled. Once the raw data arrives at this server s, the server then decides on which container to run the task. Thus, we further decompose the optimization problem into a three-layer decomposition problem as follows:

$$\begin{aligned} &\max \sum_{i\in N}\sum_{m=1}^{M}\sum_{o\in O} F((\alpha\Delta_c(i) + (1-\alpha)\Delta_a(i))y^3_{i,m,o}) \\ &\text{s.t } L_S(Q_S + \sum_{i\in N}\sum_{m=1}^{M} y^1_{i,m}) < p \\ &\quad \frac{1}{S}\sum_{s\in S} L_s(Q_s + \sum_{i,m\in N,M} y^2_{i,m,s}) \\ &\qquad \le L_S(Q_S + \sum_{i\in N}\sum_{m=1}^{M} y^1_{i,m}) \\ &\quad \frac{1}{s}\sum_{o\in s} L_o(Q_o + \sum_{i,m\in N,M} y^3_{i,m,o}) \\ &\qquad < L_s(Q_s + \sum_{i,m\in N,M} y^2_{i,m,s}), \\ &\quad s \in S \end{aligned} \tag{8}$$

where $y^1_{i,m}$ is the first layer variable equals to $x_{i,m}$. $y^2_{i,m,s}$ is the second layer variable that 1 indicates the m-th task of n_i is offloaded to the s. $y^3_{i,m,o}$ is the third layer variable that 1 indicates the m-th task of n_i is offloaded to the o of s. Q_S, Q_s and Q_o denote the current number of tasks processed by the server clusters S, server s and container o, respectively. As shown in Fig. 4, the task scheduler of the satellite is on the highest layer, the LBS of the S is on the second, and the s is the bottom layer. A given node in the higher layer can be treated as an allocator to the nodes within the corresponding cluster of the lower layer.

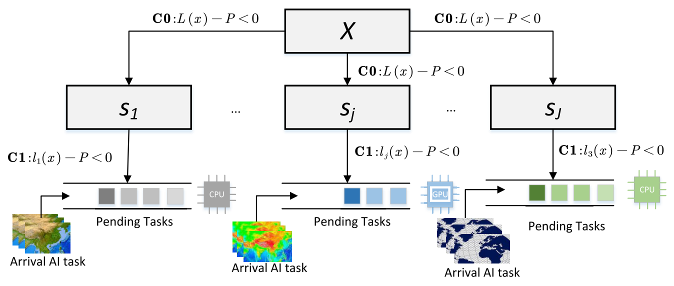

$\mathbf{C0}: L(x) - P < 0$
X
$\mathbf{C0}: L(x) - P < 0$
$\mathbf{C0}: L(x) - P < 0$
s_1
...
s_j
...
s_J
$\mathbf{C1}: l_1(x) - P < 0$
$\mathbf{C1}: l_j(x) - P < 0$
$\mathbf{C1}: l_3(x) - P < 0$
CPU
GPU
CPU
Pending Tasks
Pending Tasks
Pending Tasks
Arrival AI task
Arrival AI task
Arrival AI task

Fig. 4. Layered problem decomposition.

In the distributed solutions, each layer independently handles its local computations. Coordination among these layers is achieved by the exchange of parameters between layers. Typically, these computations are iterative. In each iteration, the upper-layer elements depend on the outcomes of multiple iterations from the lower layer, a reflection of the solution procedure's inherent nature. Consequently, the upper layers operate on slower time-scales, as they must await the results of the lower layer's iterations. In the classic setup of (1), the main controller/distributor waits for parameter transmissions from the end-users. However, with multi-layer decomposition, the main controller's communication is limited to sub-controllers (intermediate layer entities) rather than all users. Assuming that link latency is proportional to the distance and the number of concurrently communicating nodes, shifting computational and communicative loads to sub-entities via multi-layer decomposition can decrease latency. This results in accelerated iterations at the top layer.

VI. LEOEdge Task Scheduling

To solve the three-layer decomposition problem, we first consider its Lagrangian:

$$\begin{aligned} &L(\mathbf{y}, \lambda, \mu, \boldsymbol{\eta}) \\ &\quad = \sum_{i\in N}\sum_{m=1}^{M}\sum_{o\in O} F_i(y^3_{i,m,o}) - \lambda(g_S(Q_s, \boldsymbol{y^1}) - p) \\ &\qquad - \mu(\sum_{s\in S} g_s(Q_s, \boldsymbol{y}^2_s) - g_S(Q_s, \boldsymbol{y^1})) \\ &\qquad - \sum_{s\in S}\eta_s(\sum_{o\in s} g_o(Q_o, \boldsymbol{y^3}_o) - g_s(Q_s, \boldsymbol{y^2}_s)) \end{aligned} \tag{9}$$

where the

$$F_i(y^3_{i,m,o}) \triangleq F((\alpha\Delta_c(i) + (1-\alpha)\Delta_a(i))y^3_{i,m,o}) \tag{10}$$

$$g_S(Q_s, \boldsymbol{y^1}) \triangleq L_S(Q_S + \sum_{i\in N}\sum_{m=1}^{M} y^1_{i,m}) \tag{11}$$

$$g_s(Q_s, \boldsymbol{y}^2_s) \triangleq \frac{1}{S} L_s(Q_s + \sum_{i,m\in N,M} y^2_{i,m,s}) \tag{12}$$

$$g_o(Q_o, \boldsymbol{y^3}_o) \triangleq \frac{1}{s} L_o(Q_o + \sum_{i,m\in N,M} y^3_{i,m,o}) \tag{13}$$

$\boldsymbol{y^1}$, $\boldsymbol{y}_s^2$, $\boldsymbol{y^3}_o$ are the column vectors with entries $y_{i,m}^1$, $y_{i,m,s}^2$, $y_{i,m,o}^3$, $i \in N, m = 1, \ldots, M$, respectively. $\boldsymbol{\eta}$ is the column vector with the entries η_s, $s \in S$. Considering the η_o is the Lagrange multiplier corresponding to the $(\sum_{o\in s} L_o(Q_o + \sum_{i,m\in N,M} y_{i,m,o}^3) - L_s(Q_s + \sum_{i,m\in N,M} y_{i,m,s}^2)$, η_o decouples the layer 2 and 3 decisions. Similarly, the μ corresponding to the $\sum_{s\in S} L_s(Q_s + \sum_{i,m\in N,M} y_{i,m,s}^2) \leq L_S(Q_S + \sum_{i\in N}\sum_{m=1}^{M} y_{i,m}^1)$ decouples the layer 1 and 2 decision.

For (7), its dual problem objective can be written as:

$$\mathcal{G}(\lambda, \mu, \boldsymbol{\eta}) = \max_{\mathbf{y}} \mathrm{L}(\mathbf{y}, \lambda, \mu, \boldsymbol{\eta}) \tag{14}$$

and the dual problem is

$$\min_{\lambda\geq 0, \mu\geq 0, \boldsymbol{\eta}\geq 0} \mathcal{G}(\lambda, \mu, \boldsymbol{\eta}) \tag{15}$$

Given the objective of (7) is concave and the constraints are linear, we have

$$\begin{aligned}&\min_{\lambda\geq 0, \mu\geq 0, \boldsymbol{\eta}\geq 0} \mathcal{G}(\lambda, \mu, \boldsymbol{\eta}) \\ &= \sum_{i\in N}\sum_{m=1}^{M}\sum_{o\in O} F((\alpha\Delta_c(i) + (1-\alpha)\Delta_a(i))(y_{i,m,o}^3))^* \end{aligned}\tag{16}$$

where $(y_{i,m,o}^3))^*$ is the optimizer of the (7). By observing the dual forms, we can find that the dual optimization leads to the local task scheduling sub-problems for the Lagrangian operator that can be solved in a distributed manner.

