Energy-Efficient Computing Using Adaptive Model Selection for Mobile Agents in Federated Learning

Benteng Zhang, *Student Member, IEEE*, Yingchi Mao, *Member, IEEE*, Tianfu Pang, *Student Member, IEEE*, Zhenxiang Pan, Jianxin Huang, Feng Mao, Xiaoming He, *Member, IEEE*, and Jie Wu, *Fellow, IEEE*

***Abstract*—Multi-agent federated learning requires all agents to stay online until the global model converges, and its multi-round training incurs substantial on-agent computing. However, most edge mobile agents with limited computational capacity and battery life may drop out of FL training prematurely due to low battery, which can degrade the convergence speed and accuracy of the global model. Prior works assign heterogeneous models to different agents to reduce on-agent computational energy, but neglect the changes in data distributions and computational capacities on agents. This makes it difficult to save on-agent computational energy and to support global aggregation. To this end, we propose a hierarchical Federated learning framework based on Adaptive Local Training (FedALT). Specifically, FedALT can adaptively select the most suitable local model for each agent based on its computational capacity and data distribution to reduce on-agent computational energy. To achieve finer-grained savings in computational energy, FedALT dynamically adjusts the number of local epochs for each agent during local training. Furthermore, FedALT conducts edge aggregation through model layer sharing and knowledge distillation to improve the convergence speed and accuracy of the global model. We compare FedALT with three baselines by training six models on two Non-IID datasets. Experiments on both the simulation platform and the hardware platform demonstrate that FedALT can achieve the lowest computational energy, the highest average global model accuracy, and the fastest global model convergence.**

***Index Terms*—Multi-agent federated learning, edge computing, mobile agent, model selection**

I. Introduction

In multi-agent Federated Learning (FL), each edge device is modeled as an autonomous agent that enables on-device Artificial Intelligence (AI) services and coordinates with others to train a global AI model [1], [2]. However, state-of-the-art studies have indicated that most existing FL methods are based on an ideal assumption, e.g., agents in FL can stay online throughout training until the global model converges [3]. In real-world scenarios [4], before the global model converges, agents must perform high-frequency local epochs to update local models on their own datasets, and the multi-round training during FL incurs substantial on-agent computing [5]. However, most edge mobile agents that participate in FL, such as smartphones, laptops, and smartwatches [6], have limited computational capacity and battery life, which are sensitive to battery energy consumption [7]. These mobile agents may drop out of FL training prematurely due to low battery, which can degrade the convergence speed and accuracy of the global model [8]. To address this issue, prior works assign heterogeneous local models to different agents to reduce on-agent computational energy [9]. However, data distributions on agents always change as fresh data arrives (Non-IID scenarios), and the computational capacities of agents fluctuate with their remaining battery. Therefore, as illustrated in ***Challenge 1*** of Fig. 1, heterogeneous local models can not match the changes in data distributions and computational capacities on agents, leading to significant update drifts among local models.

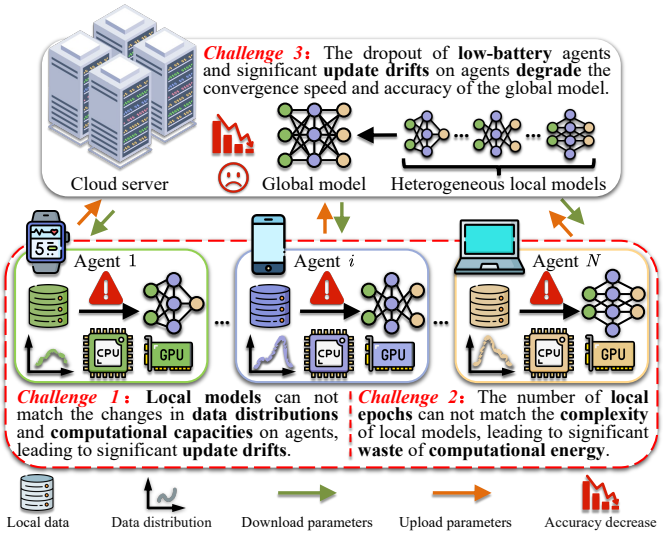


Fig. 1. Three challenges in saving computational energy for mobile agents.

Existing methods reduce the update drift among heterogeneous local models by optimizing the architecture of the Neural Networks (NNs) for agents. For example, FedEMD partitions the model into a backbone (composed of the initial layers) and a personalized-head (formed by the remaining

This work was supported in part by the National Natural Science Foundation of China under Grant 62572171; in part by the Fundamental Research Funds for the Central Universities under Grant B250205031; in part by the 16th Batch of Science and Technology Development Programs (Innovation Consortium Projects) of Suzhou under Grant LHT202404; in part by the Technology Talent and Platform Program of Yunnan Province under Grant 202405AK340002; in part by the Technology Project of Huaneng Group under Grant HNKJ24-H167; and in part by the High Performance Computing Platform, Hohai University. *(Corresponding author: Yingchi Mao.)*

Yingchi Mao, Benteng Zhang, Tianfu Pang, and Zhenxiang Pan are with the College of Computer Science and Software Engineering, Hohai University, Nanjing 211100, China (e-mail: {yingchimao; 230407040003; 250207010004; 240407010002}@hhu.edu.cn).

Jianxin Huang and Feng Mao are with the Suma Information Industry Co., Ltd, Suzhou 215300, China (e-mail: huangjx@cancon.com.cn; 15800653230@163.com).

Xiaoming He is with the College of Internet of Things, Nanjing University of Posts and Telecommunications, Nanjing 210003, China (e-mail: hexiaoming@njupt.edu.cn).

Jie Wu is with the China Telecom Cloud Computing Research Institute and Temple University, Philadelphia, PA 19122, USA (e-mail: jiewu@temple.edu).

layers) [10]. The personalized-head is retained locally during FL training and is not uploaded to the cloud server for global aggregation. Based on Quantized Neural Networks (QNNs), Kim *et al.* designed a local training energy consumption strategy by quantizing model weights and activations using fixed-precision formats [11]. Furthermore, to reduce uploaded data and energy consumption, Lei *et al.* proposed the energy-efficient FL scheme LDES by deploying Sparse or Binary Neural Networks (S/BNNs) on agents for local training [12]. However, these methods above focus on heterogeneous model architectures but neglect the relationship between the complexities of local models and the number of local epochs. To our best knowledge, reducing local model complexity may larger the update drifts among heterogeneous models, while too few local epochs may weaken the model's learning ability, which can save computational energy but degrade the global model accuracy. Therefore, as illustrated in ***Challenge 2*** of Fig. 1, the number of local epochs in these methods can not match the complexities of local models, leading to significant waste of computational energy. Only by matching model complexities with appropriate local epochs can the trade-off between computational energy and model accuracy be balanced. As a result, it can be seen in ***Challenge 3*** of Fig. 1, the dropout of low-battery agents (due to *Challenges 1* and *2*) and significant update drifts on agents (due to *Challenge 1*) degrade the convergence speed and accuracy of the global model, which remains a huge gap in FL that needs to be filled.

Motivated by the challenges mentioned above, our goal is to dynamically select the most suitable model architecture and local epochs for each agent to reduce on-agent computational energy, while effectively reducing update drifts among heterogeneous local models to better support global aggregation. To this end, we propose a hierarchical **Fed**erated learning framework based on **A**daptive **L**ocal **T**raining (FedALT).

The **contributions** of this paper are depicted as follows.

- **Cloud server side.** On the cloud server, FedALT builds an adaptive model-selection mechanism to select the most suitable model architecture for each agent based on its computational capacity and data distribution, which can reduce on-agent computational energy.
- **Edge server side.** On each edge server, FedALT conducts a dual knowledge-transfer mechanism by model layer sharing and knowledge distillation to reduce update drifts among local models, which can improve the convergence speed and accuracy of the global model.
- **Agent side.** On each agent, FedALT designs a dynamic epoch-adjustment mechanism to adjust the number of local epochs during local training, which can save on-agent computational energy in a fine-grained manner.
- **Effectiveness.** Experiments on both platforms show that compared to baselines, FedALT can achieve the lowest computational energy, the highest average global model accuracy, and the fastest global model convergence.

The remainder of this paper is as follows. Section II presents the related works. Section III gives the proposed framework. Section IV describes the design details of FedALT. Sections V and VI evaluate FedALT. Section VII concludes this paper.

II. Related Work

A. Select Heterogeneous Models for Agents

In traditional Hierarchical Federated Learning (HFL), a uniform model architecture is typically assigned to different agents for local training [13]. Due to agent heterogeneity (i.e., different hardware and computational capacities among agents), a single model architecture cannot adapt to the heterogeneous computational capacities of agents [14]. This can cause the computational energy consumed by some agents to be significantly higher than that of other agents. To reduce the local model complexity and computational energy, prior works assign heterogeneous model architectures to different agents according to their computational capacity. HeteroFL [9] utilizes subsets of the global model as local models on agents, which allows for the assignment of heterogeneous models to agents with different computational capacities by scaling the input/output dimensions of the hidden layers. This enables the local models to match the computational capacity of agents and reduces computational energy consumption. At the same time, HeteroFL designs a heterogeneous-model aggregation strategy that allows heterogeneous local models among agents to be aggregated into a unified global model. By introducing multiple different model architectures, HAF-Edge [15] can preassign the model architecture that each agent will train based on the computational capacity of agents before FL begins. Thus, HAF-Edge can lower the computational complexity and reduce computational energy on agents.

