Reliable or Green? Continual Individualized Inference Provisioning in Fabric Metaverse via Multi-exit Acceleration

Yu Qiu, Min Chen, *Fellow, IEEE*, Weifa Liang, *Senior Member, IEEE*, Dusit Niyato, *Fellow, IEEE*, Yue Wang, Yizhe Li, Victor C.M. Leung, *Life Fellow, IEEE*, Yixue Hao, Long Hu, Yin Zhang

Abstract—Fabric metaverse employs intelligence fibers embedded with flexible sensors to unknowingly gather and transmit massive hypermodal data around humans to a deep neural network-based metaverse inference service (DMS) for continual and real-time analysis. Each DMS has one primary branch and multiple side branches that allow early termination of service with differential accuracy and energy consumption. However, the continual provisioning of compute-intensive DMS with varying requirements for service model, accuracy, delay, and reliability poses a challenge for edge servers characterized by restricted computing resources and intermittent green energy. In this paper, we focus on a continual individualized DMS provisioning problem in the fabric metaverse consisting of a side branch insertion subproblem and a server activation and service deployment subproblem, and formulate them as Integer linear Programming and Markov Decision Process, respectively. Then, we propose a green continual inference (GCI) system, where a pruner with provable approximation ratios trims superfluous branches of every model to the given number K to minimize total overflow accuracy between accuracy demands and reserved branches assigned to users. Based on this exit result, each DMS is further divided into several blocks with dependencies to exploit constrained resources of computing and energy in a fine-grained manner. Finally, a learning-based scheduler is merged into GCI to maximize request throughput while minimizing the activation number of edge servers on different demand scenarios, by adaptively activating suitable servers and deploying required blocks and their corresponding backups on selected servers. Theoretical analyses, simulations, and experiments demonstrate that the GCI is promising compared with baseline algorithms.

Index Terms—Metaverse, Edge intelligence, Fabric computing, 6G, Inference, Reliability, Green, Accuracy, Early exit point.

1 Introduction

The fabric metaverse, composed of intelligence fibers embedded with flexible sensors, edge servers, and a remote cloud, heralds a new era of human-centered networks to boost service performances directly from data sources and interactions [1]. Compared to shape-fixed traditional sensors, intelligence fibers are woven into various fabrics (e.g., clothes, hats, and cushions) to mitigate the resistance emotion from users and successfully lurk around them [2], [3]. Leveraging by this characteristic, intelligence fibers can unknowingly collect and transmit hypermodal data to personalized deep neural network-enabled metaverse inference services (DMSs) hosted on servers via 6G networks for continuous and real-time data analysis. To reduce delay and energy consumption during uploading, the fabric metaverse integrates mobile edge computing and renewable energy harvesting technologies to place rechargeable edge servers around users. This integration not only mitigates waiting times and single-point of failures caused by high workloads on sensory devices and cloud centers in scenarios with numerous users [4], [5], [6], [7], [8], [9], but also addresses concerns regarding carbon emissions and the greenhouse effect resulting from the increased carbon emissions and intensity associated with cloud centers and wide-area networks [10].

Model accuracy emerges as a pivotal metric guiding personalized DMS provisioning in the fabric metaverse. Different services typically require heterogeneous accuracy [11], such as the requirements for computer vision fall between 66.4% to 83.5% [12], whereas ones are between 74.1% to 81.2% [13] for natural language processing. Moreover, within a single service type, such as fault detection of AR equipment, bearing data acquisition sensors need 80% accuracy to identify facility faults, while low-bearing data acquisition sensors must attain 90% accuracy to complete the same effect [14]. Consequently, according to the distribution of accuracy requirements for all users, each computation-intensive DMS is carefully structured with a high-accuracy backbone and selected lower-accuracy sub-

- *Corresponding author: Min Chen (email: minchen@ieee.org).*
- *Yu Qiu, Min Chen, Yue Wang, and Yizhe Li are with the School of Computer Science and Engineering, South China University of Technology, Guangzhou, P. R. China (email: csqiuyu@mail.scut.edu.cn, g4r3nwang@gmail.com, and hyoi_cmath@outlook.com).*
- *Weifa Liang is with the Department of Computer Science, City University of Hong Kong, Hong Kong, P. R. China (email: weifa.liang@cityu.edu.hk).*
- *Dusit Niyato is with the College of Computing & Data Science (CCDS), Nanyang Technological University, Singapore (email: dniyato@ntu.edu.sg).*
- *V. C. M. Leung is with the College of Computer Science and Software Engineering, Shenzhen University, Shenzhen 518060, China, and also with the Department of Electrical and Computer Engineering, the University of British Columbia, Vancouver, BC V6T 1Z4, Canada (email: vleung@ieee.org).*
- *Yixue Hao and Long Hu with the School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan, P. R. China (email: yixuehao@hust.edu.cn and hulong@hust.edu.cn).*
- *Yin Zhang is with the School of Information and Communication Engineering, University of Electronic Science and Technology of China, Chengdu, P. R. China (email: yin.zhang.cn@ieee.org).*

branches that support accelerating inference processes. This model structure allows each service to be further divided into fine-grained blocks with dependencies but requiring fewer computing resources and less energy. In addition, the impact of primary block failures due to hardware or software issues can be mitigated by providing corresponding fine-grained backups. These blocks are distributed across multiple edge servers powered by renewable energy and a cloud center fueled by fossil fuels, and a confident prediction is agilely returned from a certain subbranch running on a nearby server or the final subbranch on the cloud.

However, the edge-cloud cooperative inference scheme in fabric metaverse networks needs to tackle the following challenges. (1) The side branches need to be inserted and trained at all feasible positions of each DMS with heterogeneous structure in advance, which increases the energy consumption and computation of the model itself due to additional parameters from these intermediate classifiers. (2) Hypermodal data gathered by intelligence fibers with extremely weak computing capacities must be continuously offloaded to DMSs hosted on nearby edge servers, yet the data of some users may be discarded or not processed when serving massive users due to the limited computing resources of servers. (3) To enjoy a low-latency service experience, users desire requested DMS services to be offloaded to edge servers as close to them as possible. However, in some cases, the majority of users move to a few specific regions, resulting in intensifying energy competition for edge servers. Thus, high competition among services for intermittent and limited renewable energy of advantageous edge servers can compromise user experiences and even cause service determination and server failures due to imbalanced workloads. In the worst case where backups are deployed to improve service reliability, the energy competition is further intensified due to the additional consumption of backup monitoring energy. Thus, how to optimally prune redundancy branches of overdecorated and heterogeneous DMSs to the given number K to minimize accuracy overflow, and distribute pruned DMSs across selected edge servers to maximize service throughput while minimizing the number of activated servers, subject to user demands on delay, reliability, and accuracy and server restrictions on compute and energy resources, is a challenging issue.

The novelty of this study is to explore the continual individualized DMS provisioning in a fabric metaverse (or device-user-service-network) for the first time. A green continual inference provisioning system (GCI) was developed, as shown in Fig. 1. And a performance-guaranteed algorithm for the problem built on the system is then designed, where intelligence fibers collect hypermodal data from humans and environments without being perceived, and continuously transmit it to pruned DMS services hosted on activated servers, so as to achieve the goal of processing as many DMS requests as possible with the least activated servers, while meeting various constraints in real-time.

The main contributions of this paper are listed as follows.

- We formulate an Integer Linear Programming, a Markov Decision Process for a side branch insertion subproblem, and a server activation and service deployment subproblem in DMS provisioning. We also show the NP-hardness of the defined subproblems.
- To reduce resource consumption and improve inference speed of the model itself by pruning superfluous branches to a given K, we develop an exact pruner for a small problem size with high requirements. Otherwise, we devise an approximate pruner with a provable approximation ratio for a large problem size, which achieves near-optimal performance to compress overdecorated DNN models.
- To maximize service throughput while minimizing the number of servers, we devise a scheduler with two symbiotic components: a learning-based activator specifies deployment scope by adaptively activating a subset of servers based on real-time users, model, and network information. Concurrently, for each DMS with a differential delay sensitivity, we develop two deployers that are based on an approximation algorithm and a heuristic algorithm respectively, by striking for non-trivial trades-off between different user demands and resource limitations, near-optimally placing DNN blocks and their backups to activated servers to provide training rewards to the activator.
- To evaluate the performance of the proposed algorithms, we conduct extensive simulations, which reveal that the throughput delivered by the GCI system is from 1.1 to 1.6 times of DMS requests but consumes from 7% to 22% less green energy compared to the baseline algorithms while approaching the optimal solution. Additionally, we build a hardware prototype to provide a realistic application scenario.

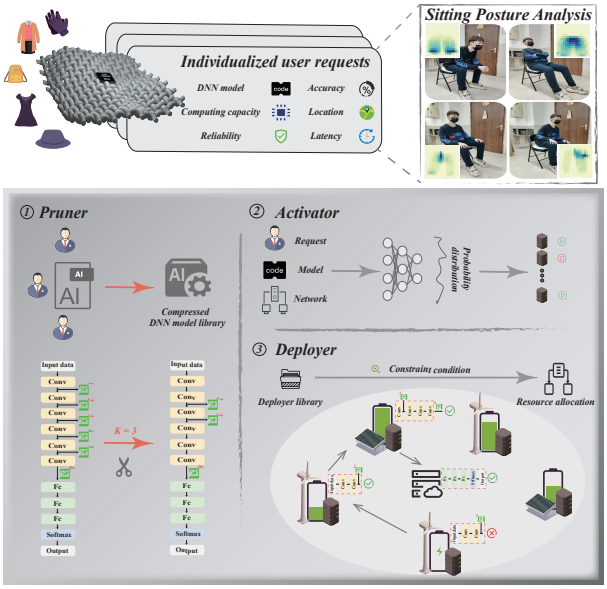


Fig. 1: The GCI system for fabric metaverse network

The rest of this article is organized as follows. Section 2 reviews related works and analyzes their limitations. Section 3 introduces the fabric metaverse network, performance metric, and other mathematical models. Section 4 introduces the GCI system and shows mathematical analyses of algorithms in the system. Section 5 evaluates the performances of our algorithms. Section 6 summarizes the paper.

2 Related Work

Intelligence fabrics continuously collect and transmit massive hypermodal data about humans and environments [1], [15], providing a stable data foundation for building various data-hungry virtual worlds supported by DMS in the fabric metaverse. In this section, we introduce relevant key technologies during DMS provisioning to further evolve it towards immersive, reliable, and green.

Multi-exit acceleration technology: Kaya *et al.* [16] formulated an overthinking problem and designed a deep shallow neural network architecture with internal classifiers, namely side branches, to alleviate it. Laskaridis *et al.* [17] proposed a hardware-aware progressive inference system to insert side branches under stringent delay and hardware limitations for maximizing service accuracy. Jo *et al.* [18] constructed a low-cost early exit network that combines dynamic branch pruning and novel branch structures to reduce the energy consumption of side branches, and developed hardware architectures to support the operations. Hu *et al.* [19] presented a layer-aware approximation algorithm to minimize the progressive inference delay of models with acyclic graph structure models across edge devices.

Failure resistance technique: Inspired by residual connection, Kaya *et al.* [20] designed a fault-resistant distributed neural network with inserted several extended layers and hyperconnection layers connecting different physical nodes. By skipping the part of neural networks running on failure nodes, the continual and reliable transmissions of intermediate parameters are ensured at the cost of some accuracy. In addition, Huang *et al.* [21] designed a hybrid elastic inference algorithm to best-effort ensure service accuracy under random interruption time by dynamically saving intermediate results and skipping some failure branches. Laskaridis *et al.* [22] utilizes multiple exit points to calculate low-precision rollback results for best-effort to meet user demands, where each model with graph structure is divided into two inference parts on dynamic edge cloud environments.

Energy harvesting technology: Mao *et al.* [23] surveyed energy optimization and management problems in different green communication networks from AI and energy harvesting perspectives, and made a detailed summary and comparison. Zhu *et al.* [24] proposed a base station sleeping algorithm based on a recurrent neural network to predict user traffic distribution that guides the opening or closing of base stations to save energy. Shen *et al.* [25] designed a two-timescale algorithm based on stochastic optimization to control the active or sleep of base stations for minimizing the purchase cost of power gird, renewable generator, and energy switching. Ma *et al.* [10] studied inference offloading and carbon emission right purchasing problem, and established an online algorithm based on the Lyapunov optimization technique to minimize accuracy loss under a given cost budget. Gu *et al.* [26] designed a model-free deep reinforcement learning algorithm to solve a service management and energy scheduling problem for minimizing the long-term energy cost.

Unlike the studies in [16], [24], [18], [19] that only considered single mode or hardware limitations during multi-exit acceleration, the GCI considers multi-user and imperfect dynamic network limitations. In contrast to studies [20], [21], [22] where intermediate results are returned by previous side branches when a fault occurs to make the best effort to recover accuracy, the GCI adopts a redundant backup method to return satisfactory results by processing data to the specified branch. In addition, existing works [23], [24], [25], [10], [26] ignored backup monitoring energy consumption and intermittent energy competition among users, and they cannot be applied to the continual individualized DMS provisioning problem in the fabric metaverse directly.

3 Preliminaries

In this section, we introduce the system model, notions and notations, and the problem definition precisely.