A. Distributed Algorithm

In practice, the offloading algorithm updates in iterations that solve the dual variables of the optimum AI inference task offloading. Let the states of the optimization on iteration $\theta(k)$ as follows:

$$\boldsymbol{a}(k) = (\mathbf{y}(k), \lambda(k), \mu(k), \boldsymbol{\eta}(k)) \tag{17}$$

We can write the update rules, for every k, the third layer iterative as follows:

$$\begin{aligned} y_{i,m,o}^3(k+1) &= y_{i,m,o}^3(k) + \gamma\frac{\partial L(\boldsymbol{a}(k)}{\partial y_{i,m,o}^3(k)} \\ &= y_{i,m,o}^3(k) + \gamma\frac{\partial F_i(y_{i,m,o}^3(k))}{\partial y_{i,m,o}^3(k)} \\ &\quad - \gamma\frac{\partial L_o(Q_o + \sum_{i,m\in N,M} y_{i,m,o}^3(k))}{s\partial y_{i,m,o}^3(k)} \end{aligned}\tag{18}$$

$$\begin{aligned} \eta_s(k+1) &= \eta_s(k) - \gamma\frac{\partial L(\boldsymbol{a}(k)}{\partial \eta_s} \\ &= \eta_s(k) - \gamma'(\sum_{o\in s} g_o(Q_o, \boldsymbol{y^3}_o(k)) \\ &\quad - g_s(Q_s, \boldsymbol{y^2}_s(k))) \end{aligned}\tag{19}$$

For the second layer iteration:

$$\begin{aligned} &y_{i,m,s}^2(k+1) \\ &= y_{i,m,s}^2(k) + \gamma'\frac{\partial L(\boldsymbol{a}(k))}{\partial y_{i,m,s}^2(k)} \\ &= y_{i,m,s}^2(k) + (\eta_s(k) \\ &\quad - \frac{\mu(k)}{S})\frac{\partial L_s(Q_s + \sum_{i,m\in N,M} y_{i,m,s}^2(k))}{\partial y_{i,m,s}^2(k)} \end{aligned}\tag{20}$$

$$\begin{aligned} \mu(k+1) &= \mu(k) - \gamma'\frac{\partial L(\boldsymbol{a}(k)}{\partial \mu(k)} \\ &= \mu(k) - \gamma'(\sum_{s\in S} g_s(Q_s, \boldsymbol{y}_s^2) - g_S(Q_S, \boldsymbol{y^1})) \end{aligned}\tag{21}$$

For the top layer iteration:

$$\begin{aligned} &y_{i,m}^1(k+1) \\ &= y_{i,m}^1(k) + \gamma''\frac{\partial L(\boldsymbol{a}(k))}{\partial y_{i,m}^1(k)} \\ &= y_{i,m}^1(k) - (\lambda(k) + \mu(k))\frac{\partial L_S(Q_S + \sum_{i\in N}\sum_{m=1}^{M} y_{i,m}^1(k))}{\partial y_{i,m}^1(k)} \end{aligned}\tag{22}$$

$$\begin{aligned} &\lambda(k+1) \\ &= \lambda(k) + \gamma''\frac{\partial L(\boldsymbol{a}(k))}{\partial \lambda(k)} \\ &= \lambda(k) + \gamma''(g_S(Q_s, \boldsymbol{y^1}) - p) \end{aligned}\tag{23}$$

Based on the above iterations, our task offloading algorithm works as follows:

(1) In the third layer, each container finds the optimal decision on $y_{i,m,o}^{3*}$ for the fixed local operator η_s of its deployed server by using (18). The container updates the η_s according to local optimizer $y_{i,m,o}^{3*}$ and fixed upper layer optimizer $y_{i,m,s}^{2*}$ by using the (19). The pseudo-code of the detailed iterations is given in **Algorithm 1**. The fourth line of the algorithm represents a crucial step executed on the container, where each container calculates the optimal task allocation strategy. The nineteenth line is a key step on the server side, where the server updates η_s based on the observed information. The **Algorithm 1** shows that the container decides on task offloading by only communicating with the server.

(2) In the second layer, each server determines its optimal total number of offloaded tasks by $\sum_{i,m\in N,M} y_{i,m,s}^{2*}$ according to the fixed operator μ from the LBS by using (20). The LBS updates its operator μ by (21) with the fixed value of $y_{i,m,s}^{2*}$ and upper layer $\boldsymbol{y}^1$. The pseudo-code of the second layer iteration is given in **Algorithm 2**. The third line in the algorithm represents a crucial step executed on the server, which involves computing the optimal strategy for allocating tasks to different containers. The sixteenth line is a key step on the LBS side, where μ is updated based on the observed information. The **Algorithm 1** shows that the server decides on task offloading by only communicating with the LBS.

(3) In the top layer, each satellite n_i its optimal task offloading strategy $y_{i,m}^{1*}$ for each generated task m by using (22), according to the fixed value of λ and μ, the LBS updates the λ according to (23) with the given $\boldsymbol{y}^1$. The pseudo code of the top layer iteration is given in **Algorithm 3**. In the algorithm, the fifth line represents a critical step executed on the satellite, determining whether to offload tasks to edge servers. The seventeenth line is a key step on the LBS side, updating λ based on the received information.

Hence, according to the output and input in **Algorithm** 1, 2 and 3, the information exchange only occurs between the neighboring layers. The algorithm executors of neighboring layers are directly linked and thus ensure communication efficiency. Besides, each layer can individually iterate to the optimum without synchronization between the layers.

Algorithm 1 **Bottom Layer Task Scheduling** //Run on Container and Server

```
Input: model execution time $\{c(m_i)\}_{i\in N}$, $c(m_e)$;
       execution accuracy $\{a_i\}_{i\in N}$, $a_s$; transmission
       latency $\{d_{n_i}\}_{i\in N}$; Pending task queue
       $\{\{Q_o\}_{o\in s}\}_{s\in S}(k), k = 1,\ldots$; $\epsilon$;
Output: $\{y_o^{3*}\}_{o\in s}$, $Q_o(k)$, $\{\eta_s^*\}_{o\in s}$;
1  while $!k$ do
2      // Container $o$:
3      Observe the local $Q_o(k)$ and $\eta_s(k+1)$;
4      Calculate $\{y_{i,m,o}^{3*}\}_{i\in N,m=1,\ldots,M}$ by (18);
5      Push $\{y_{i,m,o}^{3*}\}_{i\in N,m=1,\ldots,M}$, $Q_o(k)$ to server $s$;
6      //Server $s$:
7      for $o \in S$ do
8          Observe $Q_o(k)$;
9          $\boldsymbol{y^3}_o(k) \leftarrow \{y_{i,m,o}^{3*}\}_{i\in N,m\in 1,\ldots,M}$;
10         if $\boldsymbol{y^3}_o(k) - \boldsymbol{y^3}_o(k-1) < epsilon$ then
11             $\boldsymbol{y^3*}_o \leftarrow \boldsymbol{y^3}_o(k)$;
12         end
13         $g_o(Q_o, \boldsymbol{y}_0^3) \leftarrow g_o(Q_o(k), \boldsymbol{y^3}_o(k))$;
14     end
15     Observe $Q_s(k)$;
16     Observe $\{y_{i,m,s}^2(k)\}_{i\in N,m=1,\ldots,M}$;
17     $\boldsymbol{y^2}_s(k) \leftarrow \{y_{i,m,s}^2(k)\}_{i\in N,m=1,\ldots,M}$;
18     $g_s(Q_s, \boldsymbol{y}_s^2) \leftarrow g_s(Q_s(k), \boldsymbol{y^2}_s(k))$;
19     Use $g_s(Q_s, \boldsymbol{y}_s^2)$ and $\{g_o(Q_o, \boldsymbol{y}_0^3)\}_{o\in s}$ to update
       $\eta_s(k+1)$ by (19);
20     Push $\eta_s(k+1)$ to $o, o \in s$ and LBS;
21     if $\eta_s(k+1) - \eta_s(k) \le \epsilon$ then
22         return $\{\boldsymbol{y^{3*}}_o\}_{o\in s}$, $\eta_s^* \leftarrow \eta_s(k+1)$
23     end
24     k++;
25 end
```