However, to our best knowledge, the heterogeneous-model selection strategies in the above methods do not consider the dynamic changes in computational capacities and data distributions on agents, which may lead to significant waste of computational energy, larger update drifts among heterogeneous models, and degrade the global model accuracy.

B. Adjust Local Training Epochs on Agents

Beyond reducing the local model complexity, prior works have reduced the computational energy consumed during local training by adjusting the number of local epochs on different agents. For example, Wang *et al.* [16] proposed an epoch-control algorithm that can optimize on-agent computational energy in resource-constrained edge computing systems by adjusting local epochs and global aggregation frequency. FedProx [17] adds a proximal term to the local loss to limit the divergence between local and global models, which enables agents to dynamically adjust their local epochs according to local training conditions and thus save computational energy. Although FedProx can mitigate the update drift caused by model heterogeneity, FedProx also constrains the local model's ability to learn from its local dataset. Heterogeneous data distributions across agents may require local models to capture fine-grained features, yet the proximal term weakens learning on local datasets and can degrade global model accuracy.

Moreover, we find that existing methods for adjusting local epochs do not consider the relationship between the complexities of heterogeneous models and the number of local epochs. Heterogeneous models with different levels of complexity require different local epochs to achieve the target learning

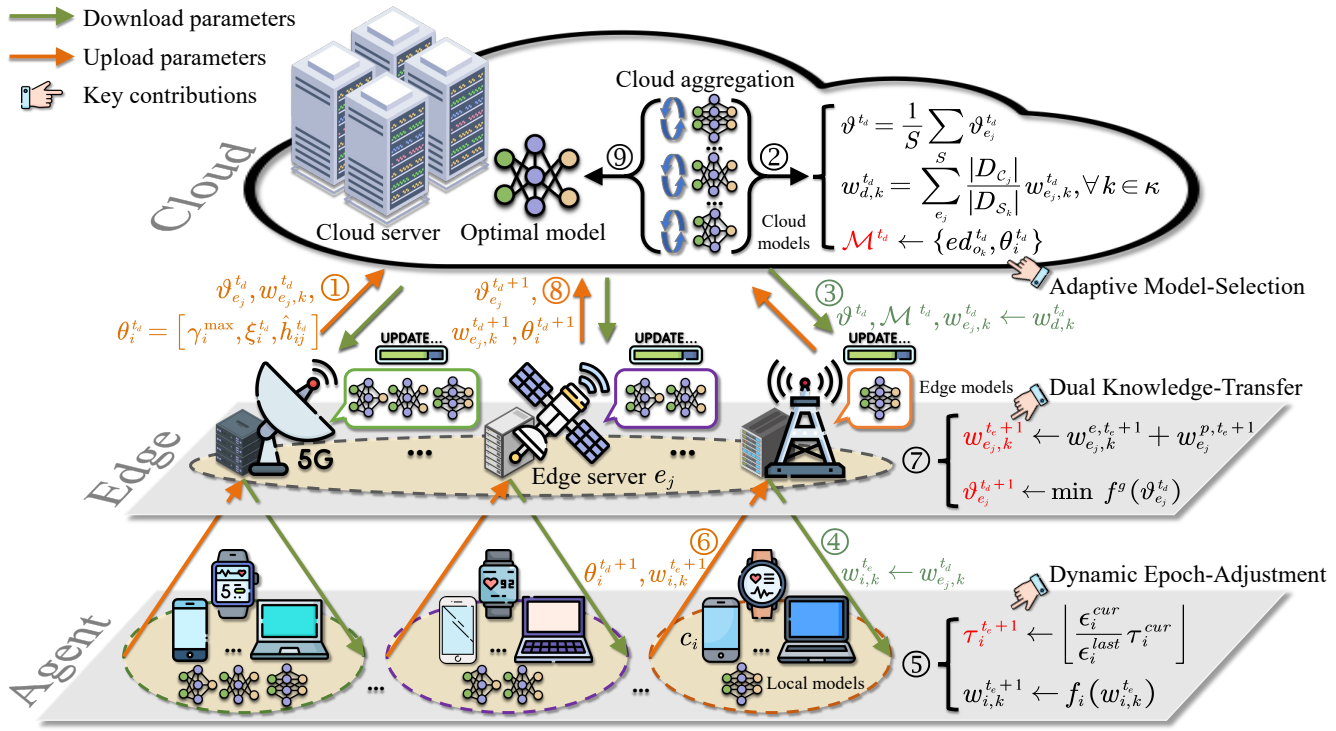


Fig. 2. The overview of FedALT. ① Upload the edge model, edge generator, and data distribution metric. ② Aggregate cloud models, update the matrix $\mathcal{M}^{t_d}$ and cloud generator ϑ^{t_d}. ③ Send cloud models to edge servers. ④ Send edge models to mobile agents. ⑤ Local training and adjust local epochs. ⑥ Upload the local model and data distribution metric. ⑦ Knowledge distillation, aggregate edge models, and update the edge generator. ⑧ Upload the new edge model, edge generator, and data distribution metric. ⑨ Select the optimal cloud model as the final global model.

effectiveness [18]. This makes it difficult for us to dynamically increase or decrease the number of local epochs based on model complexities, and to precisely save the computational energy consumed by agents. Therefore, designing an epoch-adjustment mechanism that can dynamically sense the changes in model complexities still remains a major challenge [19].

C. Knowledge Transfer among Heterogeneous Models

To reduce update drifts among local models and aggregate heterogeneous models [20], prior works have utilized Knowledge Distillation (KD) to transfer knowledge among heterogeneous models [21], [22], which can improve the convergence speed and accuracy of the global model. For example, FedMD allows different agents to hold heterogeneous local models and aggregates the logits of local models on their local datasets to obtain the shared knowledge [23]. The heterogeneous models use this shared knowledge as supervision for local knowledge fusion, instead of exchanging and aggregating model parameters. Jiang *et al.* proposed HDHRFL [24], which extracts the logits that heterogeneous models output on a public dataset and achieves knowledge transfer among heterogeneous models by bringing their logits closer to each other.

However, we find that FedMD and HDHRFL rely heavily on public datasets. In real-world scenarios, public datasets in certain fields are often unavailable or difficult to obtain, and public data can even introduce additional noise [25]. Therefore, how to conduct a knowledge-transfer mechanism that does not rely on public datasets is a key challenge to improve the efficiency of KD among heterogeneous models.

III. Proposed Framework

A. On-Agent Local Training

As shown in Fig. 2, our proposed HFL framework consists of κ types of model architectures. In the t_d-th cloud training round, we define a matrix $\mathcal{M}^{t_d} \in \mathbb{R}^{m\times\kappa}$ to represent the model architecture category trained by each mobile agent locally, and $\{m_{ik} = 1\} \in \mathcal{M}^{t_d}$ indicates that mobile agent $c_i \in \mathcal{C}$ trains the k-th type of model architecture, $\forall i \in \{1,\dots,N\}$, $\forall k \in \{1,\dots,\kappa\}$. In the t_e-th edge training round, $w_{i,k}^{t_e}$ is the local model of agent c_i that trains the k-th type of model architecture on its local dataset D_i within edge server $e_j \in \mathcal{S}$, $\forall j \in \{1,\dots,S\}$. During local training, the local loss $f_i(w_{i,k}^{t_e})$ of agent c_i can be defined as

$$f_i(w_{i,k}^{t_e}) = \frac{1}{|D_i|}\sum\nolimits_{(x_j,y_j)}^{D_i} \mathcal{L}(\hat{y}(x_j, w_{i,k}^{t_e}), y_j), \tag{1}$$

where $\hat{y}(x_j, w_{i,k}^{t_e})$ is the prediction of $w_{i,k}^{t_e}$ on sample x_j, and $\mathcal{L}(\cdot)$ is the loss between the data label y_j and the prediction $\hat{y}$. The optimization objective of our HFL is to obtain the optimal global model w^f, i.e., to minimize the average loss of w^f over the loss functions $f_i(\cdot)$ on all agents, which is given by

$$\min_{w^f} \frac{1}{N}\sum\nolimits_{c_i}^{\mathcal{C}} f_i(w^f). \tag{2}$$

During local training, in the t_c-th local epoch, agent c_i utilizes Stochastic Gradient Descent (SGD) [26] to update the local model $w_{i,k}^{t_c}$ for the (t_c+1)-th epoch, which is given by