TABLE 1: Notations

Notation	Definition
G	The fabric metaverse network
C and V	The cloud center and the set of edge servers
Cap_j	The thread capacity of an edge server v_j
μ	The floating point operations rate of a thread
E	The set of link e connecting different servers
b_e and d_e	The bandwidth and distance of link e
$\mathcal{M}$	The set of DNN models required by metaverse services
P_m	The set of early exit points of an overdecorated DNN model m_j
$\hat{P}_{\hat{m}}$	The set of early exit points of a pruned DNN model $\hat{m}_j$
γ_i	The DMS inference request
η_i	The number of threads required by γ_i
f_{s_i}	The data sampling rate of inference request γ_i
σ_i	The the confidence threshold of a request γ_i
$FLOP_{b_k}$	The floating point operations for the kth block in γ_i
d_{b_k}	The size of input data for kth block in γ_i
$r_{i,k}$	The reliability of the kth block in request γ_i
$D_{t,i}$	The transmission delay of an inference request γ_i
$D_{p,i}$	The propagation delay of an inference request γ_i
$D_{c,i}$	The calculation delay of an inference request γ_i
E_j^{green}	The total available green energy of edge server v_j
E_j^b	The available energy of battery in edge server v_j
E_j^{cm}	The consumed communication energy of edge server v_j
E_j^{cp}	The consumed computation energy of edge server v_j
E_j^{bu}	The consumed backup monitoring energy of edge server v_j
$\mathscr{R}$	The reward function in SASD problem modeled by MDP
$g_{i,k}$	Whether early exit point p_k is allocated to request γ_i
h_k	Whether early exit point p_k is reserved in a model
$x_{i,k,j}$	Whether the block b_k in γ_i is deployed on a edge server v_j
$w_{v,u}^{i,k,k+1}$	Whether $\mathscr{P}_{v,u}$ transmit data from block b_k to b_{k+1}
f_i	Whether all demands of a request γ_i are met
z_j	Whether an edge server v_j is activated

3.1 Fabric Metaverse Network

The system considers a finite time horizon $\mathbb{T}$ with equal time slots. We model a fabric metaverse network as an undirected weighted graph $G = (\{v_0\} \cup V, E)$, which consists of a cloud data center v_0, the set $V = \{v_1, \ldots, v_n\}$ of edge servers, and the set E of links with each link $e \in E$ having length d_e and bandwidth b_e. The edge server $v_j \in V$ co-located with an access point (AP) has computing capacity Cap_j, which refers to the number of threads with a floating point operation rate of μ. We assume that the cloud center is powered by fossil fuels with ample resources, while the edge server powered by a renewable energy collector has a battery capacity of E^b. The energy collector uses solar photovoltaic panels and wind turbines to convert solar and wind energy, respectively, generated at each time slot $t \in \mathbb{T}$ into electricity and stores it in available battery E_j^b. Typically, the solar energy $E_j^s(t)$ and wind energy $E_j^w(t)$ are dynamic,

intermittent, and affected by local environments, such as time-varying solar radiation $Q_j(t)$ and wind speed $s_j(t)$ per slot [27], [28]. The total amount of green energy $E_j^{green}(t)$ at each time slot t thus is

$$E_j^{green}(t) = \min\{E_j^s(t) + E_j^w(t) + E_j^b(t-1), E^b\}, \quad (1)$$

$$E_j^s(t) = Q_j(t) \cdot S_{panel} \cdot \varphi, \quad (2)$$

$$E_j^w(t) = 1/2 \cdot \rho \cdot S_{turbine} \cdot s_j(t), \quad (3)$$

where S_{panel} and $S_{turbine}$ are receptor areas of energy collector devices, respectively. φ represents photoelectric conversion efficiency, and ρ indicates the air density.

3.2 Deep Neural Network-based Metaverse Inference Service Request

The metaverse services based on various DNN models provide real-time and continual individualized inference by processing hypermodal data from intelligence fibers worn by humans. Each user issues individualized DMS request $\gamma_i(t) = (M_i(t), A_i(t), R_i(t), D_i(t), L_i(t))$ with diverse requirements including service model $M_i(t)$, accuracy $A_i(t)$, reliability $R_i(t)$, delay $D_i(t)$, and user location $L_i(t)$. Note that each user moves once randomly at begin of the time slot $t \in \mathbb{T}$ and generates a new DMS request. Let $\mathcal{R}(t)$ be the set of DMSs issued by all users wearing intelligence fibers at slot t.

There are $\mathcal{M}$ types of DMSs in the fabric metaverse network, such as sitting posture analysis, human activity recognition, and sleep stage classification. To end service earlier with confident accuracy A_i, an intermediate classifier, consisting of fully connected layers and a softmax layer, is inserted between adjacent convolution layers of each model as early exit points. Intuitively, these exit points are natural partition points, because each partition not only returns an intermediate result without supernumerary energy and delay due to subsequent calculations, but also gathers as many layers with a complex graph structure into one block as possible. Thus, every DNN model $m \in \mathcal{M}$ even with a directed acyclic graph structure can be simplified and modeled as a block-aware linear structure, where the vertex represents a block with part of the primary branch and a side branch, and the link indicates a logical relationship between any two adjacent blocks. Note that adding these intermediate classifiers does not affect the structures of original DMSs, and even improves service accuracy due to additional parameters [17].

The hypermodal data d_{b_1} related to human bodies and their surroundings is collected by intelligence fiber with sample rate f_s, and transmitted to edge servers as the input to the first block of each DMS m. Then, for each block b_k $(k \leq K)$, the data has to execute floating point operations $flop_{b_k} = flop_{m_k} + flop_{p_k}$ from the primary branch m_k and an exit point p_k to obtain a prediction with varying accuracy a_{p_k} and intermediate parameter with data size $d_{b_{k+1}}$. K is a given threshold that represents the maximum number of branches (exit points).

3.3 Metrics of User Service Experience

The below metrics are used to measure DMS service qualities: accuracy, reliability, energy consumption, and delay.

Accuracy: The sampling rate f_s of intelligence fibers and the confidence threshold σ jointly determine inference accuracy, and are constrained by accuracy requirements $A_i = \sigma_i \cdot (A_{M_i}^{max} - A_{M_i}^{min})$. Specifically, let $\mathscr{F}$ be the set of available sample rates, and the quality and quantity of collected data gradually increase with the improvement of f_s. The optimal sampling rate $f_{s_i}^*$ under different models has been calculated in advance from offline data sets by our previous algorithm [1], which is regarded as known information. The maximum probability a_{p_k} of all prediction types of an early exit point is regarded as the confidence criterion. That is, if an intermediate accuracy a_{p_k} is not lower than the confidence threshold σ_i of a user, then the prediction process exits early, and otherwise continues to next one in the group $P_m = \{p_{i,k} | 0 < k \leq K\}$ of candidate branches. Note that the early exit point $p_{i,k'}$ assigned to request γ_i can meet the accuracy demand, while the subsequent part of the DNN model in the cloud is usually idle and executed to obtain an entire result when the user is not satisfied with the intermediate result at this time.

Reliability: A DMS can be divided into K blocks with a chain dependency for cooperative execution, according to the threshold K of the early exit point. Each block has a different reliability due to unpredictable failures of software and/or hardware, such as priority preemption, power shortage, or natural disasters. The reliability r indicates the percentage of running time in a normal state to the total detection time, which can be obtained based on historical log traces. Deploying redundant backups is a practical way to improve service reliability. Thus, the enhanced reliability $\widehat{r}_{i,k}$ of the kth block in model M_i with its $q_{i,k}$ backup blocks is defined as follows:

$$\widehat{r}_{i,k} = 1 - (1 - r_{i,k})^{q_{i,k}+1}, \quad (4)$$

where $r_{i,k}$ is the initial reliability of the kth block. We assumed that failure between different blocks occurs independently and does not propagate because these blocks run in a completely isolated virtual environment [29]. Thus, the reliability r_i of whole DMS requested by γ_i that is composed of multiple DNN blocks are given as follows:

$$r_i = \prod_{k=1}^{k'} \widehat{r}_{i,k}. \quad (5)$$

Note that the blocks deployed on the cloud are assumed to be infallible because the cloud has sufficient resources to employ multiple failure prevention measures. Thus, we only consider the reliability of blocks deployed on edge servers with limited resources and energy, namely the part before the early exit point $p_{i,k'}$.

Energy Consumption: Typically, active and asleep are two states of an edge server, which can be dynamically switched based on the number of received users to reduce energy consumption and carbon footprint. In terms of the asleep state, servers and base stations only need to consume a small sleep energy E_j^{sleep} to maintain basic functions, while the collector is still operating and continues to collect and store renewable energy.

In terms of the active state, communication, computation, backup, and leakage energy E^{leak} are considered. For an edge server v_j, the communication energy E_j^{cm} is mainly

used to transmit intermediate data among edge servers,

$$E_j^{cm} = \sum_{\gamma_i \in \mathcal{R}} \sum_{k=1}^{k'} \mathcal{P}^{tra} \cdot \frac{d_{b_{i,k}}}{b_e} \cdot x_{i,k,j}, \tag{6}$$

where $\mathcal{P}^{tra}$ is the transmission power of servers, and $x_{i,k,j}$ is a binary variable representing whether the block b_k required by γ_i is deployed on an edge server v_j. The computation energy is used to perform the float point calculations required by different DMSs for users and expressed as follows:

$$E_j^{cp} = \sum_{\gamma_i \in \mathcal{R}} \sum_{k=1}^{k'} \mathcal{P}^{com} \cdot \eta_i \cdot \frac{flop_{b_{i,k}}}{\eta_i \cdot \mu} \cdot x_{i,k,j}, \tag{7}$$

where $\mathcal{P}^{com}$ is the computation power per thread, and η_i denotes the number of threads required by user request γ_i. The backup monitoring energy E_j^{bu} is provided to the backup block to listen to the status information of the corresponding primary block,

$$E_j^{bu} = \sum_{\gamma_i \in \mathcal{R}} \sum_{k=1}^{k'} \mathcal{P}^{bu} \cdot q_{i,k,j} \cdot x_{i,k,j}, \tag{8}$$

where $q_{i,k,j}$ is the number of backup blocks for kth block of request γ_i in server v_j, and $\mathcal{P}^{bu}$ is backup monitoring power. In addition, the on-site backup scheme is adopted in all backup scenarios in this paper due to its energy-friendly property. Specifically, both the primary and backup blocks are deployed in the same edge server. Compared with the off-site backup scheme, each server in the on-site backup scheme only needs to provide the running energy of the primary block and listening energy of its backups, and it does not need to reserve additional running energy required for these backups, because backups usually remain idle unless the primary block fails. Thus, the total energy consumption E_j^{total} of edge server v_j and its remaining battery $E_j^b(t)$ at time slot t are expressed as follows:

$$E_j^{total} = \begin{cases} E_j^{cm} + E_j^{cp} + E_j^{bu} + E^{leak}, & \text{if } z_j = 1, \\ E^{sleep}, & \text{otherwise}, \end{cases} \tag{9}$$

$$E_j^b(t) = \max\{E_j^{green}(t) - E_j^{total}(t), 0\}, \tag{10}$$

where z_j is a binary variable indicating whether edge server $v_j \in V$ is active or not.

Delay: The inference delay is affected by the early exit point $p_{i,k'}$. The DMS is divided into one part with multiple blocks distributed on edge servers and another part in the cloud data center according to the $p_{i,k'}$. The inference part of a DNN model on edge servers becomes larger as the confidence threshold σ increases, and DMS accuracy and delay become larger. Each DMS provisioning will experience transmission $D_{t,i}$, propagation $D_{p,i}$, and calculation delay $D_{c,i}$, which are defined as follows.

$$D_{t,i} = \frac{(f_{s_i} - 1) \cdot d_{i,b_1} + \sum_{k \le K} d_{i,b_k} \sum_{j \in V} x_{i,k,j}}{b_e}, \tag{11}$$

$$D_{p,i} = \sum_{j \in V} \frac{\mathscr{P}_{i,j} \cdot x_{i,1,j}}{3 \times 10^8} + \sum_{k<K} \sum_{v,u \in V} \frac{\mathscr{P}_{v,u} \cdot w_{v,u}^{i,k,k+1}}{3 \times 10^8}, \tag{12}$$

$$D_{c,i} = \sum_{k \le K} \sum_{j \in V} \frac{flop_{i,b_k}}{\eta_i \cdot \mu} \cdot x_{i,k,j}, \tag{13}$$

where $\mathscr{P}_{v,u}$ represents the shortest path from server v to server u, and $\mathscr{P}_{i,j}$ is the distance from the user to its nearest server plus the shortest distance from the nearest server to a server v_j, and $3 \times 10^8 km/s$ is electromagnetic waves speed. $w_{v,u}^{i,k,k+1} = \max\{x_{i,k,v} + x_{i,k+1,u} - 1, 0\}$, and it is a binary variable indicating whether the shortest path $\mathscr{P}_{v,u}$ is used to transmit intermediate results block b_k and b_{k+1}.