Remark 1: Our layered scheduling also applies to air segments, which we analyse here using UAVs. AI applications in UAVs [25] include computer vision, auto-navigation, and communications. However, UAVs typically lack the requisite on-board processing capabilities [26], which means that executing complex neural network models in UAVs doesn't work. Besides, UAVs contain various types of processors [27]. Our approach is equally capable of addressing its heterogeneous computing and improving higher inference accuracy and lower latency when applied to UAVs. Compared to the applications of LEO satellites in layered scheduling, UAVs are also positioned at the top layer of task scheduling, and we need to consider only the delay calculation between the UAV and GS. Firstly using a 3D coordinate system, we denote the coordinates of the GS as $w = \{x, y, 0\}$. We assume that the UAV is able to perform its mission properly at altitude H, so the coordinates of the i-th UAV at time t is denoted as $U_{n_i}(t) = \{x_t, y_t, H\}$. Similar to [28], we express the channel

Algorithm 2 **Second Layer Task Scheduling** //Run on Server and LBS

```
Input: $\eta_s(k), k = 1,\ldots$; Pending task queue
       $\{\{Q_o\}_{o\in s}\}_{s\in S}(k), k = 1,\ldots$;
       $\{y_{i,m}^1(k)\}_{i\in N,m=1,\ldots,M}, k = 1,\ldots$;$\mu(0)$;
       $\lambda(k), k = 1,\ldots$;
Output: $\{\boldsymbol{y}_s^{2*}\}_{s\in S}$,$\mu^*$;
1  while $!k$ do
2      //Server $s$:
3      Calculate the $\{y_{i,m,s}^{2*}(k)\}_{i\in N,m=1,\ldots,M}$ by (20);
4      Push the $\{y_{i,m,s}^{2*}(k)\}_{i\in N,m=1,\ldots,M}$ and $Q_s(k)$ to
       LBS;
5      //LBS:
6      for $s \in S$ do
7          Observe $\{y_{i,m,s}^{2*}\}_{i\in N,m=1,\ldots,M}$ ;
8          $\boldsymbol{y}_s^2(k) \leftarrow \{y_{i,m,s}^{2*}\}_{i\in N,m=1,\ldots,M}$;
9          if $\boldsymbol{y^2}_s(k) - \boldsymbol{y^2}_s(k-1) < epsilon$ then
10             $\boldsymbol{y^2*}_s \leftarrow \boldsymbol{y^2}_s(k)$;
11         end
12         $g_s(Q_s, \boldsymbol{y}_s^2) \leftarrow g_s(Q_s(k), \boldsymbol{y}_s^2(k))$;
13     end
14     $\boldsymbol{y}^1(k) \leftarrow \{y_{i,m}^1(k)\}_{i\in N,m=1,\ldots,M}$;
15     $g_S(Q_S, \boldsymbol{y^1}) \leftarrow g_S(Q_S(k), \boldsymbol{y^1}(k))$;
16     Use the $\{g_s(Q_s, \boldsymbol{y}_s^2)\}_{s\in S}$ and $g_S(Q_S, \boldsymbol{y^1})$ to update
       the $\mu(k+1)$;
17     Push the $\mu(k+1)$ to LBS;
18     if $\mu(k+1) - \mu(k) \le \epsilon$ then
19         return $\{\boldsymbol{y^2*}_s\}_{s\in S}$, $\mu^* \leftarrow \mu(k+1)$
20     end
21     k++;
22 end
```

power between the UAV and the base station as:

$$H_{n_i}(t) = \frac{\beta}{||U_{n_i}(t) - w||^2} \tag{24}$$

where β indicates the channel gain for a distance of 1m. According to Shannon formula, the transmission rate between UAV and GS is expressed as:

$$R_{n_i}(t) = B_i \log(H_{n_i}(t)P/\alpha + 1) \tag{25}$$

where B denotes the bandwidth between the i-th UAV and GS. P is transmit power. α is noise ratio. Thus, data transmission delay can be presented by

$$d_{n_i} = \frac{t_{n_i}p}{R_{n_i}(t)} \tag{26}$$

VII. Seamless Data Transmission

During the offloading of computational tasks, due to the rapid movement of LEO satellites and unstable communication links, raw data cannot be transferred within the communication window with the GS. Additionally, some inference tasks might require the GS to feedback the computation results to the satellite, which could also fail due to link failures or the satellite leaving the communication range. Missing the communication

Algorithm 3 **Top Layer Task Scheduling** //Run on Satellite and LBS

```
Input: $\mu(k), k = 1, \ldots$; Pending task queue
       $\{\{Q_o\}_{o\in s}\}_{s\in S}(k), k = 1, \ldots$;$\lambda(0)$;
       $\mu(k), k = 1, ldots$;
Output: $\boldsymbol{y}^{1*}$,$\lambda^*$;
1  while !$k$ do
2      // Top layer scheduling
3      //Satellite $i$:
4      Receive $\mu(k)$ and $\lambda(k)$;
5      Calculate the $\{y^{1*}_{i,m}(k)\}_{m=1,\ldots,M}$ by (22);
6      Push the $\{y^{1*}_{i,m}(k)\}_{m=1,\ldots,M}$ to LBS;
7      //LBS $i$:
8      Observe the $Q_S(k)$;
9      for $i \in N$ do
10         Observe the $\{y^{1*}_{i,m}\}_{i\in N, m=1,\ldots,M}$;
11         $\{\boldsymbol{y}^1(k)\}_i \leftarrow \{y^{1*}_{i,m}\}_{i\in N, m=1,\ldots,M}$;
12         if $\{\boldsymbol{y}^1(k)\}_i - \{\boldsymbol{y}^1(k-1)\}_i$ then
13             $\{\boldsymbol{y}^{1*}\}_i \leftarrow \boldsymbol{y}^1(k)\}_i\}$;
14         end
15     end
16     $g_S(Q_S, \boldsymbol{y}^1) \leftarrow g_S(Q_S(k), \boldsymbol{y}^1(k))$;
17     Use the $g_S(Q_S, \boldsymbol{y}^1)$ to update the $\lambda(k+1)$ by (23);
18     if $\{\lambda(k+1)\}_i - \{\lambda(k+1)(k)\} \leq \epsilon$ then
19         return $\{\boldsymbol{y}^{1*}\}$; $\lambda^* \leftarrow \lambda(k+1)$;
20     end
21 end
```

window necessitates the satellite to reroute to the ground station for transmission completion, leading to additional delays. To address this issue, we propose a neighbor-assisted seamless data transmission scheme to manage this challenge. Given the uncertainty and diversity inherent in interstellar terrestrial link channels [29], the influence exerted by channels across various frequency bands varies significantly. Therefore, we do not restrict our analysis to any specific channel conditions. Instead, we operate under the assumption that both data rates and link characteristics are stochastic in nature.

A. Raw Data Delivering

We first consider the case when delivering the raw data. When n_i determines to offload the inference task to the GS, it first determines whether the transmission can be completed within the communication time window. The communication time $T(o_i, l_G)$ between the satellite n_i and the GS can be calculated based on the satellite's orbital altitude o_i and the coordinates of the GS, as referenced in [30]. Since the orbit of the satellite is generally pre-given, this value can be directly calculated and considered a constant. Let the current relative time of entering the GS communication range be $T_0(i)$, the available communication time $T_a(i)$:

$$T_a(i) = T(o_i, l_G) - T_0(i) \tag{27}$$

If the raw data transmission time $d_{n_i} \leq T_a(i)$, n_i directly sets the GS as the destination and begins to transfer the data. Otherwise, if the raw data transmission time $d_{n_i} \leq T_a(i)$, the satellite selects the neighbor satellites as the relay to forward the raw data to GS. n_i first checks the neighbor satellite n_j within the same orbit is free to transfer the data. If the n_j is free, n_i calculates the communication time between n_j and n_i. Let the $\overline{rtt}$ denote the average round-trip-time between two satellites, the instance capacity between two satellites is C_{ij}, which can be estimated as in [31]. We have the instance data rate $k_{ij} = G_{ij}/\overline{rtt}$. Thus, the estimated transmission time using the n_j as the relay can be given by:

$$d_{n_{ij}} = \frac{t_{n_i} \times (p_{ij} + p)}{\min\{k_{ij}, l(n_i, S)\}} \tag{28}$$

If the $d_{n_{ij}} \leq T_a(i)$, n_i forward the raw data to n_j. If the n_j is busy (i.e., offloading task to GS) or $d_{n_{ij}} \geq T_a(i)$, then n_j selects the satellite from other orbits as the relay by the similar rule. For example, for each neighbor $n_{j'}$ from the other orbit, calculates the $d_{n_{ij'}}$ by using (28), and select n_{j*} with $\max_{N_i} d_{n_{ij'}}$ and $d_{n_{ij'}} \leq T_a(i)$. Otherwise, if the communication time of all neighbors is larger than the $T_a(i)$, n_i executes the task locally.