$$w_{i,k}^{t_c+1} = w_{i,k}^{t_c} - \eta\nabla f_i(w_{i,k}^{t_c}), \tag{3}$$

TABLE I
LIST OF MAIN SYMBOLIC PARAMETERS

Symbols	Descriptions
x	The sample category
κ	Total types of model architectures
k	The k-th type of model with dimension d_k
c_i	The i-th mobile agent in $\mathcal{C}$, $\|\mathcal{C}\| = N$
e_j	The j-th edge server in $\mathcal{S}$, $\|\mathcal{S}\| = S$
t_c	The c-th local training epoch
t_e	The e-th edge training round
t_d	The d-th cloud training round
w^f	The final global model
n_x	The number of samples of category x
γ_i	CPU frequency of mobile agent c_i
δ_i	CPU effective switching capacitance
$\xi_i^{t_d}$	CPU utilization of mobile agent c_i
D_i	Local dataset of mobile agent c_i
ϑ^{t_d}	The generator model with dimension v
$\tau_i^{t_e}$	Total local training epochs
$\epsilon_i^{t_e}$	The average execution time of all local epochs
$\hat{h}_{ij}^{t_d}$	The data distribution metric
$\mathcal{M}^{t_d}$	Matrix records the types of models trained by agents
$\pi(\cdot)$	CPU cycles for c_i to process a single sample
$f_i(\cdot)$	The loss function of mobile agent c_i
$f^g(\cdot)$	The loss function of generator
$f_k^{kd}(\cdot)$	The distillation loss of k-th type of edge model
p_i^{cpu}	CPU power of mobile agent c_i
$w_{d,k}^{t_d}$	Cloud model in the t_d-th cloud training round
$w_{i,k}^{t_e}$	Local model on c_i in the t_e-th edge training round
$w_{e_j,k}^{t_e}$	Edge model on e_j in the t_e-th edge training round
$E_{i,k}^{comp}$	The computational energy on mobile agent c_i

where $t_c = 1, 2, ..., \tau_i^{t_e}$, η is the learning rate, and $\nabla f_i(\cdot)$ is the gradient calculated by the local loss $f_i(\cdot)$.

B. Edge Aggregation of Heterogeneous Local Models

After mobile agent c_i completes $\tau_i^{t_e}$ local training epochs, agent c_i uploads its local model $w_{i,k}^{t_e+1} \leftarrow w_{i,k}^{\tau_i^{t_e}+1}$ to its associated edge server e_j. Then, the edge server e_j performs a weighted aggregation of local models uploaded by all agents within the group of e_j based on their model architecture. Therefore, the k-th edge model $w_{e_j,k}^{t_e+1}$ on edge server e_j for the (t_e+1)-th edge training round can be denoted as

$$w_{e_j,k}^{t_e+1} = \sum\nolimits_{c_i}^{\mathcal{C}_j^k} \frac{|D_i|}{|D_{\mathcal{C}_j^k}|} w_{i,k}^{t_e+1}, \tag{4}$$

where $\mathcal{C}_j^k$ is the set of agents within e_j that train the k-th type of model architecture, and $|D_{\mathcal{C}_j^k}|$ is the total dataset of agents in $\mathcal{C}_j^k$. After this, edge server e_j sends $w_{e_j,k}^{t_e+1}$ to the corresponding agents and initiates the (t_e+1)-th edge training round until T_e edge training rounds are completed.

C. Cloud Aggregation of Heterogeneous Edge Models

In the t_d-th cloud training round, after edge server e_j completes T_e edge training rounds, e_j uploads κ edge models $w_{e_j,k}^{t_d+1} \leftarrow w_{e_j,k}^{T_e+1}$ to the cloud server. The cloud server then performs a weighted aggregation of the κ edge models according to their model architecture. Therefore, we can obtain the cloud model $w_{d,k}^{t_d+1}$ of the k-th type of model architecture for the (t_d+1)-th cloud training round, which is given by

$$w_{d,k}^{t_d+1} = \sum\nolimits_{e_j}^{\mathcal{S}} \frac{|D_{\mathcal{C}_j}|}{|D_{\mathcal{S}_k}|} w_{e_j,k}^{t_d+1}, \tag{5}$$

where $|D_{\mathcal{S}_k}|$ is the total dataset of agents that train the same k-th type of model architecture. The cloud server sends $w_{d,k}^{t_d+1}$ to the corresponding agents based on $\mathcal{M}^{t_d}$ and initiates the (t_d+1)-th cloud training round. After T_d cloud training rounds, we choose the k-th cloud model $w_{d,k}^{T_d+1}$ that shows the highest accuracy among κ cloud models as the final global model w^f.

D. On-Agent Computational Energy Consumption

In each local training epoch, we denote $\pi(w_{i,k}^{t_e})$ as the number of CPU cycles required by agent c_i to process a single data sample, which trains the k-th type of model architecture in the t_e-th edge training round [27]. As the complexity of the local model architecture on agent c_i increases, $\pi(w_{i,k}^{t_e})$ typically exhibits a significant upward trend [28]. We denote γ_i as the CPU frequency of agent c_i. Then, the time required to execute one local training epoch using the k-th type of model architecture can be calculated by

$$\epsilon_{i,k}^{comp} = \frac{1}{\gamma_i} \pi(w_{i,k}^{t_e}) |D_i|. \tag{6}$$

Consistent with [29], we believe that agent c_i can adjust its CPU frequency within $[\gamma_i^{min}, \gamma_i^{max}]$ by using Dynamic Voltage and Frequency Scaling (DVFS) [30]. Therefore, the CPU power p_i^{cpu} of agent c_i can be denoted as

$$p_i^{cpu} = \delta_i \gamma_i^3, \tag{7}$$

where δ_i is the effective switching capacitance of the CPU on agent c_i. According to (6) and (7), the computational energy consumption on agent c_i for training the k-th model architecture in one local training epoch is given by

$$E_{i,k}^{comp} = p_i^{cpu} \times \epsilon_{i,k}^{comp} = \delta_i \pi(w_{i,k}^{t_e}) |D_i| \gamma_i^2. \tag{8}$$

Since we employ the variable local training epoch $\tau_i^{t_e}$ to achieve finer-grained savings in computational energy consumed by agents during local training. Therefore, in the t_d-th cloud training round, the total computational energy consumed by all agents during local training can be calculated by

$$E_{t_d}^{comp} = \sum\nolimits_{t_e=1}^{T_e} \sum\nolimits_{i=1}^{m} \sum\nolimits_{k=1}^{\kappa} m_{ik} \tau_i^{t_e} E_{i,k}^{comp}. \tag{9}$$

E. Problem Formulation

According to (8), it can be seen that we can reduce the computational energy on agent c_i during one local training epoch by reducing the complexity of the local model $w_{i,k}^{t_e}$. Furthermore, according to (9), we can adjust the number of local training epochs for agent c_i while reducing the local model complexity, which can further save the computational energy consumed by local training. Therefore, our final optimization objective is given by

$$\min\nolimits_{\mathcal{M}^{t_d}, \tau_i^{t_e}} E^{comp} \tag{10a}$$

$$\text{s.t.} \quad \sum\nolimits_{i=1}^{m} m_{ik} > 0, \quad \forall k \in \{1, \ldots, \kappa\}, \tag{10b}$$

$$\sum\nolimits_{k=1}^{\kappa} m_{ik} = 1, \quad \forall c_i \in \mathcal{C}, \tag{10c}$$

$$\tau_i^{\min} \leqslant \tau_i^{t_e} \leqslant \tau_i^{\max}, \quad \forall c_i \in \mathcal{C}, \tag{10d}$$

Algorithm 1: Adaptive Model-Selection (Cloud)

Input: $\mathcal{S}$, $\mathcal{C}$, D, ϑ^1, T_d, T_e, $\tau_i^{\max}$, $\tau_i^{\min}$, learning rate η, and κ cloud models $w_{d,k}^1$

Output: The final global model w^f

for $t_d = 1, 2, ..., T_d$ **do**
- Each edge server e_j calculates $\hat{h}_{ij}^{t_d}$ using (12) and uploads $[\gamma_i^{\max}, \xi_i^{t_d}, \hat{h}_{ij}^{t_d}]$ to the cloud server;
- Cloud server calculates $ed_{o_k}^{t_d}$ using (14);
- Cloud server generates $\mathcal{M}^{t_d}$ based on $ed_{o_k}^{t_d}$ and sends $[\vartheta^{t_d}, w_{d,k}^{t_d}, \mathcal{M}^{t_d}]$ to all edge servers;
- $EdgeTraining(\vartheta^{t_d}, w_{d,k}^{t_d}, \mathcal{M}^{t_d})$;
- All edge servers upload κ edge models $w_{e_j,k}^{t_d+1}$ and edge generators $\vartheta_{e_j}^{t_d+1}$ to the cloud server;
- Cloud server aggregates $w_{d,k}^{t_d+1}$ using (5);
- Cloud server aggregates ϑ^{t_d+1} using (23);

end

Cloud server selects the optimal cloud model from κ cloud models $w_{d,k}^{T_d+1}$ as the final global model w^f;

return w^f

where (10b) indicates that, in each cloud training round, every model architecture must be held by at least one mobile agent. (10c) stipulates that the local model of a mobile agent must belong to exactly one model architecture. (10d) specifies that the number of local training epochs performed by any mobile agent can not exceed the maximum epochs $\tau_i^{\max}$ and can not be less than the minimum epochs $\tau_i^{\min}$. The main symbolic parameters are reported in Table I.