3.4 Problem Definitions

Definition 1. ***The DNN model side branch insertion and allocation problem (DIA):*** *Given an integer $K \in \mathbb{Z}^+$, the set of DNN models $\mathcal{M} = \{m \mid 1 \le m \le |\mathcal{M}|\}$, and the set of inference requests in $\mathcal{R}$ on the models. Each model m has one primary branch and multiple side branches, namely early exit points, $P_m = \{p_k \mid 1 \le k \le |P_m|\}$, and each request $\gamma_i \in \mathcal{R}$ has a minimum accuracy demand A_i and a service model demand M_i to preliminarily immerse in a metaverse service enabled by the DNN model M_i. The DIA is to select respectively one of the K exit points for each overdecorated DNN model $m \in \mathcal{M}$ to get the set $\hat{\mathcal{M}}$ of pruned DNN models $\hat{m} \in \hat{\mathcal{M}}$ with a set of subbranches $\hat{P}_{\hat{m}}(|\hat{P}_{\hat{m}}| = K)$, to minimize overflow accuracy of all users from an accuracy demand to the exit point $p_{i,k'}$ allocated to a user in a fabric metaverse.*

Definition 2. ***The server activation and service deployment problem (SASD):*** *Given a fabric metaverse network $G = (\{v_0\} \cup V, E)$ where each edge server has computing capacity cap_j and renewable energy E_j^{green}. There is the set $\hat{\mathcal{M}}$ of pruned DNN models $\hat{m}$ with selected K exit points $\hat{P}_{\hat{m}}$, and the set $\mathcal{R}$ of inference requests, where each request $\gamma_i \in \mathcal{R}$ has a service model demand M_i with early exit point $p_{i,k'}$, a reliability demand R_i, delay demand D_i, location demand L_i. The SASD problem is to strategically activate a subset of edge servers, and deploy DMS consisting of multiple blocks and its corresponding backup blocks on the set of activated edge servers, to maximize throughput of DMS requests with the minimal number of activated servers while meeting the model, reliability, and delay demands of all DMS requests, subjected to resource capacity and green energy budget on the network.*

Theorem 1. *The DIA problem in a fabric metaverse network is NP-hard.*

Proof. The NP-hardness of DIA is shown by reducing the well-known NP-hard problem - $\mathcal{K}$-median problem to the DIA problem in polynomial time. The $\mathcal{K}$-median problem is defined as follows: Given a weighted undirected graph $\mathcal{G} = (\mathscr{L} \cup \mathscr{U}, \mathscr{E})$ with K facilities, where $\mathscr{L}$ is the set of potential locations $l_j \in \mathscr{L}$ for the facilities, $\mathscr{U}$ is the set of users $u_i \in \mathscr{U}$, and $\mathscr{E}$ is the set of links $e'_{i,j}$ between user i and location j with the Euclidean distance $dis_{i,j}$, the $\mathcal{K}$-median problem is to locate K facilities into K locations in $\mathscr{L}$ so that the sum of distances between all users and their nearest facilities is minimized.

We show that an instance of the $\mathcal{K}$-median problem can be reduced to a special case of the DIA problem. Assume that there is only a DNN model with $|P_m| \ge K$ insertable positions of side branches, and all inference requests in $\mathcal{R}$ are issued at the same time. Given a branch threshold K, the DIA problem is to insert K side branches into K available positions to form a pruned DNN model $\hat{m}$ with an exit point

set $\hat{P}_{\hat{m}}$, namely $K = |\hat{P}_{\hat{m}}| \leq |P_m|$, so as to minimize the sum of the overflow accuracy of all inference requests from each accuracy demand to the exit point allocated to the user.

A solution to the $\mathcal{K}$-median problem returns a solution to the DIA problem, and the reduction process is polynomial. Since the $\mathcal{K}$-median problem is NP-hard, the DIA problem is NP-hard, too. □

Theorem 2. *The SASD problem in a fabric metaverse network is NP-hard.*

Proof. The NP-hardness of the SASD problem is confirmed by a polynomial reduction from the NP-hard problem - Generalized Assignment Problem (GAP), which is defined as follows: Given a group of items $Item = \{I_1, \ldots, I_n\}$, each I_i with a profit $profit_i$ and size $size_i$, and a collection of bins with $Bin = \{b_1, \ldots, b_m\}$, and each b_j has capacity cap_j. The GAP is to maximize the overall profit by allocating as many items as possible into the m bins, subject to capacity cap_j on each bin b_j with $j \in \{1, \ldots, m\}$.

We consider a special case of the SASD problem, where each DNN model with computing demand η_i as a whole and has only one main branch, each server v_j with computing resources Cap_j remains activated and has plenty of green energy, and all inference requests γ_i in $\mathcal{R}$ are issued at the same time. We further assume that the reliability demand of each user is one additional backup, and that delay demand is neglectable. The system generates a profit of 1 when a DNN model and its backup required by a user γ_i is successfully deployed on edge server v, and otherwise, 0. The goal of the SASD problem is to maximize the number of receiving DMS requests by deploying as many satisfied DNNs as possible into edge servers in a fabric network, subject to the capacity cap_j on each edge server.

It can be seen that the special case of the problem is equivalent to the GAP problem, so the SASD problem is NP-hard. □

3.5 Problem Formulations

In this section, we mainly present the modeling of SASD as a Markov Decision Process (MDP) by a tuple $(\mathbb{S}, \mathbb{A}, \mathbb{P}, \mathbb{R})$, while the procedure of modeling DIA as an Integer Linear Programming (ILP) is detailed in Section 4.1.1. The fundamental components in the MDP are defined as follows:

- **State space $\mathbb{S}$:** The fabric metaverse network system observed by an agent (the scheduler in the GCI) at each time slot t consists of a finite group of environment states $s_t \in \mathbb{S}$ containing various information about users, networks, and models, namely:

$$s_t = \{\mathcal{R}, \mathbb{C}, \mathbb{E}, \mathbb{B}, \hat{\mathcal{M}}\},$$

  where $\mathcal{R}$ is the set of DMS requests issued by fabric users, $\mathbb{C} = \{cap_1, \ldots, cap_{|V|}\}$ indicates the set of remaining thread capacities on edge servers, $\mathbb{E} = \{E_1^{green}, \ldots, E_{|V|}^{green}\}$ is the set of newly arrived energy, $\mathbb{B} = \{E_1^b, \ldots, E_{|V|}^b\}$ is the set of available green energy at batteries, and $\hat{\mathcal{M}}$ represents a group of pruned DNN models. Note that at the beginning of each time slot, the above network resources fluctuate dynamically, due to the random movements of users wearing intelligence fabrics and the new DMS requests with individual demands issued by them. In addition, the unpredictability of collected green energy further makes the environment more chaotic.
- **Action space $\mathbb{A}$:** Based on the observed environment state s_t at each time slot t, the agent selects an action a_t from action space $\mathbb{A}$ accordingly, which includes the decision about server activation, block backup, and service deployment, namely $a_t = \{\mathbb{Z}, \mathbb{BN}, \mathbb{MD}\}$, where $\mathbb{Z} = \{z_{v_1}, \ldots, z_{v_{|V|}}\}$ is a group of binary variables indicating whether a server is activated. $\mathbb{BN} = \{BN_1, \ldots, BN_{|\mathcal{R}|}\}$ indicates the DMS backup scheme for all users, where $BN_i = \{q_{i,1}, \ldots, q_{i,|K|}\}$ with length of K stores the number of backup blocks. $\mathbb{MD} = \{MD_1, \ldots, MD_{|\mathcal{R}|}\}$ represent the DNN block deployment scheme for all users. Each $MD_i = \{x_{i,1}, \ldots, x_{i,|K|}\}$ is a vector of length K, which stores the indexes of deployment servers. Note that the cloud center is indicated as 0, namely $x_{i,|K|} = 0$.
- **State transition probability:** The fabric metaverse network system achieves the state transition from the current s_t to the next s_{t+1} when the agent executes an action a_t with conditional probability $p(s_{t+1}|s_t, a_t)$. The environmental state transition is dependent on itself and independent of the agent, where $\mathbb{C}$, $\mathbb{E}$, and $\mathcal{R}$ are randomly extracted from their respective discrete range, and affect other states.
- **Reward:** The reward value acts as a feedback signal from the fabric network environment to evaluate the contribution of action a_t to the agent for achieving the final goal in state s_t. Let f_i be a binary variable indicating that the user demands and server constraints are met for DMS request γ_i. Thus, the reward function $\mathscr{R}_t$ is designed according to the objective and constraints for gradually guiding the agent,

$$\mathscr{R}_t = \psi \cdot \frac{1}{|\mathcal{R}|} \cdot \sum_{i \in \mathcal{R}} f_i - (1 - \psi) \cdot \frac{1}{|V|} \cdot \sum_{j \in V} z_j, \quad (14)$$

  where $\psi > 0$ represents an adjustment coefficient.

Following the MDP modeling, solving SASD is to find the optimal policy to maximize the long-term return $\mathbb{R}$, namely the average reward of all slots, earned by metaverse service providers during a given time horizon $\mathbb{T}$. The definition of the optimal solution policy is given as follows:

Definition 3. ***The optimal policy π^* for the SASD problem:*** *A policy π is defined as a mapping from state s_t to action a_t with reward $\mathscr{R}_t$, namely, $\mathbb{S} \xrightarrow{\pi} \mathbb{A}$. In simpler terms, the scheduler can take action $a_t = \pi(s_t)$ in response to any observed state s_t from the fabric metaverse networks under policy π. The optimal determination policy π^* refers to maximizing the return $\mathbb{R}$ while considering user demands on service model, reliability, delay, and accuracy, as well as server limitations on resource and energy.*

4 Green Continual Inference (GCI) System

In this section, we study continuously individualized DMS provisioning in a fabric metaverse by developing a Green Continual Inference system, namely GCI, as shown in Fig. 1.

The system consists of two parts to solve the DIA and SASD subproblems: the DMS pruner and the DMS scheduler.

4.1 DMS Pruner

The DMS pruner is mainly responsible for personalized model compression based on user and service information, which compresses each overdecorated model by pruning the number of side branches to K based on the practical distribution of accuracy demands for massive users. In the following, we develop two kinds of DMS pruners for the case of fewer models and more models, respectively.

4.1.1 Exact Pruner

According to the above definition, the DIA subproblem can be formulated as an ILP, where the optimization objective of each model $m_j \in \mathcal{M}$ is to

$$\text{Minimize}_{g_{i,k},h_k} \quad \sum_{i\in\mathcal{R}}\sum_{k\in P_m} oa_{i,k}\cdot g_{i,k}, \tag{15}$$

subject to:

$$oa_{i,k} = \begin{cases} a_{p_{i,k}} - A_i, & \text{if } a_{p_{i,k}} \geq A_i, \\ +\infty, & \text{otherwise}, \end{cases} \tag{16}$$

$$\sum_{k\in P_m} g_{i,k} = 1,\ i \in \mathcal{R} \tag{17}$$

$$\sum_{k\in P_m} h_k = K,\ m \in \mathcal{M} \tag{18}$$

$$g_{i,k} \leq h_k,\ k \in P_m, i \in \mathcal{R} \tag{19}$$

$$g_{i,k} \in \{0,1\},\ k \in P_m, i \in \mathcal{R} \tag{20}$$

$$h_k \in \{0,1\},\ k \in P_m, i \in \mathcal{R} \tag{21}$$

Constraint (16) ensures that the accuracy overflow $oa_{i,k}$ of an exit point $p_{i,k}$ exists only when exit accuracy $a_{p_{i,k}}$ is greater than accuracy requirement A_i, i.e., the accuracy gap between the latter and the former. Constraints (17) and (18) indicate that each user can only be assigned one early exit point, and the number of early exit points of each model is equal to K, respectively. Constraint (19) guarantees that whenever a user i is assigned to an early exit point k, then the point must have been available at the k in model M_i. Constraints (20) and (21) are two binary variables, where $g_{i,k}$ indicates if DMS request i is assigned to the kth early exit point in model M_i, and h_k indicates if the kth early exit point in model M_i is inserted, namely reserved.

4.1.2 Approximate Pruner

To avoid the excessive complexity of ILP pruner that experiences exponential growth in running time with problem size, we design an approximation algorithm `Algorithm` 1 to solve DIA in polynomial time as follows.

First, we initialize DNN models and candidate user sets that require the same model (lines 3-5). Specifically, the main branch with the maximum accuracy is set as the default one in branch set $\hat{P}_{\hat{m}}$ of each pruned model. Then, part of the users who choose the same model m are merged into the candidate user set CU_m. In addition, let TU_m and TP_m be temporary variables for CU_m and P_m for subsequent updates to users and early exit points, respectively.

Algorithm 1: AppPruner

Input: A threshold $K \in \mathbb{Z}^+$ of branch number, the set of DNN model m with all exit points P_m, and the set $\mathcal{R}$ of inference requests.
Output: The set of pruned DNN model $\hat{m}$ with selected K exit points $\hat{P}_{\hat{m}}$.