It is worth mentioning that the reason we do not consider the data transmission time with different cases in the offloading optimization problem is that such calculation can be computationally intensive. Because each container requires calculating and comparing the communication time of different cases. Instead, let each satellite determine the way to deliver the data is more efficient and practical.

B. Result Return

For inference tasks that require the computational results to be sent back to the satellite, the GS will first evaluate the communication windows of the nodes on the previously established raw data transmission link. If the communication window size is larger than what is required for the task return, the inference results will be sent back through the previously established link after the computation is completed. Otherwise, the satellite will further consider a new link for data delivery.

Let the data size of the inference result denote as s_{n_i}, for the case the n_i directly forwards the raw data, the total transmission time n_i to GS can be given by:

$$d'_{n_i} = \frac{(t_{n_i} + s_{n_i})p}{l(n_i, S)} + c(m_e) \tag{29}$$

The reason of adding $c(m_e)$ in the above equation is that when the n_i is required to send back the computational results, it is required to wait until the computation is completed at the GS before the data can be transmitted back. In the case of forwarding data with the assistance of relay n_j, we have the communication time:

$$d'_{n_{ij}} = \frac{(t_{n_i} + s_{n_i}) \times (p_{ij} + p)}{\min\{k_{ij}, l(n_i, S)\}} + c(m_e) \tag{30}$$

Similar to the previous scenario of transmitting only raw data, we first determine to directly forward raw data to GS if $d'_{n_i} \leq T_a(i)$. Otherwise, the n_i will first select an idle neighbor within the same orbit whose $d'_{n_{ij}} \leq T_a(i)$ as the relay. The above condition is not satisfied, a satellite with $d'_{n_{ij}} \leq T_a(i)$ in other orbits will be selected. If none of the above conditions are met, then the task is executed locally. The pseudo code of the seamless data transmission is given in **Algorithm 4**.

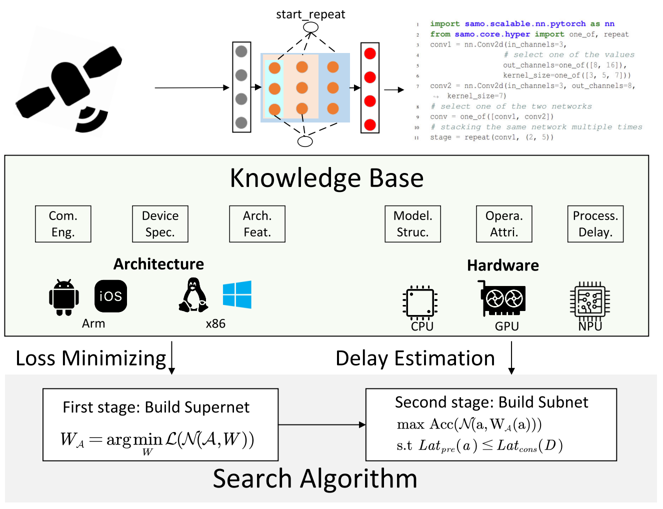
Algorithm 4 Seamless Data Transmission //Run on Satellite

```
Input: $T_a(i)$; $T_0(i)$; neighbor data rate $\{k_{ij}\}_{N_i}$ and
       packet loss rate $\{p_{ij}\}_{N_i}$; $t_{n_i}$; $s_{n_i}$; $p$; $T(o_i, l_G)$ ;
1  //Calculate the communication time;
2  Case I: Inference results required to be returned ;
3  Case II: Inference results not to be returned;
4  Case I:$d \leftarrow \frac{(t_{n_i}+s_{n_i})p}{l(n_i,S)} + c(m_e)$;
5  Case II: $d \leftarrow d_{n_i}$;
6  if $d \leq T(o_i, l_G) - T_0(i)$ then
7      Deliver the raw data to GS;
8  end
9  else
10     Case I: $d' \leftarrow \frac{(t_{n_i}+s_{n_i})\times(p_{ij}+p)}{\min\{k_{ij}, l(n_i,S)\}} + c(m_e)$;
11     Case II: $d' \leftarrow \frac{t_{n_i}\times(p_{ij}+p)}{\min\{k_{ij}, l(n_i,S)\}}$;
12     if $d' \leq T(o_i, l_G) - T_0(i)$ then
13         Deliver the raw data via $j$ to GS;
14     end
15     else
16         for $j \in N_i$ do
17             Case I: $d'_j \leftarrow \frac{(t_{n_i}+s_{n_i})\times(p_{ij}+p)}{\min\{k_{ij}, l(n_i,S)\}} + c(m_e)$;
18             Case II: $d'_j \leftarrow \frac{t_{n_i}\times(p_{ij}+p)}{\min\{k_{ij}, l(n_i,S)\}}$;
19             if $d_j \leq T(o_i, l_G) - T_0(i)$ then
20                 $K.append(d_j)$;
21             end
22         end
23         if $K$ is not empty then
24             Deliver the raw data via $j$ with $\min_K d_j$;
25         end
26         else
27             Execute the task locally;
28         end
29     end
30 end
```

VIII. Adaptive Modeling

The main purpose of adaptive modeling is to adaptively control the size and structure of the model based on the computation capability. As shown in Fig. 5, our proposed adaptive modeling allows developers to define the processing delay interval and the structure size interval. It automatically generate a network with the size that fits the device capability and declared requirement. This design goal is achieved by the combination of search space, search algorithm, and operator knowledge base.

Search space refers to the set of all possible architectures. For LEOEdge, search spaces should be able to cover all hardware configurations within the device-edge scenarios, decouple from the search algorithm, and be convenient for code modifications. Based on PyGlove, we apply two functions `oneof` and `repeat`. The function `oneof` defines the size of


start_repeat
Knowledge Base
Com. Eng.
Device Spec.
Arch. Feat.
Model. Struc.
Opera. Attri.
Process. Delay.
Architecture
Hardware
iOS
Arm
x86
CPU
GPU
NPU
Loss Minimizing
Delay Estimation
First stage: Build Supernet
$W_{\mathcal{A}} = \arg\min_W \mathcal{L}(\mathcal{N}(\mathcal{A}, W))$
Second stage: Build Subnet
max Acc($\mathcal{N}$(a, W$_{\mathcal{A}}$(a)))
s.t $Lat_{pre}(a) \leq Lat_{cons}(D)$
Search Algorithm

```
import samo.scalable.nn.pytorch as nn
from samo.core.hyper import one_of, repeat
conv1 = nn.Conv2d(in_channels=3,
                  # select one of the values
                  out_channels=one_of([8, 16]),
                  kernel_size=one_of([3, 5, 7]))
conv2 = nn.Conv2d(in_channels=3, out_channels=8,
 ↪  kernel_size=7)
# select one of the two networks
conv = one_of([conv1, conv2])
# stacking the same network multiple times
stage = repeat(conv1, (2, 5))
```

Fig. 5. Illustration of adaptive modeling.

convolution block where 'out_channels' and 'kernel_size' can be searched within the pre-defined space. `oneof` also supports the search over different convolution modules. The function `repeat` supports stacking the same network block multiple times by defining the stack times interval.