IV. Adaptive Local Training and Heterogeneous Model Aggregation

A. Adaptive Model-Selection Mechanism on Cloud Server

According to [31], we also assume that all mobile agents can sustainably provide their maximum computational capacity in the absence of external interference. Therefore, we use the maximum CPU frequency $\gamma_i^{\max}$ of agent c_i to represent its peak computational capacity, where a higher $\gamma_i^{\max}$ indicates stronger computational capacity. Influenced by environmental factors, task load, and battery, the real-time available computational resources of a mobile agent may vary dynamically [32]. Thus, we use the CPU utilization $\xi_i^{t_d}$ of agent c_i to reflect its current available computational resources. A lower $\xi_i^{t_d}$ indicates a lower current computational load and more abundant remaining resources. To account for the changes in local data distribution, we employ a data distribution metric $\hat{h}_{ij}$ based on class information features to represent the quality of the local data distribution on a mobile agent set $\mathcal{C}_j$ within edge server e_j. Therefore, the data distribution of $\mathcal{C}_j$ is denoted as

$$dis\left(D_{\mathcal{C}_j}\right) = \{(x, n_x) \mid x \in \{1, 2, \cdots\}\}, \tag{11}$$

where $D_{\mathcal{C}_j} = \bigcup_{c_i}^{\mathcal{C}_j} D_i$ is the total dataset for $\mathcal{C}_j$, x is the sample category, and n_x is the number of samples of category x. Therefore, the data distribution metric $\hat{h}_{ij}^{t_d}$ in the t_d-th cloud training round can be calculated by

$$\hat{h}_{ij}^{t_d} = \sum\nolimits_{(x,n_x)}^{dis\left(D_{\mathcal{C}_j} \cup D_i\right)} -p_x \log_2 p_x, \tag{12}$$

where $p_x = n_x / |D_{\mathcal{C}_j} \cup D_i|$ is the probability of category x samples appearing in $D_{\mathcal{C}_j} \cup D_i$. Note that the larger the value of $\hat{h}_{ij}$, the better the data distribution on c_i. In the t_d-th cloud training round, each mobile agent uploads its $\gamma_i^{\max}$, $\xi_i^{t_d}$, and $\hat{h}_{ij}$ to the cloud server via the edge server. The computational capacity representation vector θ_i is defined as

$$\theta_i^{t_d} = [\gamma_i^{\max}, \xi_i^{t_d}, \hat{h}_{ij}^{t_d}]. \tag{13}$$

We use the K-means [33] to partition agents in $\mathcal{C}_j$ into κ clusters $\{o_k \mid k \in \{1, 2, \ldots, \kappa\}\}$, and denote θ_{o_k} as the centroid of cluster o_k. We define a computational capacity anchor as $\theta^a = [\gamma^{\max}, \xi^{\min}, \hat{h}^{\max}]$, where $\gamma^{\max}$, $\xi^{\min}$, and $\hat{h}^{\max}$ represent the maximum CPU frequency, the minimum CPU utilization, and the best data distribution metric in $\mathcal{C}_j$, respectively. We compute the distance ed_{o_k} between each cluster centroid θ_{o_k} and the computational capacity anchor θ^a using the L_2-norm, which is given by

$$ed_{o_k}^{t_d} = \|\theta_{o_k} - \theta^a\|_2. \tag{14}$$

Then, we allocate κ types of model architectures to different agents within each cluster in ascending order of $ed_{o_k}^{t_d}$. In the t_d-th cloud training round, agents with the smallest $ed_{o_k}^{t_d}$ use the highest-complexity model architecture for local training, and agents with the largest ed_{o_k} use the lowest-complexity model architecture. In this step, we can obtain the matrix $\mathcal{M}^{t_d}$ of model-architecture categories held by all agents.

Privacy Guarantee. It can be observed that the metric $\hat{h}_{ij}^{t_d}$ in vector θ_i transmitted between the agents and the cloud server may potentially expose sensitive information from local data. In fact, the metric $\hat{h}_{ij}^{t_d}$ does not directly expose the raw local data, feature vectors, labels, or sample-level records of any mobile agent. Instead, $\hat{h}_{ij}^{t_d}$ is a highly compressed scalar statistic computed from the joint class distribution of $D_{\mathcal{C}_j} \cup D_i$, and thus only reflects the coarse-grained diversity and balance of the local data distribution. Since the mapping from high-dimensional local data to a single entropy-based value is many-to-one and non-invertible, $\hat{h}_{ij}^{t_d}$ cannot be used to reconstruct the original samples or infer fine-grained local data contents. In this sense, $\hat{h}_{ij}^{t_d}$ serves only as a distribution-quality indicator for model assignment and scheduling, rather than as a carrier of recoverable local data information. Therefore, compared with transmitting raw data or gradients, the privacy leakage risk introduced by $\hat{h}_{ij}^{t_d}$ is limited to coarse statistical characteristics and is substantially reduced.

B. Dynamic Epoch-Adjustment Mechanism on Agents

In each cloud training round, the computational capacity of mobile agents may fluctuate due to task load or battery [34], which prevents heterogeneous model architectures from matching each agent's computational capacity. To this end, FedALT introduces a dynamic epoch-adjustment mechanism for agents. By adaptively adjusting the local epochs on each agent and collaborating with the adaptive model-selection mechanism, the local model architectures can more accurately match the agent's computational capacity. Therefore, we can further save the computational energy consumed by agents during local training. In the t_e-th edge training round, when

Algorithm 2: Dual Knowledge-Transfer (Edge)

Input: ϑ^{t_d}, $w_{d,k}^{t_d}$, $\mathcal{M}^{t_d}$
Output: edge models $w_{e_j,k}^{t_d+1}$, edge generators $\vartheta_{e_j}^{t_d+1}$

function $EdgeTraining(\vartheta^{t_d}, w_{d,k}^{t_d}, \mathcal{M}^{t_d})$;
for *each edge server* $e_j \in \mathcal{S}$ **do**
  κ edge models $w_{e_j,k}^{1} \leftarrow w_{d,k}^{t_d}$ and $\vartheta_{e_j}^{1} \leftarrow \vartheta^{t_d}$;
  for $t_e = 1, 2, ..., T_e$ **do**
    e_j sends $w_{e_j,k}^{t_e}$ to all agents based on $\mathcal{M}^{t_d}$;
    $LocalTraining(w_{e_j,k}^{t_e})$;
    All agents upload the local model $w_{i,k}^{t_e+1}$ to e_j;
    e_j aggregates $w_{e_j,k}^{e,t_e+1}$ and $w_{e_j}^{p,t_e+1}$ of κ edge models using (17) and (18);
    e_j aggregates κ edge models $w_{e_j,k}^{t_e+1}$ using (4);
    e_j updates edge generator $\vartheta_{e_j}^{t_d+1}$ using (22);
  end
  $w_{e_j,k}^{t_d+1} \leftarrow w_{e_j,k}^{T_e+1}$;
end
return $w_{e_j,k}^{t_d+1}$, $\vartheta_{e_j}^{t_d+1}$

agent c_i within the group of edge server e_j performs local training, we record the average execution time $\epsilon_i^{t_e}$ of all local training epochs, which is given by

$$\epsilon_i^{t_e} = \frac{1}{\tau_i^{t_e}} \sum\nolimits_{t_c=1}^{\tau_i^{t_e}} \epsilon_i^{t_c}, \tag{15}$$

where $\epsilon_i^{t_c}$ is the execution time on agent c_i in the t_c-th local training epoch. We denote $\tau_i^{cur} = \tau_i^{t_e}$ as the total number of local training epochs on agent c_i, denote $\epsilon_i^{cur} = \epsilon_i^{t_e}$ as the average execution time of all local training epochs, and denote $\epsilon_i^{last} = \epsilon_i^{t_e-1}$ as the average execution time in the previous edge training round [35]. Then, we can dynamically adjust the number of local training epochs $\tau_i^{t_e+1}$ for agent c_i in the (t_e+1)-th edge training round, which is given by

$$\tau_i^{t_e+1} = \max(\min(\lfloor \frac{\epsilon_i^{last}}{\epsilon_i^{cur}} \tau_i^{cur} \rfloor, \tau_i^{\max}), \tau_i^{\min}), \tag{16}$$

where $\lfloor \cdot \rfloor$ is the floor operation. During edge training, if the computational capacity of an agent is degraded, FedALT can adaptively reduce the number of local epochs for the next edge training round, which can save more computational energy consumed by agents. Conversely, when an agent's computational capacity improves, FedALT can automatically increase the number of local epochs, which can better help local models to learn the characteristics of the local dataset. Note that we initialize the number of local epochs $\tau_i^1 = \tau_i^{\max}$ for agent c_i in the first cloud training round.