1 **begin**
2 **for** *each DNN model* $m \in \mathcal{M}$ **do**
3 Merge part users who select the m model into the candidate user set CU_m;
4 Revert to the main branch $p_{|P_m|}$ as the last exit point, and pre-assign all users to it;
5 $TU_m \leftarrow CU_m$, $TP_m \leftarrow P_m$, $\hat{P}_{\hat{m}}[K] \leftarrow p_{|P_m|}$;
6 **for** $n \leftarrow K-1$ *to* 1 **do**
7 $index \leftarrow n+1$;
8 **for** *each early exit point* $p_k \in TP_m$ **do**
9 Merge users in TU_m whose accuracy demands are within $[0, a_{p_k}]$ into Ω_k^n;
10 Calculate total accuracy overflow OA_k^n of users in Ω_k^n, OA_{index}^n of all users in $TU_m \backslash \Omega_k^n$ allocated to $\hat{P}_{\hat{m}}[index]$;
11 Calculate the overflow indicator of this branch $OI_k^n = OA_k^n + OA_{index}^n$;
12 Select a branch $p_{k'}$ with the smallest $OI_{k'}^n$ and nonempty $\Omega_{k'}^n$ as the reserved exit point, $\hat{P}_{\hat{m}}[n] \leftarrow p_{k'}$. If not exist, select the former branch of $\hat{P}_{\hat{m}}[index]$ in P_m;
13 Update available exit point set $TP_m \leftarrow TP_m \backslash \left\{p_k | k' \leq k \leq \hat{P}_{\hat{m}}[index]\right\}$;
14 Assign users in $TU_m \backslash \Omega_{k'}^n$ to the branch $\hat{P}_{\hat{m}}[index]$, and pre-assign users in $\Omega_{k'}^n$ to the selected branch $p_{k'}$, $TU_m \leftarrow \Omega_{k'}^n$;

Next, the personalized and pruned model $\hat{m}$ is obtained by iteratively pruning branches to K based on different accuracy demands of users in TU_m and the overflow indicators OI_k^n (lines 8-13). Each early exit point has its own coverage $[0, a_{p_k}]$, and users can only be assigned to the points with higher accuracy than they need. For a certain branch p_k at the n-th iteration, all users in TU_m could be divided into two parts, where Ω_k^n indicates users allocated to this branch and $TU_m \backslash \Omega_k^n$ represents users who belong to the last identified branch (line 10). In addition, the overflow indicator OI_k of a certain branch p_k refers to the sum of the accuracy overflow of the above two parts for all users at this branch, namely OA_k and OA_{index}. The extra computation is less as the overflow indicator is reduced, which in turn saves more energy. Thus, the branch $p_{k'}$ with the minimum overflow indicator is preferred as the reserved early exit point. Note that the remaining branches between the two identified and adjacent branches should be removed because the branch insertion process moves from high to low accuracy for a pre-trained overdecorated model in Tab 2 (line 13).

Finally, we assign users to the selected early exit point iteratively (line 14). The process of user assignment has hysteresis, that is, the user assigned to the $n-1$-th branch is determined only when the n branch is determined. In the last iteration, the pre-assigned users are assigned to the first branch of the model if they exist.

4.2 DMS Scheduler

Another component of the GCI system is the DMS scheduler, which serves to activate the suitable number of servers and deploy DMS requests across these activated edge servers based on the dynamic user and network resource limitations in the fabric metaverse. In the subsequent sections, we design two distinct schedulers based on ILP and reinforcement learning for implementing continual and personalized DMS provisioning in dynamic fabric metaverses with different detection times and network scales.

4.2.1 Exact Scheduler

The first scheduler relies on ILP technology, presenting an exact and optimal scheme for each time slot $t \in \mathbb{T}$ at smaller scales of the fabric metaverse. In particular, the exact scheduler is as an oracle, comprehensively observing the entire state space s and calculating the optimal approach a^* for server activation $\mathbb{Z}^*$, block backup $\mathbb{BN}^*$, and service deployment $\mathbb{MD}^*$. More specifically, the exact scheduler transforms the SASD subproblem from an MDP into an ILP to maximize the average reward $\mathbb{R}$ at all slots as follows.

$$\text{Maximize}_{x_{i,k,j}(t),q_{i,k}(t),z_j(t)} \quad \frac{1}{|\mathbb{T}|}\sum_{t\in\mathbb{T}}\mathscr{R}_t, \tag{22}$$

subject to:

$$x_{i,k,j}(t) \le z_j(t), i \in \mathcal{R}, k \le K, v_j \in V \tag{23}$$

$$\sum_{v_j\in V} x_{i,k,j}(t) = 1, i \in \mathcal{R}, k \le K \tag{24}$$

$$r_i(t) \ge R_i(t), i \in \mathcal{R} \tag{25}$$

$$(E_j^{cm}(t) + E_j^{cp}(t) + E_j^{bu}(t) + E^{leak}) \cdot z_j(t) + E^{sleep} \cdot (1 - z_j(t)) \le E_v^{green}(t), v_j \in V \tag{26}$$

$$\sum_{k\le K}\sum_{i\in\mathcal{R}} \eta_i(t) \cdot x_{i,k,j}(t) \cdot (1 + q_{i,k}) \le Cap_j, v_j \in V \tag{27}$$

$$x_{i,k,j}(t) \in \{0,1\}, i \in \mathcal{R}, k \le k', v_j \in V \tag{28}$$

$$x_{i,k,j}(t) = 0, i \in \mathcal{R}, k > k', v_j \in V \tag{29}$$

$$z_j(t) \in \{0,1\}, v_j \in V \tag{30}$$

Constraints (23) and (24) indicate that each block of a DMS is deployed to only an activated server. Constraint (25) is user demands on service reliability. Constraints (26) and (27) represent that the green energy and computing resources required to receive DMS requests do not exceed the limitation of the fabric metaverse network. Then, an exact algorithm `Algorithm` 2 based on an ILP solver is designed to maximize the throughput of DMS requests while minimizing the number of activated servers.

4.2.2 Hybrid reinforcement learning-based Scheduler

The ILP solution cannot predict subsequent server activation and is only suitable for small problems because its running time increases exponentially with problem size. To overcome this shortcoming, the second scheduler based on a hybrid reinforcement learning technique is designed, which can address the activation prediction challenges over a period of time $T \in \mathbb{T}$ during continual DMS provisioning in large-scale fabric metaverse caused by intermittent energy, time-varying computing resources, and dynamic random moving users with personalized demands. This scheduler has two symbiotic components, i.e., an activator and a deployer library, where the former specifies the deployment scope over time T, and the latter provides a near-optimal deployment solution at each time slot $t \in T$ as training rewards to the former.

Algorithm 2: ExaScheduler

```
Input: The fabric metaverse states s_t at every slot.
Output: The optimal scheme a_t^* at every slot.
begin
    for each time slot t ∈ T do
        for each DMS request γ_i ∈ R do
            Initialize reliability list RL_i by using the
              default reliability of each block;
            while r_i < R_i do
                Add a backup block for the block b_k with
                  the lowest reliability in RL_i;
                Update the reliability r̂_{i,k} of the block b_k
                  in RL_i by the formula (4);
                Calculate the reliability r_i of the request
                  by the formula (5);
        Construct an ILP model;
        Obtain optimal solution on server activation Z^*
          and service deployment MD^* by an ILP solver;
    return The optimal scheme a_t^* at every slot.
```

The training process of the learning-based activator in HrlScheduler is shown in `Algorithm` 3 in detail. First, compress models and activate the appropriate edge servers at begin of each time interval T. By running the pruner `Algorithm` 1, the overdecorated models are pruned, and early exit points meeting accuracy demands are assigned to users. In addition, a noise process $\mathcal{N}$ is added to the sample of activation action to avoid getting stuck in a local optimum. The actual activation action is the value of the probability distribution calculated by the policy minus the random noise because metaverse service providers want to use the fewest edge servers to maximize user throughput.

Next, deployers in the deployer library perform service deployment actions and calculate corresponding rewards according to formula (14), which increases the expandability of the training framework. Note that only one deployer can be used during training. The deployer can find a near-optimal block backup and service deployment solutions based on the current server activation status and constraints at each slot $t \in T$. For example, `Algorithm` 4 only considers the server constraints on energy and computing resources and user demands on reliability and model, while `Algorithm` 5 additionally considers user demands on service delay. After deployment, the gap between the percentage of accepting requests and activated servers is stored in a replay buffer as a reward by using formula (14).

Finally, a batch of data from the replay buffer is randomly selected to update the related neural networks. Specifically, the critic network θ_c is updated to minimize the loss function $\mathcal{L}$ using gradient descent, where $y_{T_i} = r_{T_i} + \theta_c^t(s_{T_{i+1}}, \theta_p^t(s_{T_{i+1}}))$. Subsequently, the upgraded critic network is utilized to refine more accurate values that aid in the policy network θ_p updating via the policy gradient. Then, both the target neural networks θ_c^t and θ_p^t are updated

Algorithm 3: Training process of HrlScheduler

Input: The fabric metaverse state s_t at each slot, initialized policy θ_p, critic θ_c, and target policy θ_p^t and critic θ_c^t neural network.
Output: A near-optimal policy neural network θ_p.

begin
  for *each episode* $ep \in EP$ **do**
    Prune overdecorated DMSs and allocate exit points for users by `Algorithm` 1;
    Observe state s_T and select activation action $\mathbb{Z}_T = \theta_p(s_T)$ - ϵ, where $\epsilon \sim \mathcal{N}$ is noise;
    for *each time slot* t *in slot period* $T \in \mathbb{T}$ **do**
      if *DelayFlag* == *False* **then**
        Execute `Algorithm` 4 to get scheme on $\mathbb{BN}_t$ and $\mathbb{MD}_t$, and reward $\mathscr{R}_t$;
      else
        Execute `Algorithm` 5 to get scheme on $\mathbb{BN}_t$ and $\mathbb{MD}_t$, and reward $\mathscr{R}_t$;
      Observe the next state s_{t+1};
    Calculate average reward $\mathscr{R}_T$ in the period;
    Generate new state s_{T+1} of green energy, computing resources, and DMS requests;
    Prune overdecorated DMSs and allocate exit points for users by `Algorithm` 1;
    Store $(s_T, \mathbb{Z}_T, s_{T+1}, \mathscr{R}_T)$ in replay buffer;
    if ep % *update interval* == 0 **then**
      Randomly sample a batch of data from the replay buffer;
      Update parameter of critic θ_c by gradient descent based on loss function $\mathcal{L}$, where $\mathcal{L} = \frac{1}{N_{bath}} \sum_{T_i \in N_{bath}} (y_{T_i} - \theta_c(s_{T_i}, \mathbb{Z}_{T_i}))^2$;
      Update parameter of policy θ_p by gradient ascent based on the following formula $\nabla_p = \frac{1}{N_{bath}} \sum_{T_i \in N_{bath}} \nabla_{\theta_p} \theta_c(s_{T_i}, \theta_p(s_{T_i}))$;
      Soft update target network θ_p^t and θ_c^t;
  return A near-optimal policy neural network θ_p.

using the soft target update method.

4.2.3 Different deployers in the HrlScheduler

In the following, two deployers are designed to place DMSs with differential delay constraints.

Approximate Deployer: The approximate deployer is geared toward DMS provisioning without delay requirements, such as sleep detection and social media. Therefore, it primarily considers the server constraints on green energy and computing resources and user demands on services model and reliability. The AppDeployer `Algorithm` 4 is described as follows. First, the algorithm determines the minimum number of backups by providing backups for the block with the least reliability in each request iteratively until it meets user reliability requirements. Then, it constructs a GAP instance, where all blocks and their backups in each DMS request are regarded as items with two dimensions demands on computing and energy resources, and all activated edge servers are represented as bins with dynamic limitations on computing and battery capacity (line 3). Finally, the defined GAP instance is solved by integrating a two-dimensional knapsack algorithm into a framework proposed by [30], and the deployment location of each block

Algorithm 4: AppDeployer

Input: The fabric metaverse states s_t at each time slot, and server activation scheme $\mathbb{Z}_T$ obtained by a policy θ_p at a time period T.
Output: The near-optimal scheme on block backups $\mathbb{BN}$ and service deployment $\mathbb{MD}$ at a slot t.

begin
  Calculate the optimal block backup $\mathbb{BN}^*$ based on formulas (4), (5), and (25);
  Construct a GAP instance where each activated server with Cap_j and E_j^{green} is as bins, and a block and its backups in each request is as items;
  A two-dimensional knapsack algorithm with an approximation ratio of 2 is combined with the architecture [30] to obtain a feasible solution $\mathcal{S}$;
  for *each DMS request* $\gamma_i \in \mathcal{R}$ **do**
    if $\min \mathcal{S}_i >= 0$ **then**
      Store deploy location $\mathcal{S}_{i,k}$ of each block and its backups in $x_{i,k,\mathcal{S}_{i,k}}$, and allocate energy and computing resources;
    else
      Reject the DMS request γ_i;
  Calculate reward $\mathscr{R}_t$ by the formula (14);
  return A scheme on $\mathbb{BN}^*$, $\mathbb{MD}$, and reward $\mathscr{R}_t$

Algorithm 5: HeuDeployer

Input: The fabric metaverse states s_t at each time slot, and server activation scheme $\mathbb{Z}_T$ obtained by a policy θ_p at a time period T.
Output: The near-optimal scheme on block backups $\mathbb{BN}$ and service deployment $\mathbb{MD}$ at a slot t.

begin
  Calculate the optimal block backup $\mathbb{BN}^*$ based on formulas (4), (5), and (25);
  for $k \leq K$ **do**
    Sort DMS requests in ascending order based on its available delay threshold D_i;
    for *each DMS request* γ_i *in order* **do**
      if $k == 1$ **then**
        Find an activated server v_j with shortest path $\mathscr{P}_{i,j}$ and with sufficient energy and computing resources for the first block and its backups in the request γ_i ;
        Update usable resources and remaining delay $D_i = D_i - D_{t,i,1} - D_{c,i,1} - D_{p,i,1}$;
      else if $1 < k \leq k_i'$ **then**
        Find the set $\{v_{i,k}\}$ of activated servers meeting two kinds of resource demands and delay threshold D_i;
        if $\{v_{i,k}\} \neq \varnothing$ **then**
          Allocate the block $b_{i,k}$ and its backups to edge server $v_{i,k}'$ with maximum resources;
          Update usable resources and delay $D_i = D_i - D_{t,i,k} - D_{c,i,k} - D_{p,i,k}$;
        else
          Reject γ_i and recycle resources allocated to previous blocks;
      else
        Deploy this block in the cloud;
  Calculate reward $\mathscr{R}_t$ by the formula (14);
  return A scheme on $\mathbb{BN}^*$, $\mathbb{MD}$, and reward $\mathscr{R}_t$

and its backups is obtained. The approximation algorithm with an approximation ratio of 2 for the two-dimensional knapsack problem is given as follows: It sorts all items in descending order of their average value that is the inverse of the sum of computing resources and green energy. The sorted items are then placed in bins one by one until the capacity of the bin is violated. The final items placed in a bin are all items before the one without violating the bin constraint, or the item with the bigger profit after the one yet without violating. This procedure continues until all items are placed or no bin has room for further item placements.