A. Neural Searching Algorithm

The structure and size of models that perfectly fit the device's capabilities are selected from the defined search space $\mathcal{A}$ via the search algorithm. LEOEdge supports personalized search algorithm deployment. Here, we use the one-shot searching algorithm as an example to illustrate how to generate the network for different computational platforms. This algorithm mainly consists of three stages: First, supernet weight optimization determines the weight of network $W_{\mathcal{A}}$ from $\mathcal{A}$ for ML models, by

$$W_{\mathcal{A}} = \arg\min_{W} \mathcal{L}(\mathcal{N}(\mathcal{A}, W)) \tag{31}$$

where $\mathcal{L}$ is loss function of the model, $\mathcal{N}(\mathcal{A}, W)$ denotes the network in space $\mathcal{A}$ with the weight W. The second optimization is to optimize the subnet structure by maximizing the estimated accuracy within the given processing time constraint,

$$\begin{aligned} &\max\ \mathrm{Acc}(\mathcal{N}(\mathrm{a}, \mathrm{W}_{\mathcal{A}}(\mathrm{a}))) \\ &\mathrm{s.t}\ \mathrm{Lat}_{\mathrm{pre}}(\mathrm{a}) \leq \mathrm{Lat}_{\mathrm{cons}}(\mathrm{D}) \end{aligned} \tag{32}$$

where $\mathrm{Acc}(\mathcal{N}(\mathrm{a}, \mathrm{W}_{\mathcal{A}}(\mathrm{a})))$ denotes the estimated accuracy for the subnet $\mathcal{N}(a, W_{\mathcal{A}}(a))$. $Lat_{pre}(a, D)$ indicates the latency of processing the structure a over device D. $Lat_{cons}(D)$ is the delay constraint over device D. Numerous methods such as genetic algorithms solve the above problem. Besides, generating subnet for different devices is easy under such an optimization framework, by only manipulating the $Lat_{cons}(D)$ in the second problem.

The final optimization is using Nesterov's AGD (NAG) method to make the loss function converge faster. We will define the tensor of the loss function as $\Gamma(k)$. Also introduce the auxiliary variable $\gamma[k]$ and $\gamma[0] = 1$. For the k-th iteration:

$$\Gamma[k] = \gamma[k] - \beta \times \nabla\mathcal{L}(\gamma[k]) \tag{33}$$

where β is set to 0.01 by repeated experiment. In contrast to the traditional setting of the momentum coefficient as a constant, our momentum coefficient is a function, which is denoted as $\nabla\mathcal{L}$. Meanwhile, this method extends the heavy ball approach by evaluating the gradient at an intermediate point $\gamma[k+1]$ rather than $\Gamma[k]$. For the each of $\gamma[k+1]$, the formula is as follow:

$$\gamma[k+1] = \Gamma[k+1] - Q[k](\Gamma[k] - \Gamma[k+1])$$
$$Q[k] = \frac{\gamma[k-1] - \gamma^2[k-1]}{\gamma^2[k-1] + \gamma[k]} \tag{34}$$

The loss function measures the degree of inconsistency between the model's predicted and true values. When the value of the loss function decreases rapidly, this indicates that the model is able to adjust its parameters more quickly to more accurately predict or fit the training data. For fast moving LEO satellites, our search algorithm will be more advantageous.

The searching performance mainly relies on the estimation accuracy of $Lat_{pre}(a, D)$ of any device-subnet pairs, which is non-trivial. This is because there are 300+ types of devices and edge servers with different computation capabilities, memory sizes, and various types of neural network operators. Testing all possible subnet over each type of device is infeasible in practice, given the millions of possibilities required to run. To overcome this problem, we design a device knowledge base for the estimation of processing delay.

B. Device Knowledge Base

To provide high accuracy and fast estimations on processing time, we design a device knowledge base. The basic idea of the device knowledge base is to store the operator-level processing time for the devices and estimate the subnet processing delay by accumulating the time of all operators in the subnet. Specifically, the device knowledge base mainly includes four attributes: compute engine, device spec, operator attributes, and processing delay.

C. Compute Engine

The type of the compute engine, i.e., TFLite, PyTorch, and their versions. **Device spec**: device type, e.g., iPhone 15 Pro, hardware configurations, i.e., CPU, GPU, and memory configurations. **Operator attributes**: operator type (e.g., binary operators, convolutional operators, transformer operators), input data type (e.g., fp32/int32), operator-related parameters (for the convolutional operator this includes `filter_h`, `filter_w`, `padding`, `stride_h`, `stride_w`, `fused_activation_function`). **Processing delay**: operator-level processing delay on CPU and GPU.

To test operator delay, the single-operator model was executed over the devices. We use the average running time (by calculating the average single-round time when a running model with 1s warm-up and 5s inference) to imply the operator delay.

Based on the above knowledge base, the running time of the model over CPU can be calculated as follows: Initially, the delay in transferring the model for implementation is calculated. Subsequently, the operators employed within the model are identified, and the time consumed by each operator is determined by referencing the knowledge base. These single-operator delays are then aggregated to estimate the running time of the model.

For the case running over GPU, the model inference time is not simply accumulating the running time of the included operators. Instead, it consists of CPU instruction in-queue time, CPU to GPU, and GPU inference.

IX. Evaluation Test

In this section, we evaluate the performance of the proposed LEOEdge. We first provide the settings of our test and then investigate the results of the evaluation.

A. Settings

We consider a LEO constellation consisting of 10 LEO satellites. The orbit height of these satellites ranges from $500km$ to $700km$, the communication time between each satellite and GS ranges from 92.75 to 104.56 minutes. The satellite-ground data rate is set at 10Mbps based on our observation from the satellite experiment system deployed at the Miyun GS. The link capacity between satellites is 100Mbps. To simulate the heterogenous computation capability on LEO constellations, the computation units on satellites include CPU, GPU, and NPU. We select these processors from various mainstream ARM architecture processors including the Kirin and Snapdragon series, among others. The ground-edge server clusters consist of 3 computing servers and each server has 4 containers to process the received inference tasks. Each container uses a 4 cores x86 CPU with 4Ghz frequency.

For the AI inference, we apply two networks `MobileNetV3` [32] and `ShuffleNetV2` [33]. `MobilenetV3` is a lightweight deep neural network architecture optimized for mobile and embedded vision applications. `ShufflenetV2` is a highly efficient CNN architecture designed for mobile and embedded devices with limited computational resources. As an evolution of the original ShuffleNet, `ShuffleNetV2` focuses on improving both speed and accuracy while minimizing memory and computational usage.

B. Adaptive Modeling Results

The main purpose of adaptive modeling is to select a model with a suitable size that can be executed within the time constraints while also ensuring accuracy. We mainly test the execution time and accuracy of `MobileNetV3` and `ShuffleNetV2` with and without the use of adaptive modeling. Table II shows the results of `MobileNetV3`. By our data, the original accuracy of the `MobileNetV3` is 72%. When applying the adaptive modeling, we set the delay constraint is set to $[15, 20]ms$. The results show that the adaptive modeling controls all generated models' latency within the target range and ensures the accuracy close to the original model. In several cases, adaptive modeling can even achieve better inference accuracy. Especially, the generated models running over new version processors such as Snapdragon 8 Gen 1, Google Tensor, and Kirin 900. This is

TABLE II
MOBILENETV3 PERFORMANCE

Processor	Latency (ms) w/o	Latency (ms) w	ACC (%)
Snapdragon 8 Gen 1	24.9	18.0	74.02
Snapdragon 835	36.8	19.8	71.16
Snapdragon 855	33.0	18.0	71.02
Kirin 980	35.7	19.2	72.96
Kirin 990	28.2	18.9	73.02
Google Tensor	22.1	13.8	76.82
Kirin 9000	22.2	13.8	75.24
Snapdragon 695	37.5	14.7	71.34

TABLE III
SHUFFLENETV2 PERFORMANCE

Processor	Latency (ms) w/o	Latency (ms) w	ACC (%)
Snapdragon 8 Gen 1	20.1	18.2	69.23
Snapdragon 835	34.2	19.2	68.25
Snapdragon 855	31.3	18.4	69.13
Kirin 980	35.0	19.1	68.84
Kirin 990	26.1	18.3	69.33
Google Tensor	23.2	18.5	68.92
Kirin 9000	20.9	14.5	71.91
Snapdragon 695	33.5	14.2	69.01