C. Dual Knowledge-Transfer Mechanism on Edge Servers

1) Partial Model Layer Sharing. In the t_e-th edge training round, the local model $w_{i,k}^{t_e+1}$ on mobile agent c_i consists of a feature-extractor $w_{i,k}^{e,t_e+1}$ and a prediction-head w_i^{p,t_e+1}. Note that the feature-extractor architectures differ across agents, while the prediction-head maintains a globally unified architecture. Thus, the k-th type of edge models $w_{e_j,k}^{t_e+1}$ aggregated by edge server e_j is constructed from the k-th type of edge feature-extractors $w_{e_j,k}^{e,t_e+1}$ together with a shared edge prediction-head $w_{e_j}^{p,t_e+1}$. Specifically, as shown in step ⑦ of Fig. 2, during aggregating edge models, edge server e_j receives κ types of local model architectures uploaded by agents within its group. Edge server e_j first performs weighted aggregation on the feature-extractor of the k-th type of local models to obtain the k-th edge feature-extractor $w_{e_j,k}^{e,t_e+1}$ for the (t_e+1)-th edge training round, which is given by

$$w_{e_j,k}^{e,t_e+1} = \sum\nolimits_{c_i}^{\mathcal{C}_j^k} \frac{|D_i|}{|D_{\mathcal{C}_j^k}|} w_{i,k}^{e,\tau_i^{t_e}+1}, \tag{17}$$

where $w_{i,k}^{e}$ is the k-th type of feature-extractor uploaded by agent c_i. Then, the edge server e_j performs a weighted aggregation on the shared prediction-head of the κ types of local models to obtain the edge prediction-head $w_{e_j}^{p,t_e+1}$ for the (t_e+1)-th edge training round, which is given by

$$w_{e_j}^{p,t_e+1} = \sum\nolimits_{c_i}^{\mathcal{C}_j} \frac{|D_i|}{|D_{\mathcal{C}_j}|} w_i^{p,\tau_i^{t_e}+1}. \tag{18}$$

2) Data-Free Knowledge Distillation. In the t_d-th cloud training round, to improve the effectiveness of edge model aggregation, FedALT deploys a generator model $\vartheta_{e_j}^{t_d}$ on edge server e_j. Specifically, after aggregating κ types of edge models, the edge server e_j treats the k-th type of edge model $w_{e_j,k}^{t_e+1}$ as the student model and the other $\kappa-1$ edge models as the teacher model. Based on augmented samples $\tilde{x}$ generated by the generator $\vartheta_{e_j}^{t_d}$, FedALT can distill the teachers' knowledge into the student model. The generator $\vartheta_{e_j}^{t_d}$ takes noise $\mathcal{N}$ and sample label y as inputs and generates the augmented sample $\tilde{x}$, where the sample label y can be drawn from the overall data distribution of edge server e_j. For the k-th type of edge model $w_{e_j,k}^{t_e+1}$, we first use the edge generator $\vartheta_{e_j}^{t_d}$ to generate the augmented sample $\tilde{x}$ with label y. Then, we use the edge model $w_{e_j,k}^{t_e+1}$ and the other $\kappa-1$ edge models to predict on the sample $\tilde{x}$, and the outputs of the prediction-heads are passed through the *softmax* function to obtain soft labels [36]. Next, we utilize the *Kullback-Leibler* (KL) divergence to measure the discrepancy between the soft label output by the edge model $w_{e_j,k}^{t_e+1}$ and the mean soft label output by the other $\kappa-1$ edge models [37]. Therefore, the distillation loss $f_k^{kd}(w_{e_j,k}^{t_e+1})$ for the k-th type of edge model $w_{e_j,k}^{t_e+1}$ can be defined as

$$\begin{aligned} &f_k^{kd}(w_{e_j,k}^{t_e+1}) = \\ &\mathcal{L}_{KL}(\sigma(\tilde{x}, w_{e_j,k}^{t_e+1}), \frac{1}{\kappa-1} \sum\nolimits_{k}^{K_{k-1}} \sigma(\tilde{x}, w_{e_j,k}^{t_e+1})), \end{aligned} \tag{19}$$

where, $\mathcal{L}_{KL}$ is the KL loss, $\sigma(\cdot)$ is the soft labels output by the model on the augmented sample $\tilde{x}$, and K_{k-1} is the index set of the other $\kappa-1$ edge models. In the t_e-th edge training round, according to $f_k^{kd}(w_{e_j,k}^{t_e+1})$, the optimization objective of edge server e_j can be defined as

$$\min\nolimits_{w_{e_j}^{t_e+1}} \frac{1}{\kappa} \sum\nolimits_{k=1}^{\kappa} f_k^{kd}(w_{e_j,k}^{t_e+1}). \tag{20}$$

As for the edge generator $\vartheta_{e_j}^{t_d}$, we expect the augmented sample $\tilde{x}$ generated by $\vartheta_{e_j}^{t_d}$ to correspond to the input label y, so that $\tilde{x}$ can be correctly predicted by the edge models.

Algorithm 3: Dynamic Epoch-Adjustment (Agent)

Input: $w_{e_j,k}^{t_e}$
Output: local model $w_{i,k}^{t_e+1}$ of each agent c_i
function $LocalTraining(w_{e_j,k}^{t_e})$;
for *each mobile agent* $c_i \in \mathcal{C}_j$ **do**
  Agent c_i set local model parameters $w_{i,k}^1 = w_{e_j,k}^{t_e}$;
  for $t_c = 1, 2, ..., \tau_i^{t_e}$ **do**
    c_i trains local model $w_{i,k}^{t_c}$ on local dataset D_i;
  end
  $w_{i,k}^{t_e+1} \leftarrow w_{i,k}^{\tau_i^{t_e}}$;
  c_i calculates the average execution time $\epsilon_i^{t_e}$ of all local training epochs using (15);
  c_i updates local training epochs $\tau_i^{t_e+1}$ using (16);
end
return $w_{i,k}^{t_e+1}$

Therefore, we define a generator loss $f^g(\vartheta_{e_j}^{t_d})$ to measure the discrepancy between the predictions and the expected label $\hat{y}$ of the κ edge models on $\tilde{x}$, which is defined as

$$f^g(\vartheta_{e_j}^{t_d}) = \frac{1}{\kappa}\sum\nolimits_{k=1}^{\kappa} \mathcal{L}(\hat{y}(\tilde{x}, w_{e_j,k}^{t_e+1}), y). \tag{21}$$

Thus, the optimization objective of the edge generator $\vartheta_{e_j}^{t_d}$ on edge server e_j can be defined as

$$\min\nolimits_{\vartheta_{e_j}^{t_d}} f^g(\vartheta_{e_j}^{t_d}). \tag{22}$$

In each edge training round, edge servers aggregate local models and utilize SGD to optimize the edge generators. When edge server e_j completes T_e edge training rounds, e_j uploads both κ edge models and the generator $\vartheta_{e_j}^{t_d}$ to the cloud server. The cloud server aggregates κ edge models to obtain κ cloud models for the (t_d+1)-th cloud training round based on (5). Meanwhile, the cloud server aggregates the edge generator to obtain the cloud generator ϑ^{t_d+1} for the (t_d+1)-th cloud training round, which is given by

$$\vartheta^{t_d+1} = \frac{1}{S}\sum\nolimits_{e_j}^{\mathcal{S}} \vartheta_{e_j}^{t_d}. \tag{23}$$

D. Algorithm Design Details of FedALT

The design details of FedALT are shown in **Algorithms 1-3**. FedALT can be divided into three key stages as follows.

1) Cloud Training. **Algorithm 1** shows the adaptive model-selection mechanism on the cloud server. In the t_d-th cloud training round, after agent c_i calculates the local data distribution metric $\hat{h}_{ij}$, c_i uploads its CPU maximum frequency $\gamma_i^{\max}$, CPU utilization $\xi_i^{t_d}$, and $\hat{h}_{ij}$ to the cloud server (line 2). The cloud server dynamically generates the matrix $\mathcal{M}^{t_d}$. Next, the cloud server sends the κ cloud models and the generator ϑ^{t_d} to the corresponding edge servers for edge training based on $\mathcal{M}^{t_d}$ (lines 3–4). After edge training, all edge servers upload κ edge models $w_{e_j,k}^{T_e+1}$ and generators $\vartheta_{e_j}^{t_d+1}$ to the cloud server (line 6). The cloud server aggregates κ cloud models $w_{d,k}^{t_d+1}$ and the generator ϑ^{t_d+1} for the (t_d+1)-th cloud training round (lines 7–8). After T_d cloud training rounds, the cloud server selects the optimal model with the highest accuracy from κ cloud models $w_{d,k}^{T_d+1}$ as the final global model w^f (line 10).

2) Edge Training. **Algorithm 2** shows the dual knowledge-transfer mechanism on edge servers. Before edge training, all edge servers first initialize the edge model parameters (line 3). In the t_e-th edge training round, edge server e_j first sends κ edge models $w_{e_j,k}^{t_e}$ to all agents based on the matrix $\mathcal{M}^{t_d}$ (line 5). After local training, all agents upload the local model $w_{i,k}^{t_e+1}$ to e_j (line 7). Then, edge server e_j aggregates the feature-extractor $w_{e_j,k}^{e,t_e+1}$ and prediction-head $w_{e_j}^{p,t_e+1}$ of κ edge models (line 8). Finally, edge server e_j aggregates κ edge models $w_{e_j,k}^{t_e+1}$ for the (t_e+1)-th edge training round and updates the generator $\vartheta_{e_j}^{t_d}$ (lines $9-10$).