Heuristic Deployer: The second deployer is to place delay-sensitive DMSs such as human action recognition and emotion recognition, which takes into account the delay constraint. The HeuDeployer `Algorithm` 5 is described as follows. Regarding algorithm AppDeployer, the best backup scheme is calculated first. Followed by a semi-lazy allocation strategy to consider the delay constraint. Specifically, for request γ_i of user i, the first block and its backups are always placed to an activated server v_j with sufficient resources closely to the user, i.e, there is the shortest path $\mathscr{P}_{i,j}$ between user i and server $v_j \in V$ in G (line 7). For subsequent blocks, namely before the index k_i' of assigned exit point, they are deployed to the activated servers that just meet the remaining delay threshold and have the most remaining resources of all servers to reduce resource competition among users. In addition, during the deployment process, this algorithm selects the appropriate server for each requested block in turn, which further reduces competition and improves fairness. Note that each time a block is successfully deployed, the remaining delay is updated.

The detailed implementation of the GCI system is presented in `Algorithm` 6, where `Algorithm` 3 is executed once every slot period T, and `Algorithm` 4 or `Algorithm` 5 is executed at each time slot t to obtain near-optimal action a_t.

Algorithm 6: GCI system

Input: The well-trained policy θ_p and the fabric metaverse states s_t at each time slot.
Output: The near-optimal action a_t at a slot t.

1 **begin**
2 **for** *each time slot* $t \in \mathbb{T}$ **do**
3 Prune overdecorated DMSs and allocate exit points for users by `Algorithm` 1;
4 **if** t mod *slot period* T == *0* **then**
5 Observe state s_T and calculate server activation $\mathbb{Z}_T$ by policy θ_p in `Algorithm` 3;
6 **if** *DelayFlag* == *False* **then**
7 Execute `Algorithm` 4 to get schemes on backup $\mathbb{BN}_t$ and deployment $\mathbb{MD}_t$ based on the activation situation $\mathbb{Z}_T$;
8 **else**
9 Execute `Algorithm` 5 to get schemes on backup $\mathbb{BN}_t$ and deployment $\mathbb{MD}_t$ based on the activation situation $\mathbb{Z}_T$;

4.3 Algorithm Analysis

At the end of this section, we analyze the approximation ratio and time complexity of the proposed algorithms.

Lemma 1. *The accuracy overflow of all selected users declines with the insertion of side branches for any single DMS in* `Algorithm` *1.*

Proof. For a DMS, the total accuracy overflow $A^n_{overflow}$ in the nth iteration includes two parts: the verified accuracy overflow $\sum_n^{K-1} OA^n_{index}$ for some users with assigned exit points, and the uncertain accuracy overflow $OA^n_{k'}$ for some pre-assigned users. The gap between the accuracy overflows between iterations n and $n-1$ is estimated as follows:

$$\begin{aligned}
&A^n_{overflow} - A^{n-1}_{overflow}\\
=&OA^n_{k'} + \sum_{n}^{K-1} OA^n_{index} - (OA^{n-1}_{k'} + \sum_{n-1}^{K-1} OA^n_{index})\\
=&OA^n_{k'} - (OA^{n-1}_{k'} + OA^{n-1}_{index})\\
=&|\Omega^{n-1}_{k'}|(a_{\hat{P}[n]} - a_{\hat{P}[n-1]}) \qquad (31)\\
\geq&0,
\end{aligned}$$

where inequality (31) represents the reduction value of accuracy overflow because some users are assigned to closer early exit points in the next iteration ($n-1$). Thus, the total accuracy overflow decreases as the iteration process of insertion branches increases. □

Theorem 3. *Given the set $\mathcal{R}$ of DMS inference requests from different users wearing intelligence fibers, and a given branch number threshold K, there is an approximation algorithm,* `Algorithm` *1, with bounded approximation ratio 2ξ to compress each model by pruning the branches to K to reduce the green energy consumption in $O(|M||K|(|\mathcal{R}||P_m|+1))$, where $|P_m|$ is the branch number of an overdecorated DNN model m, δ represents the average value of overflow accuracy of all DMS requests for the DNN model m, $a_{\hat{P}[K]}$ is the accuracy of the main branch $\hat{P}_{\hat{m}}[K]$, and $\xi = a_{\hat{P}[K]}/\delta$.*

Proof. To simplify the proof, we calculate only the approximation ratio for a single model, while the actual approximation ratio is the maximum of the upper limit of overflow accuracy among all models. To analyze the approximation ratio of `Algorithm` 1, we denote by Φ the larger one between the number of chosen users $\Omega^n_{k'}$ and the number of not chosen users $TU_m \backslash \Omega^n_{k'}$ for a model m, assuming that each user requests only one DMS model each time, then $\Phi = \max_{k' \in \hat{P}_{\hat{m}}} \{\Omega^n_{k'}, TU_m \backslash \Omega^n_{k'}\}$. The upper bound on the gap $Gap^{n,n-1}_{OI}$ of accuracy indicators between two consecutive iterations is

$$\begin{aligned}
Gap^{n,n-1}_{OI} =&OI^n_{k'} - OI^{n-1}_{k'}\\
=&OA^n_{k'} + OA^n_{index} - (OA^{n-1}_{k'} + OA^{n-1}_{index})\\
=&|\Omega^{n-1}_{k'}|(a_{\hat{P}[n]} - a_{\hat{P}[n-1]}) + OA^n_{index} \qquad (32)\\
=&|\Omega^{n-1}_{k'}|(a_{\hat{P}[n]} - a_{\hat{P}[n-1]})\\
&+ \sum_{i \in TU_m \backslash \Omega^n_{k'}} (a_{\hat{P}[n+1]} - A_i)\\
\leq&|\Omega^{n-1}_{k'}|(a_{\hat{P}[n]} - a_{\hat{P}[n-1]})\\
&+ |TU_m - \Omega^n_{k'}|(a_{\hat{P}[n+1]} - a_{\hat{P}[n]})\\
\leq&|\Phi|(a_{\hat{P}[n]} - a_{\hat{P}[n-1]} + a_{\hat{P}[n+1]} - a_{\hat{P}[n]}) \qquad (33)\\
\leq&|\Phi|(a_{\hat{P}[n+1]} - a_{\hat{P}[n-1]}), \qquad (34)
\end{aligned}$$

where formula (32) indicates the reduction in the total accuracy overflow after a new branch is inserted, and inequality (33) holds because of the property of Φ itself. Based on inequality (34), the following inequality is derived,

$$\begin{aligned} OI_{k'}^{K-1} - OI_{k'}^{1} &= \sum_{n=2}^{K-1} (OI_{k'}^{n} - OI_{k'}^{n-1}) \\ &\leq |\Phi|(a_{\hat{P}[K]} - a_{\hat{P}[1]}). \end{aligned} \tag{35}$$

Next, by inequality (35), the upper bound on the accuracy indicator $OI_{k'}^{K-1}$ is given as follows:

$$\begin{aligned} OI_{k'}^{K-1} &\leq OI_{k'}^{1} + |\Phi|(a_{\hat{P}[K]} - a_{\hat{P}[1]}) \\ &\leq |\Phi|(a_{\hat{P}[K]} + a_{\hat{P}[2]}). \end{aligned} \tag{36}$$

Thus, the approximation ratio of `AppPruner` is

$$\begin{aligned} \frac{A_{overflow}}{A^*_{overflow}} &= \frac{OA_{k'}^{1} + \sum_{n=1}^{K-1} OA_{index}^{n}}{A^*_{overflow}} \\ &= \frac{OI_{k'}^{1} + \sum_{n=2}^{K-1} OA_{index}^{n}}{\sum_{i\in\mathcal{R}} oa_{i,k'}} \\ &\leq \frac{OI_{k'}^{K-1} + \sum_{n=2}^{K-1} OA_{index}^{n}}{\sum_{i\in\mathcal{R}} oa_{i,k'}} \qquad (37) \\ &\leq \frac{OI_{k'}^{K-1} + \sum_{n=2}^{K-1} OA_{index}^{n}}{|\Phi|\cdot\delta}, \\ &\text{let } \delta = \frac{\sum_{i\in\mathcal{R}} \min_{a_{p_{i,k}} - A_i > 0 (k\in P_m)} (a_{p_{i,k}} - A_i)}{|\mathcal{R}|} \qquad (38) \\ &\leq \frac{|\Phi|(a_{\hat{P}[K]} + a_{\hat{P}[2]} + a_{\hat{P}[K]} - a_{\hat{P}[2]})}{|\Phi|\cdot\delta} \qquad (39) \\ &\leq 2\cdot\xi, \text{where } \xi = \frac{a_{\hat{P}[K]}}{\delta} \end{aligned}$$

Inequality (37) holds because the accuracy overflow decreases with the increase in branch number (Lemma 1). Inequality (38) indicates the ideal lowest bound of overflow accuracy is that all services exit from the nearest exit point of an overdecorated DNN m, and δ is the average of overflow accuracy in this case. Inequality (39) holds based on inequality (36) and the same principle in proof of $Gap_{OI}^{n,n-1}$.

The time complexity of `Algorithm 1` is analyzed as follows. Initializing related variables and grouping users takes $O(|\mathcal{R}|)$ time. For an optional branch, the calculation and selection of the overflow indicators consume $O(|\mathcal{R}|\cdot|P_m|)$ time. Branch insertion, user assignment, and their updating process take $O(1)$ time. This process is performed for each branch and each model, so the total time complexity of the algorithm is $O(|M|\cdot|K|(|\mathcal{R}|\cdot|P_m|+1))$. □

Lemma 2. *The approximation ratio of the sub-algorithm for the two-dimensional knapsack problem in* `Algorithm` *4 is 2.*

Proof. Recall that the profit per item I_u, namely the number of blocks accepted, is $pr_u = 1/K$ in Theorem 2 when a request with K block and their backups. Assuming that the total profit of loaded items LI calculated by the approximation algorithm is tp and the theoretical optimal profit is tp^*. The sorted items satisfy the following inequality,

$$\frac{1/K}{C_1+E_1} \geq \cdots \geq \frac{1/K}{C_u+E_u} \geq \cdots \geq \frac{1/K}{C_{\sum_{i\in\mathcal{R}} k_i'} + E_{\sum_{i\in\mathcal{R}} k_i'}},$$

where C_u and E_u represent resource demands on the computing capacity and energy for an item, and k_i' is an index of exit point assigned to request γ_i. When the $\widetilde{u}$-th item violates the server's energy and compute capacity simultaneously, the approximation profit is $tp = \max\{\sum_{u\in LI} pr_u, pr_{\widetilde{u}}\}$. Then,

$$\begin{aligned} \frac{tp^*}{tp} &\leq \frac{\sum_{u\in LI} pr_u + pr_{\widetilde{u}} \cdot \frac{Cap_j - \sum_{u\in LI} C_u + E_j^{green} - \sum_{u\in LI} E_u}{C_{\widetilde{u}}+E_{\widetilde{u}}}}{tp} \\ &< \frac{\sum_{u\in LI} pr_u + pr_{\widetilde{u}} \cdot \frac{C_{\widetilde{u}}+E_{\widetilde{u}}}{C_{\widetilde{u}}+E_{\widetilde{u}}}}{tp} \\ &= \frac{\sum_{u\in LI} pr_u + pr_{\widetilde{u}}}{\max\{pr_{\widetilde{u}}, \sum_{u\in LI} pr_u\}} \\ &\leq \frac{\sum_{u\in LI} pr_u + pr_{\widetilde{u}}}{\frac{1}{2}\cdot(\sum_{u\in LI} pr_u + pr_{\widetilde{u}})} = 2. \end{aligned}$$

Thus, the approximation ratio of the sub-algorithm for the two-dimensional knapsack problem is 2. □

Theorem 4. *Given a fabric metaverse network $G = (\{v_0\}\cup V, E)$, the set $\mathcal{R}$ of DMS inference requests from different users wearing intelligence fibers, and a collection of metaverse services based on pruned DNN models $\hat{\mathcal{M}}$, we consider an approximation AppDeployer* `Algorithm` *4 with a constant approximation ratio 3 in $O(|\mathcal{R}||K||V|^2)$ and a heuristic HeuDeployer* `Algorithm` *5 in $O(|\mathcal{R}||K||V|)$ to deploy respectively required blocks and their corresponding backups on selected active servers for maximizing request throughput according to whether to consider additional delay constraints.*

Proof. The approximation ratio of GAP architecture is $1+\alpha$ [30]. A sub-algorithm with an approximation ratio of 2 in `Algorithm` 4, namely $\alpha = 2$, is designed for the two-dimensional knapsack problem as shown in Lemma 2. Since the system accepts a request when all blocks before $p_{i,k'}$ in the request are placed, the conversion function from block to request can be viewed as a monotonically increasing piecewise function. Therefore, the total number of accepted requests calculated by AppDeployer is not more than 3 times the optimal result after experiencing the conversion function due to the function monotonicity and Lemma 2.