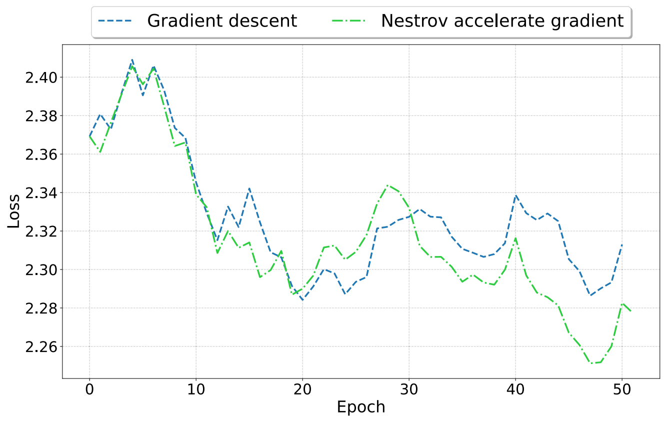


Fig. 6. Loss variants comparison: gradient descent vs. Nestrov accelerate gradient.

mainly because these processors are optimized for handling the AI computations. Similar results can be observed in the case of using `ShuffleNetV2`, which is shown in Table III. The original accuracy of the `ShuffleNetV2` is 69.1% and time constraint for the time execution is set to $[15, 20]ms$. The adaptive modeling effectively reduced the execution model's runtime on these performance-constrained processors, i.e., for the Snapdragon 835, the execution time was reduced from $34.2ms$ to $19.2ms$. Furthermore, the accuracy of the pruned model remains around 69%, ensuring that the model's inference performance is maintained while eliminating unnecessary computational overhead.

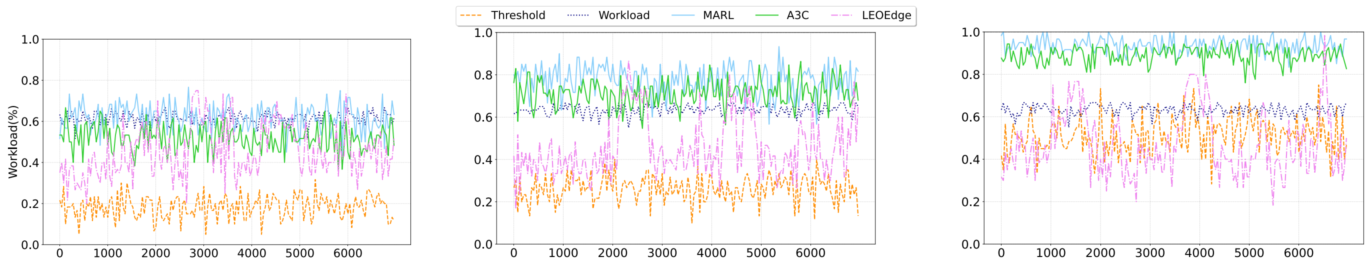
We also test the performance of the search algorithm in terms of convergence, we compare the NAG with the conventional gradient descent used in NAS. As shown in Fig. 6, NAG accelerates convergence to the optimal solution, as the loss value is lower than that of the gradient descent during the training. NAG predicts the next position based on the current momentum term (i.e., the direction and magnitude of the previous gradients), and then calculates the gradient at this anticipated position rather than at the current location. This foresight allows NAG to adjust its parameter updates more smoothly, avoiding excessive corrections, and thereby accelerating the learning process.

C. Task Offloading Results

We compare the results of offloading scheme in LEOEdge with three different offloading logic: (1) Threshold-based scheme offloads all AI inference computations with an estimated execution time higher than a given threshold. In our experiments, tasks with a single inference time exceeding 20ms are offloaded to the edge servers at the GS; (2) Workload-based scheme offloads all inference computations to the edge servers before reaching the given workload constraints. By observing the actual operation of the system, we find that when the CPU utilization remains below 50%, the execution time of the model is not significantly impacted. Consequently, offloading strategies need to aim to keep the average CPU load under 50% as much as possible. Hence, when the average CPU occupancy is lower than 50%, the satellite applying workload-based scheme performs task offloading to the edge servers; (3) Multi-agent reinforcement learning (MARL) method [34] applies collaborative training of multiple learning agents to output the task offloading scheme. Each satellite acts as an agent in a MARL setup, independently deciding whether to offload the generated tasks to an edge server based on changes in the environment. (4) Asynchronous advantage actor-critic (A3C) utilises each individual satellite as an agent to independently explore and learn task loading during training. Our investigation primarily focuses on several key aspects: variations in workload and execution time. We also investigate the communication overhead of LEOEdge.

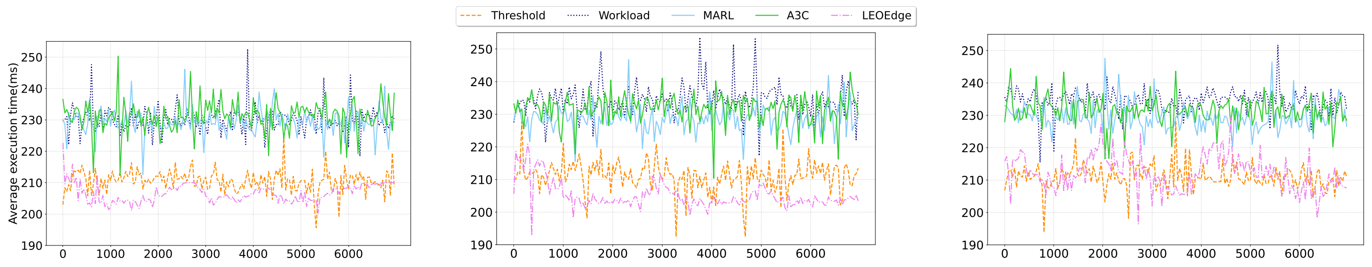
1) Workload Results: The workload of the edge server is an important metric to measure the task offloading performance. A lower workload on the edge server sides indicates the edge computation resources are not fully utilized. On the other hand, a higher workload means that most of the processing units on the edge server are busy and cannot timely the requests. We consider three different task arrival rates in our experiments: (1) Light, where each satellite randomly generates tasks based on the task arrival rate from the dataset during the 21:00-00:00 period. During this time, the task volume is relatively low, with each node generating approximately one task every 5-10 seconds; (2) Normal, where each satellite randomly generates tasks based on the task arrival rate from the dataset during the 14:00-18:00 period. In this timeframe, each node generates roughly one inference task every 2 seconds; (3) Heavy, where each satellite randomly generates tasks based on the task arrival rate from the dataset during the 18:00-20:00 period. This period experiences frequent requests, with each node generating approximately one inference task every second.

Fig. 7 presents the variation of workload over time, displayed in seconds, for four different task offloading schemes. The threshold-based scheme has the lowest workload, as the



(a) light (b) normal (c) heavy

Fig. 7. Edge server workload comparison.



(a) light (b) normal (c) heavy

Fig. 8. AI inference latency comparison.

corresponding blue curve is about 0.2, 0.3, and 0.4 during the simulation in the case of light, normal, and heavy, respectively. The average CPU utilization below 20% implies that the majority of CPUs are in an idle state and are not being fully utilized. Given that user requests arrive randomly, scheduling based on a fixed threshold struggles to adapt to the dynamic changes of the network. Furthermore, a higher threshold value may lead to excessive CPU utilization, which can adversely affect the efficiency of task execution. The edge server load for the other three strategies are controlled within a relatively reasonable range, i.e., around an average CPU utilization of 50%. This ensures the full utilization of edge server resources while also being able to cope with burst arrival requests. However, when in the heavy task arrival case, the edge server load approaches 100% using the MARL-based method, placing a significant burden on the edge servers. A3C is only a slight improvement over MARL. LEOEdge employs a probabilistic constraint to limit the load, the results still closely resemble a hard value approach, akin to the workload-based scheme, thereby validating the effectiveness of such a method. The workload-based method forcibly limits the system's load to 50%, maintaining a load rate of around 50% even under high task arrival rates, but this is achieved at the expense of compromising AI inference performance.