3) Local Training. **Algorithm 3** shows the dynamic epoch-adjustment mechanism on agents. Before local training, all agents first initialize the local model parameters (line 3). In the t_c-th local training epoch, agent c_i trains the k-th type of local model $w_{i,k}^{t_c}$ on the local dataset D_i (lines $4-7$). Then, agent c_i calculates the average execution time $\epsilon_i^{t_e}$ of all epochs and updates the number of epochs $\tau_i^{t_e+1}$ for the (t_e+1)-th edge training round (lines $8-9$).

E. Convergence Analysis of FedALT

Multi-level aggregation and multi-timescale updates in FedALT introduce multiple sources of drift across agents, edge servers, and the cloud server. As these characteristics invalidate the assumptions of existing convergence results for traditional FL, a convergence analysis is required to ensure the stability and theoretical soundness of the proposed FedALT. To prove the convergence of FedALT, we need to prove **Lemmas 1-5** and **Theorem 1**. We give **Assumptions 1-4**.

Assumption 1 (*L*-Smoothness). *We assume that for the local objective function $F_i(\cdot)$ of agent c_i, there exists a constant $L>0$ that can satisfy:*

$$\|\nabla F_i(x) - \nabla F_i(y)\| \le L\|x-y\|, \quad \forall x, y. \tag{24}$$

Assumption 1 arises from the fundamental optimization assumption that the objective function is differentiable and has bounded gradient variation. For neural networks, linear models, convex loss functions, and most differentiable empirical risk objectives, it is typically assumed that the local gradient does not change abruptly. Hence, a Lipschitz-continuous gradient is used to characterize this property. **Assumption 1** is a key prerequisite for deriving the descent **Lemma 5**. Without smoothness, the objective function may become extremely steep locally, and even after parameter updates, the function value may not be stably controlled. In that case, it would be impossible to prove that each iteration of FedALT is "overall descending," let alone establish a global convergence bound.

Assumption 2 (Unbiased Gradient). *We assume that the local SGD on agent c_i can satisfy:*

$$\mathbb{E}_{\xi\sim D_i}\left[\nabla f(w,\xi)\right] = \nabla F_i(w). \tag{25}$$

Assumption 2 is one of the core definitions of SGD. Since agents usually have access only to local mini-batch data or even a single sample, stochastic gradient estimates are used

instead of the full gradient. **Assumption 2** ensures that the gradient estimate in local SGD points in the correct direction in expectation and does not deviate from the true gradient. Thus, when deriving the expected decrease, the stochastic gradient term can be decomposed into the true gradient and noise, and the noise can be handled separately. If the gradient estimate is biased, then even after repeated updates, FedALT may accumulate deviations in the wrong direction.

Assumption 3 (**Bounded Variance**). *We assume that there exists a constant σ_i^2 that can satisfy:*

$$\mathbb{E}\|\nabla f(w,\xi) - \nabla F_i(w)\|^2 \le \sigma_i^2. \tag{26}$$

Assumption 3 is a control assumption on the magnitude of the stochastic sampling noise. Because each agent estimates the gradient using only local samples, estimation error is inevitably present; therefore, it is usually required that its second moment be bounded. **Assumption 3** is used to control the stochastic fluctuations introduced by SGD, ensuring that the noise arising from single-sample/small-batch updates does not grow without bound. In the convergence proof, all noise terms must ultimately be reduced to a finite constant in order to derive a finite upper bound on the average gradient norm. Without a variance bound, the fluctuation magnitude of stochastic updates cannot be controlled. As a result, even if each step is correct on average, the one-step oscillation may still be very large, making it impossible to prove convergence in expectation, let alone establish a convergence rate.

Assumption 4 (**Bounded Heterogeneity**). *We assume that the local objective function $F_i(\cdot)$ of agent c_i, the edge objective function $F_{e_j,k}(\cdot)$ of edge server e_j, and the cloud objective function $F_{d,k}(\cdot)$ of the k-th model can satisfy:*

$$\|\nabla F_i(w) - \nabla F_j(w)\|^2 \le \zeta_{\text{dev}}^2, \tag{27a}$$

$$\|\nabla F_{e_j,k}(w) - \nabla F_{e_v,k}(w)\|^2 \le \zeta_{\text{edge}}^2. \tag{27b}$$

$$\|\nabla F_{d,k}(w) - \nabla F(w)\|^2 \le \zeta_{\text{cloud}}^2. \tag{27c}$$

Assumption 4 is a key assumption that distinguishes FL from centralized training. Data distributions among agents are usually heterogeneous. In particular, under Non-IID settings, local gradients are not consistent across agents. Therefore, a unified upper bound is needed to characterize such discrepancies. **Assumption 4** is used to quantify agent drift, edge drift, and the deviations introduced by multi-level aggregation. In our proposed three-layer framework, this assumption is especially important because the agent layer, edge layer, and cloud layer may all introduce additional bias. Only by constraining these biases within a finite range can they be incorporated as error terms rather than treated as divergence sources that prevent convergence. Without a heterogeneity bound, it is impossible to guarantee that updates from different agents will not severely conflict with one another. In particular, under a structure with multiple local epochs, multi-edge aggregation, and parallel cloud models, local updates may continuously accumulate deviations from the global objective, causing the convergence analysis to lose closure.

Since our proposed FedALT consists of three layers (i.e., agent layer, edge layer, and cloud layer), there are three drifts between the local model and the global model. First, we need to decompose these drifts, and we give **Lemma 1**.

Lemma 1 (**Global Gradient Drift Decomposition**). *Under **Assumptions 1–4**, the stochastic gradient g^{t_d} used in the t_d-th cloud training round satisfies the following decomposition:*

$$\mathbb{E}\left\|g^{t_d} - \nabla F(w^{t_d})\right\|^2 \le 3(E_{\text{dev}} + E_{\text{edge}} + E_{\text{cloud}}). \tag{28}$$

Proof. According to **Assumption 4**, the global gradient drift of FedALT consists of the local training drift E_{dev}, the edge aggregation drift E_{edge}, and the cloud aggregation drift E_{cloud}, which are given by

$$E_{\text{dev}} = \sum\nolimits_{c_i}^{\mathcal{C}} \alpha_i^2\, \mathbb{E}\left\|\nabla F_i(w_i^{t_e}) - \nabla F_i(w^{t_d})\right\|^2, \tag{29a}$$

$$E_{\text{edge}} = \sum\nolimits_{e_j}^{\mathcal{S}} \beta_j^2\, \mathbb{E}\left\|\nabla F_{e_j}(w_{e_j}^{t_e}) - \nabla F_{e_j}(w^{t_d})\right\|^2, \tag{29b}$$

$$E_{\text{cloud}} = \frac{1}{\kappa}\sum\nolimits_{k=1}^{\kappa} \mathbb{E}\|\nabla F_{d,k}(w_{d,k}^{t_d}) - \nabla F(w^{t_d})\|^2. \tag{29c}$$

Therefore, the global gradient drift of FedALT is bounded and decomposed into three parts. **Lemma 1** concludes. □

Now, we have proved that the global gradient drift is bounded by E_{dev}, E_{edge}, and E_{cloud}. Next, we need to prove that these three drifts have upper bounds so that we can prove the convergence of FedALT. We give **Lemmas 2-4**.

Lemma 2 (**Upper Bounds of Local Training Drift**). *Under **Assumptions 1–4** and learning rate η, the local training drift E_{dev} in **Lemma 1** satisfies the following bounds:*

$$E_{\text{dev}} \le 2L^2\eta^2 \sum\nolimits_{c_i}^{\mathcal{C}} \alpha_i^2.\tau_i^{t_e}\left(\sigma_i^2 + \|\nabla F_i(w^{t_d})\|^2\right). \tag{30}$$

Proof. According to **Assumption 1** and (29a), the local training drift E_{dev} can satisfy:

$$E_{\text{dev}} \le L^2 \sum\nolimits_{c_i}^{\mathcal{C}} \alpha_i^2 \mathbb{E}\|w_i^{t_e} - w^{t_d}\|^2. \tag{31}$$

Since each agent trains $\tau_i^{t_e}$ local epochs, we have

$$w_i^{t_e} = w_{e_j}^{t_e} - \eta \sum\nolimits_{t_c=1}^{\tau_i^{t_e}} \nabla f(w_i^{t_c}). \tag{32}$$

Due to $w_{e_j}^{t_e} = w^{t_d} - \eta(g^{t_e})$, we can get

$$w_i^{t_c} - w^{t_d} = -\eta \sum\nolimits_{t_c=1}^{\tau_i^{t_e}} \nabla f(w_i^{t_c}) - \eta(g^{t_e}). \tag{33}$$

Taking the square norm and applying the boundedness of expectation and variance, we can get

$$\mathbb{E}\|w_i^{t_e} - w^{t_d}\|^2 \le 2\eta^2\tau_i^{t_e}\mathbb{E}\left\|\nabla F_i(w^{t_d})\right\|^2 + 2\eta^2\tau_i^{t_e}\sigma_i^2. \tag{34}$$