Then, the time complexity of AppDeployer and HeuDeployer are analyzed respectively. In terms of AppDeployer, it incurs the complexity of $O(|\mathcal{R}||K|)$ to calculate the backup scheme. The GAP instance is constructed by consuming $O(|\mathcal{R}||K||V|)$. The complexity is $O(|\sum_{i\in\mathcal{R}} k_i'||V| + |\sum_{i\in\mathcal{R}} k_i'|\log|\sum_{i\in\mathcal{R}} k_i'|)$ to execute the two-dimensional knapsack sub-algorithm. It also causes the complexity of $O(|V|(|\sum_{i\in\mathcal{R}} k_i'||V| + |\sum_{i\in\mathcal{R}} k_i'|\log|\sum_{i\in\mathcal{R}} k_i'|) + |\sum_{i\in\mathcal{R}} k_i'||V|)$ to solve the GAP instance [30]. Thus, the total time complexity of AppDeployer is $O(|\mathcal{R}||K||V|^2)$.

TABLE 2: Model parameters of DNN-based metaverse services

DNN model	Exit number	Accuracy	FLOPs of primary branches (e7)	FLOPs of side branches (e6)
Sitting Posture Analysis	4	[0.7408, 0.8296, 0.9796, 0.9990]	[0.0029, 0.0229, 0.0852, 0.3278]	[0.2675, 0.5338, 1.0663, 4.2986]
1D_CNN_HAR	4	[0.8755, 0.8982, 0.9060, 0.9155]	[0.1016, 2.5437, 5.0597, 5.0597]	[0.1311, 0.2622, 4.2671, 8.5129]
Sleep Stage Classification	4	[0.7124, 0.7445, 0.7617, 0.7943]	[7.0464, 3.3298, 6.6579, 0.1141]	[0.3200, 0.3226, 0.0819, 0.0192]

In terms of HeuDeployer, it also incurs the complexity of $O(|\mathcal{R}||K|)$ to calculate the backup scheme. The blocks required to deploy separately for all requests consume $O(2|K||\mathcal{R}||V|)$. Thus, the total time complexity of HeuDeployer is $O(|K||\mathcal{R}||V|)$. □

5 Performance Evaluation

In this section, we first introduce related experiment parameters and the comparison algorithms. Then, we demonstrate a hardware prototype, and conduct analog simulations and experiments. Finally, we evaluate the performance of the proposed algorithms in the GCI system.

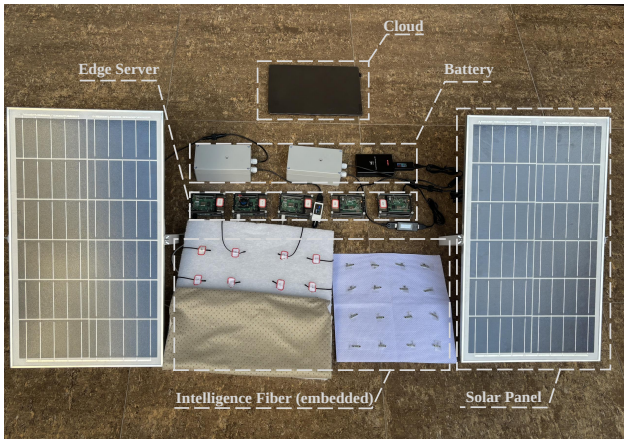


Fig. 2: The hardware prototype

5.1 Experimental environments

The below analog simulations are conducted in a fabric metaverse network that consists of a remote cloud center and the Abilene network [31] with 11 edge servers as an underlying topology, where the simulation network is deployed in a 13 Megametre × 13 Megametre square area. Within this network topology, the computing resources of each edge server are randomly allocated from 128 to 256 thread [32], and the floating-point calculation of each thread is 3.375 GFLOPS [33]. Transmission data sizes per block of a model fluctuate between 10 to 100 Mbit, while network bandwidth remains steadfast at 1000 Mbit/s [34]. Energy consumption during sleep, leakage, and backup monitoring is 5, 5, and 1 $J/slot$ respectively, while servers collect green energy between 25 to 50 $J/slot$ that is stored in a battery with a capacity of 1000 J. To facilitate concept testing and validation, we also design a hardware prototype as a simplified and preliminary platform depicted in Fig 2, where a laptop symbolizes the cloud server, interconnected Raspberry Pi serves as the edge server, a cushion embedded with multiple flexible pressure sensors embodies the intelligence fabric, a mobile charging bank signifies the battery, and the solar panel represents the green energy collector.

In a fabric metaverse, multiple DNNs based on metaverse services are introduced to process data from embedded intelligence fibers, whose parameters are shown in Tab 2. The first model utilizes a 16x16 pressure dataset from an intelligence fabric cushion to analyze sitting posture. It employs a four-layer convolutional structure with four exit points to accurately identify forward, backward, left, and right leaning positions. The 1D_CNN_HAR service [35], based on the UCI-HAR dataset [36], employs a 1D CNN model with four branches to recognize human activity. Similarly, the sleep stage classification model, based on the Sleep-EDF-78 dataset [37], also uses a four-branch architecture to assess sleep quality. In addition to model requirement for each user, the reliability demand falls within the range of 0.8 and 1, the delay demand is extracted from 1.5 to 2.5 s, the confidence threshold affecting accuracy is set between 0 and 1, and the thread resource demand is extracted between 4 and 8. Note that each user randomly moves to any location in the network at the beginning of each time slot and makes a new request. Each slot includes 60 s. The above values are default values for each trial unless otherwise specified.

5.2 Performance evaluation of the algorithms

To evaluate the performance of the proposed algorithms for the GCI system, the following algorithms are introduced as comparison benchmarks.

Random (Random + RAD [38]): It retains K branches randomly for each model, and randomly selects one of the branches that meets the accuracy requirement of each user. The RAD scheme then is employed to determine necessary backups, and services along with their backups are deployed randomly to activated servers.

GCI: It begins by utilizing the AppPruner `Algorithm` 1 to reduce excessive branches to K. It then selects a subset of edge servers to activate using the LeaActivator `Algorithm` 3. It finally adopts the AppDeployer `Algorithm` 4 to allocate services and their backups on the servers.

GCI_D: In contrast to GCI that weighs users on reliability and model requirements, and server constraints on computing and energy, GCI_D takes an additional delay requirement. It makes use of the HeuDeployer `Algorithm` 5 to deploy delay-sensitive services and offers rewards for the LeaActivator `Algorithm` 3.

ILP [39]: It first relaxes the ILPs for DIA and SASD subproblems in Section 4 into linear programming to reduce their running time. It then adopts the cplex solver in Python for the problems. In comparison experiments, the ILP algorithm is considered to deliver an optimal solution to the problem with the maximum total reward.

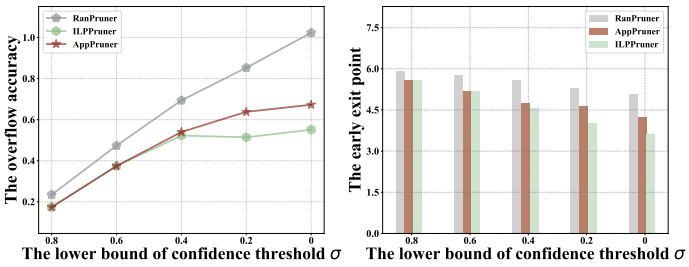
5.3 Impact of parameters on the performance of the proposed algorithms

The impacts of key parameters on the performance of different algorithms are examined, with detailed experimental configurations given in Table 3 including confidence threshold, adjustment coefficient, block reliability, green energy, computing capacity, delay demand, and request number.

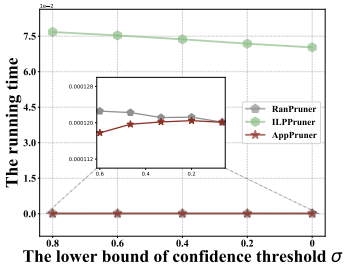
TABLE 3: Parameter settings

	ψ	σ_i	r	E_j^{green}	Cap_j	D_i	$\|\mathcal{R}\|$	$\|V\|$
Set1	0.25−0.75	0−1	0.8−1	25−50	128−256	1.5−2.5	20	11
Set2	0.5	0−1	0.8−1	25−50	128−256	1.5−2.5	20	11
Set3	0.5	0−1	0.8−1	25−50	128−256	1.5−2.5	20	11
Set4	0.5	0−1	0.8−1	18−92	128−256	1.5−2.5	20	11
Set5	0.5	0−1	0.8−1	25−50	32−192	1.5−2.5	20	11
Set6	0.5	0−1	0.8−1	25−50	128−256	0.5−1.3	20	11
Set7	-	0−1	0.8−1	-	128−256	-	10−30	5

Impact of accuracy demands on the performance of different pruner algorithms: We adopt the parameter setting **Set 1** in Table 3, wherein we widen the distribution of confidence threshold σ from $[0.8, 1]$ to $[0, 1]$ to investigate its impact on performance. From Fig. 3(a), *RanPrunner* exhibits the highest overflow accuracy between accuracy demands and reserved branches assigned to users as the accuracy distribution interval widens, while the gap between *AppPruner* and *ILPPruner* remains relatively stable. The reason for the stability in overflow accuracy is hidden in Fig. 3(b), where some users can choose earlier exit points as the accuracy distribution increases. However, the random exit method causes the accuracy overflow to continue to increase. The increase in running time of *AppPruner* is attributed to the increasing check times for early exit points to which users belong due to a widening range of accuracy demand in Fig. 3(c).



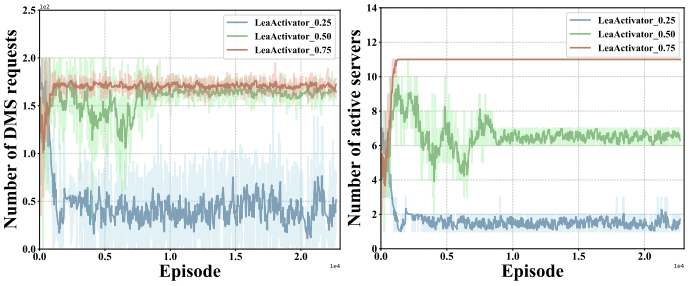
(a) The number of admitted requests (b) The number of activated servers



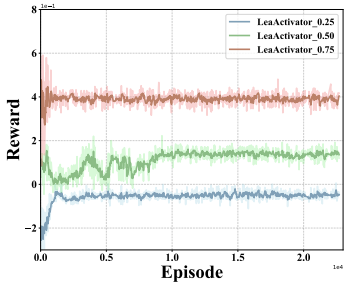
(c) The total reward

Fig. 3: The performance of different pruners by varying the distributions of confidence threshold σ form $[0.8, 1]$ to $[0, 1]$.

Impact of adjusting coefficient on the performance of the LeaActivator algorithm: We employ the parameter setting **Set 2** in Table 3, wherein we vary coefficient ψ from 0.25 to 0.75, aiming to scrutinize its influence on the performance of the proposed algorithm. We also train the policy and critic neural networks of LeaActivator algorithm, using the Adam Optimizer over 24,000 episodes with 10 time slots each. The neural network architecture consists of 4 convolutional layers (16, 32, 32, 4) and 3 fully connected layers (512, 256, 11), in which the ReLU as activation functions after each layer. From Fig. 4(c), the total reward experiences a gradual increase with the increase on the value of coefficient. This is because requests account for a relatively high proportion of the total reward, and the acceptance of requests exhibits a gradual rise alongside the increase of coefficient ψ as shown in Fig. 4(a). From Fig. 4(b), with the increase of training episodes, the activation strategy explores and learns enough unknown environments, resulting in a steady trend in the number of activated servers.