2) Average Execution Time: Since the primary purpose of introducing edge computing is to offload the inference model execution from satellites with unacceptably high execution model time to powerful edge servers, the system's average execution time is extremely important when assessing offloading performance. The shorter average execution time means that the nodes can process data faster, which is especially important for real-time or near-real-time application scenarios. The average execution time is calculated by aggregating the execution times of all AI inference tasks within this period and then taking the average. Fig. 8 shows the average execution times of four offloading schemes across three case of task arrival. The purpose curve corresponding to the case without applying edge serves as a benchmark in the test. As illustrated in the graph, all curves are lower than the case without edge computing, showing that introducing edge computing can indeed reduce the model execution time. We observe that LEOEdge achieves the shortest average task processing time across all three cases. This advantage stems from LEOEdge's hierarchical distributed optimization algorithm, which dynamically schedules tasks based on the load condition of each layer to closely approximate the optimization goal. The threshold-based scheme steadily maintains the execution time around 200 milliseconds, benefiting from the fact that all high-latency tasks are scheduled to be computed at the edge, thereby ensuring consistent execution times. The other two strategies result in a higher execution time. The workload-based scheme does not take into account the local execution time of tasks when scheduling. This leads to situations where tasks that would have a short execution time on the smartphone end up being allocated to edge servers, where they take longer to execute. A possible explanation for the poor performance of the reinforcement learning method is that the model currently used may not accurately capture the dynamics of the system, leading to a failure to learn the optimal strategy. Considering the system variations, training a learning model that can exactly capture the system dynamic and output the optimal offloading strategy is difficult. Additionally, since A3C operates through separate environment interactions for each agent, it tends to be less efficient in terms of execution time compared to MARL approaches.

3) Communication Overhead: One of the primary motivations for proposing LEOEdge is to support AI inference capabilities in mobile apps with a very large user base. Therefore, we investigate the communication overhead (measured by the number of workload information messages sent)

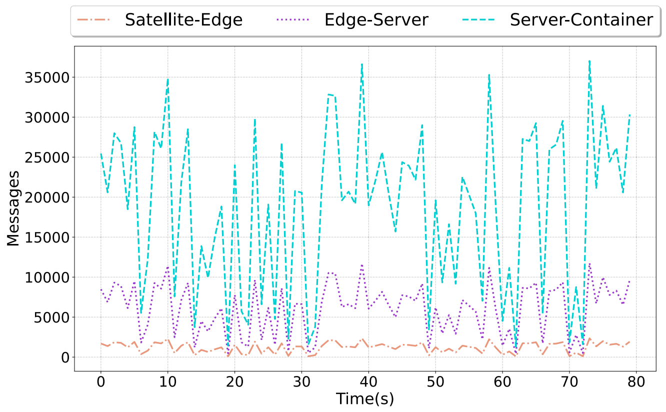


Fig. 9. Communication overhead.

when applying LEOEdge. Fig. 9 shows the communication overhead introduced by different layers of optimization: the communication overhead between satellites and LBS, between LBS and servers, between servers and containers, respectively. We can see that the communication overhead between servers and containers is the highest. This is due to the large number of containers, requiring the server to frequently interact with these containers to determine task allocation based on the load condition of each container. In contrast, the communication overhead between servers and LBS, as well as between LBS and satellites, is relatively low. Especially, the communication overhead between LBS and satellites is mostly under 1000 and linearly related to the task volume. In the satellite-to-ground scenario, since the bandwidth of the space-to-ground link is extremely valuable, the main communication overhead comes from the communication between satellites and ground stations. Compared to this, the internal communication overhead within the server cluster is almost negligible. Therefore, even though the communication overhead between servers and containers is higher, it does not significantly impact the system. Thus, LEOEdge can effectively reduce communication overhead while ensuring scheduling performance.

X. Conclusion

To support high-performance AI inference within LEO satellite constellation, LEOEdge consists of an adaptive modeling scheme to dynamically select the inference model for satellites with different computation capabilities; a distributed task offloading scheme based on our formulated layered optimization to select an optimal place to execute the AI inference tasks; a data seamless transmission scheme to predict the satellite movement and select the transmission path ensuring the data successfully delivered. Evaluation of LEOEdge has shown how our proposed schemes accelerate the AI inference for smartphones, and demonstrated the necessity and superiority of included technologies. In the future, we plan and consider the implementation of LEOEdge.

References

[1] P. Yue et al., "Low Earth orbit satellite security and reliability: Issues, solutions, and the road ahead," *IEEE Commun. Surveys Tuts.*, vol. 25, no. 3, pp. 1604–1652, 3rd Quart., 2023.

[2] Spacex. (2023). *Second Generation Starlink Satellites*. [Online]. Available: https://api.starlink.com/public-files/Gen2StarlinkSatellites.pdf

[3] G. Furano et al., "Towards the use of artificial intelligence on the edge in space systems: Challenges and opportunities," *IEEE Aerosp. Electron. Syst. Mag.*, vol. 35, no. 12, pp. 44–56, Dec. 2020.

[4] B. Shang, Y. Yi, and L. Liu, "Computing over space-air-ground integrated networks: Challenges and opportunities," *IEEE Netw.*, vol. 35, no. 4, pp. 302–309, Jul. 2021.

[5] C. Qiu, H. Yao, F. R. Yu, F. Xu, and C. Zhao, "Deep Q-learning aided networking, caching, and computing resources allocation in software-defined satellite-terrestrial networks," *IEEE Trans. Veh. Technol.*, vol. 68, no. 6, pp. 5871–5883, Jun. 2019.

[6] D. Zhou, M. Sheng, Y. Wang, J. Li, and Z. Han, "Machine learning-based resource allocation in satellite networks supporting Internet of Remote Things," *IEEE Trans. Wireless Commun.*, vol. 20, no. 10, pp. 6606–6621, Oct. 2021.

[7] X. Gao, R. Liu, and A. Kaushik, "Virtual network function placement in satellite edge computing with a potential game approach," *IEEE Trans. Netw. Service Manag.*, vol. 19, no. 2, pp. 1243–1259, Jun. 2022.

[8] Q. Tang, Z. Fei, B. Li, and Z. Han, "Computation offloading in LEO satellite networks with hybrid cloud and edge computing," *IEEE Internet Things J.*, vol. 8, no. 11, pp. 9164–9176, Jun. 2021.

[9] C. Ding, J.-B. Wang, H. Zhang, M. Lin, and G. Y. Li, "Joint optimization of transmission and computation resources for satellite and high altitude platform assisted edge computing," *IEEE Trans. Wireless Commun.*, vol. 21, no. 2, pp. 1362–1377, Feb. 2022.

[10] M. J. Veyette et al., "AI/ML for mission processing onboard satellites," in *Proc. AIAA SCITECH Forum*, Jan. 2022, p. 1472.

[11] S. Yu, X. Gong, Q. Shi, X. Wang, and X. Chen, "EC-SAGINs: Edge-computing-enhanced space–air–ground-integrated networks for Internet of vehicles," *IEEE Internet Things J.*, vol. 9, no. 8, pp. 5742–5754, Apr. 2022.

[12] M. Azami, N. Orger, V. Schulz, T. Oshiro, and M. Cho, "Earth observation mission of a 6U CubeSat with a 5-meter resolution for wildfire image classification using convolution neural network approach," *Remote Sens.*, vol. 14, no. 8, p. 1874, Apr. 2022.

[13] M. Elmahallawy and T. Luo, "AsyncFLEO: Asynchronous federated learning for LEO satellite constellations with high-altitude platforms," in *Proc. IEEE Int. Conf. Big Data*, Dec. 2022, pp. 5478–5487.