Put (34) into (31), we can get (30). **Lemma 2** concludes. □

Lemma 3 (**Upper Bounds of Edge Aggregation Drift**). *Under **Assumptions 1–4** and learning rate η, the edge aggregation drift E_{edge} in **Lemma 1** satisfies the following bounds:*

$$E_{\text{edge}} \le L^2\eta^2 T_e \sum\nolimits_{e_j}^{\mathcal{S}} \beta_j^2(\sigma_{e_j}^2 + \zeta_{\text{edge}}^2 + \|\nabla F(w^{t_d})\|^2). \tag{35}$$

Proof. According to **Assumption 1** and (29b), the edge aggregation drift E_{edge} can satisfy:

$$E_{\text{edge}} \leq L^2 \sum\nolimits_{e_j}^{\mathcal{S}} \beta_j^2 \mathbb{E}\|w_{e_j}^{t_e} - w^{t_d}\|^2. \tag{36}$$

Since E_{edge} comes from T_e edge training rounds, we have

$$w_{e_j}^{t_e} = w^{t_d} - \eta \sum\nolimits_{t_e=1}^{T_e} \nabla F_{e_j}(w_{e_j}^{t_e}). \tag{37}$$

Then, we can get

$$\mathbb{E}\|w_{e_j}^{t_e} - w^{t_d}\|^2 \leq \eta^2 T_e(\sigma_{e_j}^2 + \zeta_{\text{edge}}^2 + \|\nabla F(w^{t_d})\|^2). \tag{38}$$

Put (38) into (36), we can get (35). **Lemma 3** concludes. □

Lemma 4 (Upper Bounds of Cloud Aggregation Drift). *Under **Assumptions 1–4**, the cloud aggregation drift E_{cloud} in **Lemma 1** satisfies the following bounds:*

$$E_{\text{cloud}} \leq \zeta_{\text{cloud}}^2. \tag{39}$$

Proof. According to **Assumption 4** and **Lemma 1**, put (27c) into (29c), we can get (39). **Lemma 4** concludes. □

According to **Lemmas 1-4**, we have proved that the global gradient drift is bounded and has an upper bound. Then, we need to prove that the global gradient is descending during T_d cloud training rounds so that the global model can converge to an optimal point. We give **Lemma 5**.

Lemma 5 (Global Gradient Descent). *Under **Assumption 1**, the global gradient of w^{t_d} is descending during T_d cloud training rounds, which satisfies the following:*

$$\begin{aligned}\mathbb{E}[F(w^{t_d+1})] \leq \mathbb{E}[F(w^{t_d})] - \frac{\eta}{2}\mathbb{E}\|\nabla F(w^{t_d})\|^2 \\ + \frac{\eta}{2}\mathbb{E}\|g^{t_d} - \nabla F(w^{t_d})\|^2 + \frac{L\eta^2}{2}\mathbb{E}\|g^{t_d}\|^2.\end{aligned} \tag{40}$$

Proof. According to **Assumption 1**, the cloud model w^{t_d+1} is updated as follows:

$$w^{t_d+1} = w^{t_d} - \eta g^{t_d}. \tag{41}$$

Based on the standard descent lemma, we have

$$F(w^{t_d+1}) \leq F(w^{t_d}) - \eta\langle \nabla F(w^{t_d}), g^{t_d}\rangle + \frac{L\eta^2}{2}\|g^{t_d}\|^2. \tag{42}$$

Then, we can get

$$\begin{aligned}\langle \nabla F(w^{t_d}), g^{t_d}\rangle = \|\nabla F(w^{t_d})\|^2 \\ - \langle \nabla F(w^{t_d}), \nabla F(w^{t_d}) - g^{t_d}\rangle.\end{aligned} \tag{43}$$

Put (43) into (42), we can get the core descent inequality:

$$\begin{aligned}\mathbb{E}[F(w^{t_d+1})] \leq \mathbb{E}[F(w^{t_d})] - \eta\mathbb{E}\|\nabla F(w^{t_d})\|^2 \\ + \eta\mathbb{E}\langle \nabla F(w^{t_d}), \nabla F(w^{t_d}) - g^{t_d}\rangle + \frac{L\eta^2}{2}\mathbb{E}\|g^{t_d}\|^2.\end{aligned} \tag{44}$$

According to Young's Inequality, we have

$$\eta\langle \nabla F, \nabla F - g\rangle \leq \frac{\eta}{2}\|\nabla F\|^2 + \frac{\eta}{2}\|\nabla F - g\|^2. \tag{45}$$

Put (45) into (44), we can get the final descending form (40). Therefore, **Lemma 5** concludes. □

Until now, we have proved that the global gradient drift has an upper bound and the global gradient is descending during T_d cloud training rounds. Finally, we can prove the convergence of FedALT. We give **Theorem 1**.

Theorem 1 (Convergence of FedALT). *If **Assumptions 1-4** hold and the learning rate $\eta < \frac{1}{6L^2T_e}$, then FedALT can converge to an optimal point and satisfy the following bounds:*

$$\frac{1}{T_d}\sum_{t_d=1}^{T_d}\mathbb{E}\|\nabla F(w^{t_d})\|^2 \leq O(\frac{1}{\eta T_d}) + O(\eta T_e) + O(\eta). \tag{46}$$

Proof. According to **Lemmas 1-5**, put (30), (35), and (39) into (40), after merging the same $\|\nabla F(w^{t_d})\|^2$ item, we have

$$\begin{aligned}\mathbb{E}[F(w^{t_d+1})] \leq \mathbb{E}[F(w^{t_d})] + C_1\eta^3 + C_2\eta^3 T_e + C_3\eta \\ - \eta(\frac{1}{2} - 3L^2\eta T)\mathbb{E}\|\nabla F(w^{t_d})\|^2,\end{aligned} \tag{47}$$

where the constant $C_1 = 3L^2\sum_i \alpha_i^2\tau_i^{\max}\sigma_i^2$ comes from the local drift, $C_2 = 3L^2\sum_j \beta_j^2\sigma_{e_j}^2$ comes from the edge drift, and $C_3 = 3L^2\zeta_{\text{cloud}}^2$ comes from the cloud drift. We use $O(\cdot)$ to represent the key variable in (47) more concisely as follows:

$$\begin{aligned}\mathbb{E}[F(w^{t_d+1})] \leq \mathbb{E}[F(w^{t_d})] + O(\eta^3 T_e) + O(\eta) \\ - \eta(\frac{1}{2} - 3L^2\eta T_e)\mathbb{E}\|\nabla F(w^{t_d})\|^2,\end{aligned} \tag{48}$$

where $\frac{1}{2} - 3L^2\eta T_e > 0$ can ensure the effective direction of the gradient descent, i.e. $\eta < \frac{1}{6L^2T_e}$. Since FedALT has T_d cloud training rounds, we sum (48) from $t_d = 1$ to T_d:

$$\begin{aligned}\sum\nolimits_{t_d=1}^{T_d}\eta(\frac{1}{2} - 3L^2\eta T_e)\mathbb{E}\|\nabla F(w^{t_d})\|^2 \\ \leq F(w^1) - F(w^{T_d+1}) + O(T_d\eta).\end{aligned} \tag{49}$$

Divide both sides by $T_d\eta$, we can get (46). Thus, **Theorem 1** concludes. We have proved the convergence of FedALT. □

In summary, under the common assumptions of smoothness, unbiased gradients, bounded variance, and bounded heterogeneity, FedALT is guaranteed to converge to the optimal point as stated in **Theorem 1** when the learning rate $\eta < \frac{1}{6L^2T_e}$.

F. Complexity Analysis of FedALT

Multi-level aggregation and multi-timescale updates in FedALT introduce computational workloads, communication overhead, and storage requirements across different layers [38]. Therefore, a detailed complexity analysis is necessary to quantify the computational cost, memory consumption, and communication overhead of FedALT, and to demonstrate its scalability and practical feasibility in resource-constrained mobile and edge computing environments.

1) Computational Complexity. The cloud server aggregates κ types of edge models and the generators ϑ^{t_d} uploaded by S edge servers, with complexities $\mathcal{O}(\sum_{j=1}^{S}\sum_{k=1}^{\kappa} d_k)$ and $\mathcal{O}(S \cdot v)$, respectively. Since the complexity of calculating ed_{o_k} and generating $\mathcal{M}^{t_d}$ is constant, there are T_d cloud training rounds, and the total cloud computational complexity is $\mathcal{O}(T_d \cdot (\sum_{j=1}^{S}\sum_{k=1}^{\kappa} d_k + Sv))$. There are S edge servers that aggregate N local models with complexity $\mathcal{O}(\sum_{i=1}^{N} d_{i,k})$ and update the generator ϑ^{t_d} for G SGD steps with the complexity

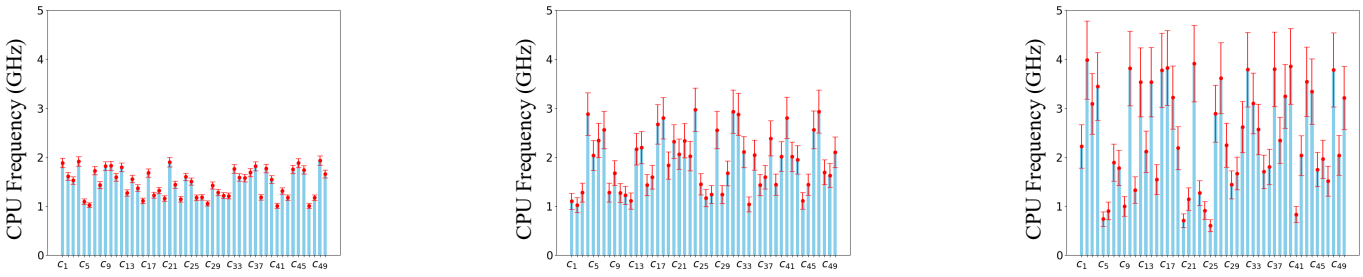


Fig. 3. The distributions and ranges of each agent's CPU frequency γ_i on the simulation platform under three agent heterogeneity scenarios $f = \{1, 2, 3\}$.