(a) The number of admitted requests (b) The number of activated servers

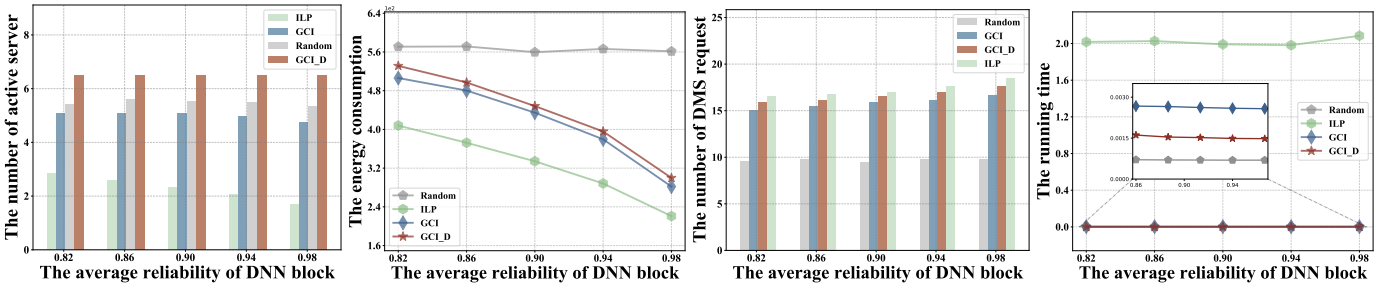


(c) The total reward

Fig. 4: The performance of the LeaActivator algorithm by varying the coefficient ψ from 0.25 to 0.75.

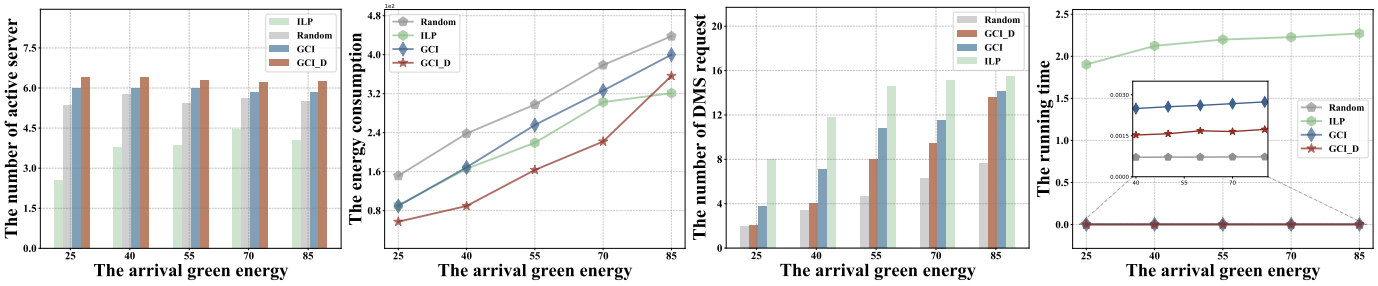
Impact of block reliability on the performance of different algorithms: We make use of the parameter setting **Set 3** in Table 3, wherein we investigate the impact of the parameter on the performance of the algorithms by increasing the average reliability r of DNN block from 0.82 to 0.98. As depicted in Fig. 5(a), *ILP* has the lowest number of activated servers, followed by the *GCI*, and both tend to decrease due to the decrease in the number of required backups caused by the increase in reliability. The reason why the data in *GCI_D* remains stable is due to increasing accepted requests, as shown in Fig. 5(c), and the abnormal increase in runtime depicted in Fig. 5(d) is also due to this trend. In addition, from Fig. 5(b), the energy consumption of all algorithms tends to decline as backup number decreases.

Impact of green energy on the performance of different algorithms: We conduct the parameter configuration **Set 4** outlined in Table 3, where we elevate the range of green energy from $[18, 32]$ to $[78, 92]$ to explore its influence on the algorithm performance. As observed in Fig. 6(a), as the



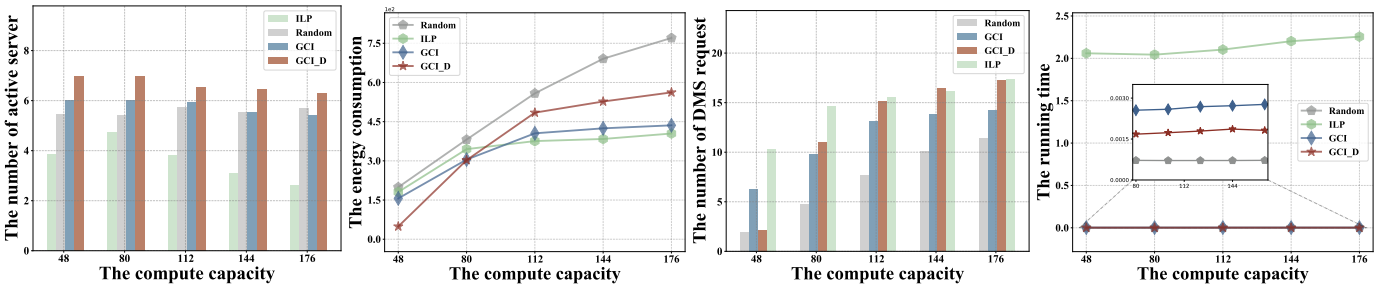
(a) The number of activated server (b) The energy consumption (c) The number of admitted requests (d) The running time

Fig. 5: The performance of different algorithms by varying the average reliability r of DNN block from 0.82 to 0.98.



(a) The number of activated servers (b) The total energy consumption (c) The number of admitted requests (d) The running time

Fig. 6: The performance of different algorithms by varying the average value of green energy E_j^{green} from 25 to 85.



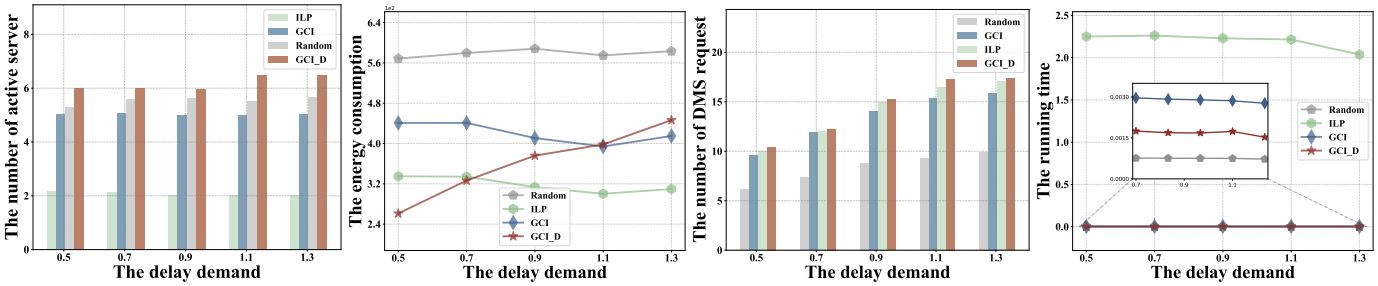
(a) The number of activated servers (b) The total energy consumption (c) The number of admitted requests (d) The running time

Fig. 7: The performance of different algorithms by varying the average value of computing capacity Cap_j from 48 to 176.

generated green energy increases, activated servers with by both *GCI* and *GCI_D* exhibit a slightly declining trend, whereas *ILP* demonstrates an initial increase followed by a decline. This phenomenon occurs because when the number of requests accepted by *ILP* stabilizes, an increase in arrival energy reduces the demand for servers due to the enhanced load capacity of edge servers. From Fig. 6(c), *Random* has the lowest request acceptance number, followed by *GCI_D* which accepts fewer requests than *GCI*, while ILP has the highest acceptance number. The fairness of the *GCI_D*, that is, locating each block of each request in turn rather than prioritizing all blocks in a single request, results in a lower number of accepted requests than *GCI* when resources are insufficient, a trend that can also be seen in Fig. 6(c). Furthermore, the primary factor driving the rise in energy consumption in Fig. 6(b) and running time in Fig. 6(d) across all methods is the increasing number of accepted requests in Fig. 6(c).

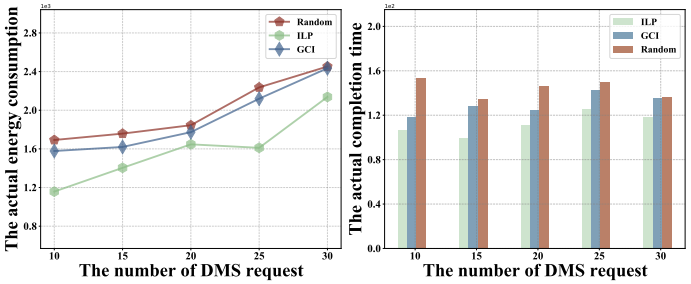
Impact of computing resource on the performance of different algorithms: We apply the parameter configuration **Set 5** form Table 3, and increase the range of computing capacity from $[32, 64]$ to $[160, 192]$ to examine its effect on performance, and set the thread demand of user is 8. Similar to simulation on energy, the related performance of all algorithms increases with the increase of computing resources, except for the decreasing trend in the number of activated servers. It is worth noting that at the start of Fig 7(c), there is a spike in the number of accepted requests for *GCI_D*, which is due to its focus on fairness. Even though the number of accepted requests by *GCI_D* exceeds *ILP* in the final phase of Fig 7(c), their objective is to maximize the reward, which is defined as the number of accepted requests minus the number of activated servers. Therefore, *ILP* remains optimal overall.

Impact of delay demands on performance of different algorithms: We apply the parameter setting **Set 6** form Table 3, where we increase the delay demand from 0.5 to 1.3 to assess its impact on performance. As depicted in Fig 8(a), with the increase in delay demand, *GCI* and *ILP* exhibit a downward trend, *GCI_D* shows an upward trend, and *Random* fluctuates irregularly. The rise in accepted requests by *GCI_D* contributes to the observed increase; However, both *ILP* and *GCI* prioritize requests with lower delay, leading to fluctuating total energy consumption within a certain



(a) The number of activated servers (b) The total energy consumption (c) The number of admitted requests (d) The running time

Fig. 8: The performance of different algorithms, by varying the delay demand D_i from 0.5 to 1.3.



(a) The actual energy consumption (b) The actual completion time

Fig. 9: The performance of different algorithms, by varying the DMS request $\mathcal{R}$ from 10 to 30.

range in Fig 8(c), even though the number of accepted requests also increases. As depicted in Fig 8(d), the increase in delay demand relaxes the delay constraints, consequently decreasing the running time of all algorithms.

Impact of DMS request numbers on performances of different algorithms: We conduct the parameter configuration **Set 7** outlined in Table 3 to explore its influence on the algorithm performance in the hardware prototype, where we increase the number of DMS requests from 10 to 30, set the user demand of computing resources to 8, and let 5 edge servers with sufficient energy always to be active. From Fig 9(a), the energy consumption in the real scene of all algorithms increases with the growth of the arrival number of DMS requests, where the energy consumption of *GCI* is lower than that of *Random* and higher than that of *ILP*. As observed in Fig. 9(b), the actual completion time of *GCI* and *ILP* show an increasing trend, while that of *Random* fluctuated within a certain region due to its randomness, where the actual completion time is the sum of the execution time of the algorithm and the time from sending all user requests to receiving the expected results.

6 Conclusion and Future Work

In this paper, we studied continuous service provisioning in a fabric metaverse, by developing a GCI system to enhance immersion, reliability, and energy sustainability. The GCI system included a performance-guaranteed approximate pruner to compress model branches to a specified number of K. We also devised a scheduler with two components: a learning-based activator and a deployment library, which can be tailored to meet various demands by different users. We finally evaluated the performance of the proposed GCI system. Theoretical analysis, empirical simulation, and practical experiment demonstrate that the proposed GCI system and associated algorithms are promising, outperforming the comparison baselines.

As a new frontier in human-centered networking, the fabric metaverse lays the groundwork for extensive real-time hypermodal data collection to support continuous metaverse services. The work in this paper takes an initial step towards this direction. Built upon the work in this paper, several potential topics in fabric metaverse are worthwhile to explore, including privacy-protected data collection and analysis for intelligence fabric, high-accuracy large language model training and inference by fusing massive fabric state (hypermodal) data, low-energy round-the-clock monitoring of mental and physical health information processed by models, and real-time data synchronization of related human digital twins, etc.

Acknowledgement

The authors appreciate the Associate Editor and three anonymous referees for their constructive comments and invaluable suggestions, which have helped us to improve the quality and presentation of the paper greatly. The work by Min Chen was supported by the National Natural Science Foundation of China under Grant No. 62276109, and the work by Weifa Liang was supported by the Research Grants Council (RGC) in Hong Kong under CityU Grant No: 7005845, 8730094, 9043510, and 9380137, respectively.

References

[1] Y. Hao, L. Hu, and M. Chen, "Joint sensing adaptation and model placement in 6g fabric computing," *IEEE Journal on Selected Areas in Communications*, 2023.

[2] M. Chen, J. Ouyang, A. Jian, J. Liu, P. Li, Y. Hao, Y. Gong, J. Hu, J. Zhou, R. Wang *et al.*, "Imperceptible, designable, and scalable braided electronic cord," *Nature Communications*, vol. 13, no. 1, p. 7097, 2022.

[3] X. Shi, Y. Zuo, P. Zhai, J. Shen, Y. Yang, Z. Gao, M. Liao, J. Wu, J. Wang, X. Xu *et al.*, "Large-area display textiles integrated with functional systems," *Nature*, vol. 591, no. 7849, pp. 240–245, 2021.