[14] M. Elmahallawy and T. Luo, "Optimizing federated learning in LEO satellite constellations via intra-plane model propagation and sink satellite scheduling," in *Proc. IEEE Int. Conf. Commun. (ICC)*, May/Jun. 2023, pp. 3444–3449.

[15] G. Giuffrida et al., "CloudScout: A deep neural network for on-board cloud detection on hyperspectral images," *Remote Sens.*, vol. 12, no. 14, p. 2205, 2020.

[16] N. Razmi, B. Matthiesen, A. Dekorsy, and P. Popovski, "On-board federated learning for dense LEO constellations," in *Proc. IEEE Int. Conf. Commun.*, May 2022, pp. 4715–4720.

[17] Z. Qin, H. Yao, T. Mai, D. Wu, N. Zhang, and S. Guo, "Multi-agent reinforcement learning aided computation offloading in aerial computing for the Internet-of-Things," *IEEE Trans. Services Comput.*, vol. 1, no. 1, pp. 1–12, Oct. 2022.

[18] L. Cheng, G. Feng, Y. Sun, M. Liu, and S. Qin, "Dynamic computation offloading in satellite edge computing," in *Proc. IEEE Int. Conf. Commun.*, May 2022, pp. 4721–4726.

[19] X. Cao et al., "Edge-assisted multi-layer offloading optimization of LEO satellite-terrestrial integrated networks," *IEEE J. Sel. Areas Commun.*, vol. 41, no. 2, pp. 381–398, Feb. 2022.

[20] R. Xie, Q. Tang, Q. Wang, X. Liu, F. R. Yu, and T. Huang, "Satellite-terrestrial integrated edge computing networks: Architecture, challenges, and open issues," *IEEE Netw.*, vol. 34, no. 3, pp. 224–231, May 2020.

[21] Y. Zhang, C. Chen, L. Liu, D. Lan, H. Jiang, and S. Wan, "Aerial edge computing on orbit: A task offloading and allocation scheme," *IEEE Trans. Netw. Sci. Eng.*, vol. 10, no. 1, pp. 275–285, Jan. 2023.

[22] D. Vasisht, J. Shenoy, and R. Chandra, "L2D2: Low latency distributed downlink for LEO satellites," in *Proc. Conf. Appl., Technol., Archit., Protocols Comput. Commun.*, 2021, pp. 151–164.

[23] Z. Lai, H. Li, Q. Zhang, Q. Wu, and J. Wu, "Cooperatively constructing cost-effective content distribution networks upon emerging low earth orbit satellites and clouds," in *Proc. IEEE 29th Int. Conf. Netw. Protocols (ICNP)*, Nov. 2021, pp. 1–12.

[24] V. Bhosale, K. Bhardwaj, and A. Gavrilovska, "Toward loosely coupled orchestration for the LEO satellite edge," in *Proc. 3rd USENIX Workshop Hot Topics Edge Comput.*, 2020, pp. 1–26.

[25] P. McEnroe, S. Wang, and M. Liyanage, "A survey on the convergence of edge computing and AI for UAVs: Opportunities and challenges," *IEEE Internet Things J.*, vol. 9, no. 17, pp. 15435–15459, Sep. 2022.

[26] K. Van Beeck, T. Ophoff, M. Vandersteegen, T. Tuytelaars, D. Scaramuzza, and T. Goedemé, "Real-time embedded computer vision on UAVs: Uavision2020 workshop summary," in *Comput. Vision–ECCV 2020*. Cham, Switzerland: Springer, 2020, pp. 665–674.

[27] A. Menshchikov et al., "Real-time detection of hogweed: UAV platform empowered by deep learning," *IEEE Trans. Comput.*, vol. 70, no. 8, pp. 1175–1188, Aug. 2021.

[28] X. Hu, K.-K. Wong, K. Yang, and Z. Zheng, "UAV-assisted relaying and edge computing: Scheduling and trajectory optimization," *IEEE Trans. Wireless Commun.*, vol. 18, no. 10, pp. 4738–4752, Oct. 2019.

[29] M. A. Qureshi, E. Lagunas, and G. Kaddoum, "Reinforcement learning for link adaptation and channel selection in LEO satellite cognitive communications," *IEEE Commun. Lett.*, vol. 27, no. 3, pp. 951–955, Mar. 2023.

[30] Magister Solutions Ltd. (2014). *Satellite Network Simulator 3*. [Online]. Available: https://www.sns3.org/

[31] C. Jiang and X. Zhu, "Reinforcement learning based capacity management in multi-layer satellite networks," *IEEE Trans. Wireless Commun.*, vol. 19, no. 7, pp. 4685–4699, Jul. 2020.

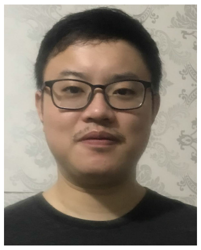
[32] A. Howard et al., "Searching for MobileNetV3," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Oct. 2019, pp. 1314–1324.

[33] N. Ma, X. Zhang, H.-T. Zheng, and J. Sun, "ShuffleNet V2: Practical guidelines for efficient CNN architecture design," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, 2018, pp. 116–131.

[34] J. Lai, H. Liu, Y. Sun, J. Zhu, W. Ma, and L. Gan, "Multi-agent deep reinforcement learning based computation offloading approach for LEO satellite broadband networks," in *Proc. IEEE Symp. Comput. Commun. (ISCC)*, Jul. 2023, pp. 1435–1440.

Su Yao received the Ph.D. degree from the National Engineering Laboratory for Next Generation Internet, Beijing Jiaotong University. Currently, he is an Associate Researcher with Beijing National Research Center for Information Science and Technology (BNRist), Tsinghua University. His research interests include future network architecture and the IoT security.

Yiying Lin received the B.S. degree in computer science and technology from Beijing Forestry University in 2024. He is currently pursuing the master's degree with the State Key Laboratory of Networking and Switching Technology. His research interests include edge computing and multimedia communications.



Mu Wang received the Ph.D. degree in computer technology from Beijing University of Posts and Telecommunications (BUPT) in 2020. He is currently an Associate Researcher with the State Key Laboratory of Network and Switching Technology. His research interests include wireless communications and multimedia communications.

Ke Xu (Fellow, IEEE) received the Ph.D. degree from the Department of Computer Science and Technology, Tsinghua University. He is currently a Full Professor with Tsinghua University. He has published more than 100 technical articles and holds 20 patents in the research areas of next generation internet and P2P systems.

Mingwei Xu (Senior Member, IEEE) received the B.Sc. and Ph.D. degrees from Tsinghua University. He is currently a Full Professor with the Department of Computer Science, Tsinghua University. His research interests include computer network architecture, high-speed router architecture, and network security.

Changqiao Xu (Senior Member, IEEE) is currently a Professor with the State Key Laboratory of Networking and Switching Technology and the Director of the Network Architecture Research Center, BUPT. He has edited two books and published more than 200 papers in prestigious international journals and conferences, including IEEE/ACM TRANSACTIONS ON NETWORKING (IEEE ToN), IEEE TRANSACTIONS ON MOBILE COMPUTING, IEEE INFOCOM, and ACM Multimedia. His research interests include future internet technology, mobile networking, multimedia communications, and network security. He was a number of international conferences and workshops as the co-chair and a TPC member. He is currently the Editor-in-Chief of *Transactions on Emerging Telecommunications Technologies* (Wiley).

Hongke Zhang (Fellow, IEEE) received the Ph.D. degree in communication and information systems from the University of Electronic Science and Technology of China, Chengdu, China, in 1992. He is currently a Professor with the School of Electronic and Information Engineering, Beijing Jiaotong University, Beijing, China, where he currently directs the National Engineering Center of China on Mobile Specialized Network. He is also an Academician of China Engineering Academy, Beijing, and the Co-Director of the PCL Research Center of Networks and Communications, Peng Cheng Laboratory, Shenzhen, China. His current research interests include architecture and protocol design for the future internet and specialized networks. He currently serves as an Associate Editor for IEEE TRANSACTIONS ON NETWORK AND SERVICE MANAGEMENT and IEEE INTERNET OF THINGS JOURNAL.