[4] S. Tuli, G. Casale, L. Cherkasova, and N. R. Jennings, "Deepft: Fault-tolerant edge computing using a self-supervised deep surrogate model," in *IEEE INFOCOM 2023 - IEEE Conference on Computer Communications*, 2023.

[5] K. Huang and W. Gao, "Real-time neural network inference on extremely weak devices: Agile offloading with explainable ai," ser. MobiCom '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 200–213. [Online]. Available: https://doi.org/10.1145/3495243.3560551

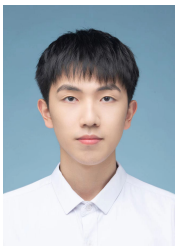
[6] H. Ma, R. Li, X. Zhang, Z. Zhou, and X. Chen, "Reliability-aware online scheduling for dnn inference tasks in mobile edge computing," *IEEE Internet of Things Journal*, 2023.
[7] Y. Qiu, J. Liang, V. C. M. Leung, X. Wu, and X. Deng, "Online reliability-enhanced virtual network services provisioning in fault-prone mobile edge cloud," *IEEE Transactions on Wireless Communications*, vol. 21, no. 9, pp. 7299–7313, 2022.
[8] Y. Qiu, J. Liang, V. C. Leung, and M. Chen, "Online security-aware and reliability-guaranteed ai service chains provisioning in edge intelligence cloud," *IEEE Transactions on Mobile Computing*, pp. 1–16, 2023.
[9] G. Liu, F. Dai, X. Xu, X. Fu, W. Dou, N. Kumar, and M. Bilal, "An adaptive dnn inference acceleration framework with end–edge–cloud collaborative computing," *Future Generation Computer Systems*, vol. 140, pp. 422–435, 2023.
[10] H. Ma, Z. Zhou, X. Zhang, and X. Chen, "Towards carbon-neutral edge computing: Greening edge ai by harnessing spot and future carbon markets," *IEEE Internet of Things Journal*, 2023.
[11] S. Hu, M. Li, J. Gao, C. Zhou, and X. S. Shen, "Digital twin-assisted adaptive dnn inference in industrial internet of things," in *GLOBECOM 2022-2022 IEEE Global Communications Conference*. IEEE, 2022, pp. 1025–1030.
[12] F. Romero, Q. Li, N. J. Yadwadkar, and C. Kozyrakis, "{INFaaS}: Automated model-less inference serving," in *2021 USENIX Annual Technical Conference (USENIX ATC 21)*, 2021, pp. 397–411.
[13] J. Zhu, J. Liu, S. Yang, Q. Zhang, and X. He, "Open benchmarking for click-through rate prediction," in *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, 2021, pp. 2759–2769.
[14] W. Wu, P. Yang, W. Zhang, C. Zhou, and X. Shen, "Accuracy-guaranteed collaborative dnn inference in industrial iot via deep reinforcement learning," *IEEE Transactions on Industrial Informatics*, vol. 17, no. 7, pp. 4988–4998, 2020.
[15] Y. Tang, N. Zhou, Q. Yu, D. Wu, C. Hou, G. Tao, and M. Chen, "Intelligent fabric enabled 6g semantic communication system for in-cabin scenarios," *IEEE Transactions on Intelligent Transportation Systems*, vol. 24, no. 1, pp. 1153–1162, 2022.
[16] Y. Kaya, S. Hong, and T. Dumitras, "Shallow-deep networks: Understanding and mitigating network overthinking," in *International conference on machine learning*. PMLR, 2019, pp. 3301–3310.
[17] S. Laskaridis, S. I. Venieris, H. Kim, and N. D. Lane, "Hapi: Hardware-aware progressive inference," in *Proceedings of the 39th International Conference on Computer-Aided Design*, 2020, pp. 1–9.
[18] J. Jo, G. Kim, S. Kim, and J. Park, "Locoexnet: Low-cost early exit network for energy efficient cnn accelerator design," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2023.
[19] C. Hu and B. Li, "Distributed inference with deep learning models across heterogeneous edge devices," in *IEEE INFOCOM 2022-IEEE Conference on Computer Communications*. IEEE, 2022, pp. 330–339.
[20] A. Yousefpour, S. Devic, B. Q. Nguyen, A. Kreidieh, A. Liao, A. M. Bayen, and J. P. Jue, "Guardians of the deep fog: Failure-resilient dnn inference from edge to cloud," in *Proceedings of the first international workshop on challenges in artificial intelligence and machine learning for internet of things*, 2019, pp. 25–31.
[21] J. Huang, Y. Gao, and W. Dong, "Elastic dnn inference with unpredictable exit in edge computing," in *2023 IEEE 43rd International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2023.
[22] S. Laskaridis, S. I. Venieris, M. Almeida, I. Leontiadis, and N. D. Lane, "SPINN: Synergistic Progressive Inference of Neural Networks over Device and Cloud," in *The 26th Annual International Conference on Mobile Computing and Networking (MobiCom)*, 2020, pp. 1–15.
[23] B. Mao, F. Tang, Y. Kawamoto, and N. Kato, "Ai models for green communications towards 6g," *IEEE Communications Surveys & Tutorials*, vol. 24, no. 1, pp. 210–247, 2022.
[24] Y. Zhu and S. Wang, "Joint traffic prediction and base station sleeping for energy saving in cellular networks," in *ICC 2021-IEEE International Conference on Communications*. IEEE, 2021, pp. 1–6.
[25] Z. Shen and G. Zhang, "Two-timescale mobile user association and hybrid generator on/off control for green cellular networks with energy storage," *IEEE Transactions on Vehicular Technology*, vol. 71, no. 10, pp. 11 047–11 059, 2022.
[26] L. Gu, W. Zhang, Z. Wang, D. Zeng, and H. Jin, "Service management and energy scheduling toward low-carbon edge computing," *IEEE Transactions on Sustainable Computing*, vol. 8, no. 1, pp. 109–119, 2022.
[27] C.-S. Yang, C. Lin, and I.-K. Fu, "Carbon-neutralized joint user association and base station switching for green cellular networks," in *2023 IEEE International Conference on Communications Workshops (ICC Workshops)*, 2023, pp. 1594–1599.
[28] X. Ren, W. Liang, and W. Xu, "Data collection maximization in renewable sensor networks via time-slot scheduling," *IEEE Transactions on Computers*, vol. 64, no. 7, pp. 1870–1883, 2015.
[29] P. C. Fonseca and E. S. Mota, "A survey on fault management in software-defined networks," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 4, pp. 2284–2321, 2017.
[30] R. Cohen, L. Katzir, and D. Raz, "An efficient approximation for the generalized assignment problem," *Information Processing Letters*, vol. 100, no. 4, pp. 162–166, 2006.
[31] F. Tian, X. Zhang, J. Liang, and Z. Yang, "Bidirectional service function chain embedding for interactive applications in mobile edge networks," *IEEE Transactions on Mobile Computing*, pp. 1–17, 2023.
[32] J. Li, W. Liang, Y. Li, Z. Xu, X. Jia, and S. Guo, "Throughput maximization of delay-aware dnn inference in edge computing by exploring dnn model partitioning and inference parallelism," *IEEE Transactions on Mobile Computing*, vol. 22, no. 5, pp. 3017–3030, 2023.
[33] M. F. Cloutier, C. Paradis, and V. M. Weaver, "A raspberry pi cluster instrumented for fine-grained power measurement," *Electronics*, vol. 5, no. 4, 2016.
[34] H. Wu, L. Chen, C. Shen, W. Wen, and J. Xu, "Online geographical load balancing for energy-harvesting mobile edge computing," in *2018 IEEE International Conference on Communications (ICC)*, 2018, pp. 1–6.
[35] E. Lattanzi, C. Contoli, and V. Freschi, "Do we need early exit networks in human activity recognition?" *Engineering Applications of Artificial Intelligence*, vol. 121, p. 106035, 2023.
[36] D. Anguita, A. Ghio, L. Oneto, X. Parra, J. L. Reyes-Ortiz *et al.*, "A public domain dataset for human activity recognition using smartphones." in *Esann*, vol. 3, 2013, p. 3.
[37] E. Eldele, Z. Chen, C. Liu, M. Wu, C.-K. Kwoh, X. Li, and C. Guan, "An attention-based deep learning approach for sleep stage classification with single-channel eeg," *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, vol. 29, pp. 809–818, 2021.
[38] X. Shang, Y. Huang, Z. Liu, and Y. Yang, "Reducing the service function chain backup cost over the edge and cloud by a self-adapting scheme," *IEEE Transactions on Mobile Computing*, vol. 21, no. 8, pp. 2994–3008, 2022.
[39] C. U. Manual, "Ibm ilog cplex optimization studio," *Version*, vol. 12, no. 1987-2018, p. 1, 1987.

Yu Qiu received the BSc degree from the Tianjin University of Technology, Tianjin, China, in 2020, the ME degree in computer technology at the Guangxi University, Nanning, China, in 2023. He is currently working toward the Ph.D. degree in computer science and technology at the South China University of Technology, Guangzhou, China. His research interests include metaverse, edge intelligence, fabric computing, and network function virtualization.

Min Chen (Fellow, IEEE) has been a Full Professor with School of Computer Science and Engineering, South China University of Technology. He is also the director of Embedded and Pervasive Computing (EPIC) Lab at Huazhong University of Science and Technology. He is the founding Chair of IEEE Computer Society Special Technical Communities on Big Data. He was an assistant professor in School of Computer Science and Engineering at Seoul National University before he joined HUST. He is the Chair of IEEE Globecom 2022 eHealth Symposium. His Google Scholar Citations reached 40,000+ with an h-index of 96. His top paper was cited 4,304+ times. He was selected as Highly Cited Researcher from 2018 to 2022. He got IEEE Communications Society Fred W. Ellersick Prize in 2017, the IEEE Jack Neubauer Memorial Award in 2019, and IEEE ComSoc APB Oustanding Paper Award in 2022. He is a Fellow of IEEE and IET.

Weifa Liang (Senior Member, IEEE) received the PhD degree from the Australian National University in 1998, the ME degree from the University of Science and Technology of China in 1989, and the BSc degree from Wuhan University, China in 1984, all in Computer Science. He is a Full Professor in the Department of Computer Science at City University of Hong Kong. Prior to that, he was a Full Professor in the Australian National University. His research interests include design and analysis of energy efficient routing protocols for wireless ad hoc and sensor networks, Mobile Edge Computing (MEC), Network Function Virtualization (NFV), Internet of Things and digital twins, design and analysis of parallel and distributed algorithms, approximation algorithms, and graph theory. He currently serves as an Editor of IEEE Transactions on Communications.

Dusit Niyato (Fellow, IEEE) is a professor in the School of Computer Science and Engineering, at Nanyang Technological University, Singapore. He received B.Eng. from King Mongkuts Institute of Technology Ladkrabang (KMITL), Thailand in 1999 and Ph.D. in Electrical and Computer Engineering from the University of Manitoba, Canada in 2008. His research interests are in the areas of sustainability, edge intelligence, decentralized machine learning, and incentive mechanism design.



Yue Wang is currently working toward the BSc degree in the South China University of Technology, Guangzhou, China, in 2024. His current research interests include mobile edge computing, fabric computing, and multimodal.

Yizhe Li received the B.E degree in 2023, and he is currently working toward a Master degree in computer technology at the South China University of Technology, Guangzhou, China. His research interests include edge intelligence and fabric computing.

Victor C.M. Leung [Life Fellow, IEEE] received the B.A.Sc. (Hons.) degree in electrical engineering from the University of British Columbia (UBC), Vancouver, BC, Canada, in 1977, and the Ph.D. degree in electrical engineering from UBC in 1982. He is a Distinguished Professor of Computer Science and Software Engineering at Shenzhen University, China. He is also an Emeritus Professor of Electrical and Computer Engineering and Director of the Laboratory for Wireless Networks and Mobile Systems at the UBC. He is serving on the editorial boards of the IEEE Transactions on Green Communications and Networking, IEEE Transactions on Cloud Computing, IEEE Access, IEEE Network, and several other journals. He is a Life Fellow of IEEE, and a Fellow of the Royal Society of Canada (Academy of Science), Canadian Academy of Engineering, and Engineering Institute of Canada. He is named in the current Clarivate Analytics list of “Highly Cited Researchers”.

Yixue Hao is an Associate Professor in the School of Computer Science and Technology at Huazhong University of Science and Technology (HUST). He received the Ph.D degree in computer science from HUST, Wuhan, China, in 2017. He was named in Clarivate Analytics Highly Cited Researchers List in 2020. His current research interests include cognitive computing, edge computing, and multi-agent reinforcement learning.

Long Hu is an Associate Professor with the School of Computer Science and Technology, Huazhong University of Science and Technology (HUST), China. His research interests include the Internet of Things, software-defined networking, caching, 5G, body area networks, body sensor networks, and mobile cloud computing.

Yin Zhang (Senior Member, IEEE) is currently a Full Professor with the School of Information and Communication Engineering, University of Electronic Science and Technology of China, Chengdu, China. He is the Co-Chair of the IEEE Computer Society Big Data Special Technical Community (STC). He serves as an Editor or an Associate Editor for IEEE Network, IEEE SYSTEMS JOURNAL, Information Fusion, and Journal of Circuits Systems and Computers.