$\mathcal{O}(SGv)$. Since there are T_e edge training rounds, and the total edge computational complexity is $\mathcal{O}(T_e \cdot (\sum_{i=1}^{N} d_{i,k} + SGv))$. Each agent updates the local model for $\tau_i^{t_e}$ epochs, and each epoch is trained on the local dataset $|D_i|$. Since the complexity of calculating $\hat{h}_{ij}$ is constant, the total computational complexity of N agents is $\mathcal{O}(\sum_{i=1}^{N} \tau_i^{t_e} |D_i| d_{i,k})$.

2) Storage Complexity. The cloud server needs to cache κ types of edge models uploaded by S edge servers with storage complexity $\mathcal{O}(\sum_{j=1}^{S} \sum_{k=1}^{\kappa} d_k)$. The storage complexity of caching the generators ϑ^{t_d} uploaded by S edge servers on the cloud server is $\mathcal{O}(S \cdot v)$, and the total cloud storage complexity is $\mathcal{O}(\sum_{j=1}^{S} \sum_{k=1}^{\kappa} d_k + Sv)$. All edge servers need to cache N local models uploaded by N agents with storage complexity $\mathcal{O}(\sum_{i=1}^{N} d_{i,k})$. Each edge server must cache one generator ϑ^{t_d} with storage complexity $\mathcal{O}(v)$, and the total edge storage complexity of S edge servers is $\mathcal{O}(\sum_{i=1}^{N} d_{i,k} + Sv)$. Each agent needs to cache one local model with storage complexity $\mathcal{O}(d_{i,k})$. Since the complexities of local SGD and the dataset are constants, the total storage complexity of N agents is $\mathcal{O}(\sum_{i=1}^{N} d_{i,k})$.

3) Communication Complexity. During T_d cloud training rounds, the cloud server sends κ types of cloud models to S edge servers, with a total communication volume $T_d S \sum_{k=1}^{\kappa} d_k$. During T_e edge training rounds, the edge server sends the edge models allocated to each agent within the group, with a total communication volume $\sum_{t_e=1}^{T_e} \sum_{j=1}^{S} \sum_{i \in \mathcal{C}_j} d_{i,k}$. After $\tau_i^{t_e}$ local training epochs, each agent uploads the local model to the corresponding edge server for T_e times, with a total communication volume $T_e \sum_{i=1}^{N} d_{i,k}$. After T_e edge training rounds, S edge server upload κ types of edge models to the cloud server for T_d times, with a total communication volume $\sum_{t_d=1}^{T_d} \sum_{j=1}^{S} \sum_{k=1}^{\kappa} d_k$.

Overall, the computational complexity of FedALT is dominated by local training on mobile agents, which scales linearly with the model size, local dataset size, and the number of local training epochs. The edge layer mainly incurs aggregation and lightweight optimization costs, while the cloud layer introduces additional overhead proportional to the number of edge servers and maintained cloud models. The communication and storage complexities scale linearly with the number of participating agents, edge servers, and model parameters, indicating that FedALT is computationally and communicationally scalable under realistic system settings.

V. Simulation Platform Evaluation

A. Experimental Settings

Simulation Environment. Our simulation experiments run on the simulation platform equipped with 2 NVIDIA *A100* (each with 80GB GPU memory) and 256GB memory, in which simulated nodes, including 1 cloud server node, 5 edge server nodes, and 50 agent nodes, share the platform's resources.

Heterogeneous Computational Capacities on Agents. We set different ranges of CPU frequencies for mobile agents to simulate heterogeneous computational capacities among agent nodes on the simulation platform. We configure three agent heterogeneity scenarios $f = \{1, 2, 3\}$, where a larger f indicates higher agent heterogeneity and more pronounced variation in computational capacity. The distribution and range of the CPU frequency on each agent are shown in Fig. 3.

Non-IID Datasets, Generator Model, and Target Models. All experiments are conducted on both the MNIST and CIFAR-10 datasets. To control the data heterogeneity across devices, we use the *Dirichlet* function $Dir(\alpha)$ to partition both datasets to generate Non-IID local datasets for each device [39]. We configure three data heterogeneity scenarios $\alpha = \{0.1, 1.0, 10\}$, where a smaller α indicates higher data heterogeneity. To ensure the fairness of our experiments, we utilize the same generator model proposed by DaFKD [40]. To provide different model architectures for devices, we train six target models as follows. The details are reported in Table II.

- *MNIST* dataset consists of $70,000$ image [41]. We train a LeNet model, a SmallCNN model, and a TinyCNN model on the MNIST dataset. All three models use the final fully connected layer as the shared prediction-head, whose input and output dimensions are 50 and 10, respectively.
- *CIFAR-10* dataset consists of $60,000$ image [42]. We train a ResNet-18 model [43], a SimpNet model [44], and an EfficientNetV2 model [45] on the CIFAR-10 dataset. All three models use the final fully connected layer as the shared prediction-head, whose input and output dimensions are 512 and 10, respectively.

Hyperparameter Settings. On the simulation platform, we set the total number of mobile agents $N = 50$ and the total number of edge servers $S = 5$. We set the edge training rounds $T_e = 3$, the number of cloud training rounds $T_d = 100$ and $T_d = 200$ on the MNIST dataset and the CIFAR-10 dataset, respectively. For local training, we set the mini-batch size

TABLE II
THE DETAILS OF DATASETS AND TARGET MODELS.

Datasets	Models	GFLOPs*	Data simples (train:test)
MNIST	LeNet	4.8×10^{-4}	70000 (6:1)
	SmallCNN	2.5×10^{-4}	
	TinyCNN	8.7×10^{-5}	
CIFAR-10	ResNet-18	5.6×10^{-1}	60000 (5:1)
	SimpNet	5.9×10^{-2}	
	EfficientNetV2	2.2×10^{-2}	

* GFLOPs denotes the number of floating-point operations required by the corresponding target model to process one single data sample, which can estimate the number of CPU cycles needed to process one data sample.

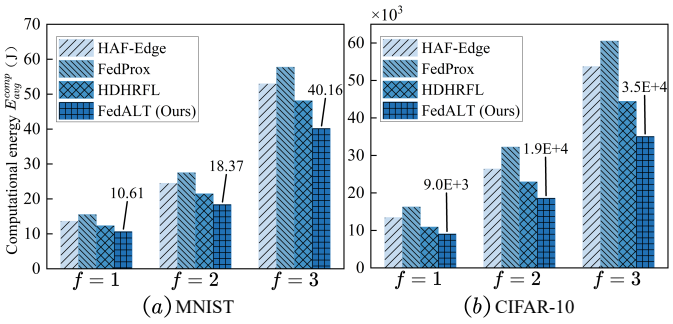


Fig. 4. Average computational energy consumption E_{avg}^{comp} on the simulation platform under high data heterogeneity $\alpha = 0.1$.

$batchsize = 20$, the learning rate $\eta = 0.01$, the maximum and minimum number of local epochs $\tau_i^{\max} = 20$ and $\tau_i^{\min} = 15$, respectively. We follow the same settings of [46] and set the CPU effective switching capacitance $\delta = 10^{-28}$.

Baselines. We choose the three baselines as follows.

- *HAF-Edge* assigns different model architectures to agents before training [15]. The cloud server transfers knowledge by aggregating the overlapping parts.
- *FedProx* incorporates a proximal term into the local loss to reduce the update drifts among local models [17]. Since FedProx can not train heterogeneous models, we train the same model for FedProx (i.e., train LeNet and ResNet-18 on MNIST and CIFAR-10, respectively).
- *HDHRFL* is the state-of-the-art method that trains heterogeneous models on the edge servers [24]. The hidden-layer of the edge model is scaled to obtain local models.

Metrics. Two criteria are used to evaluate FedALT.

- *Average Computational Energy* E_{avg}^{comp}. According to (9), we calculate the average computational energy E_{avg}^{comp} on the simulation platform, which is given by

$$E_{avg}^{comp} = \frac{1}{T_d} \sum\nolimits_{t_d=1}^{T_d} E_{t_d}^{comp}. \tag{50}$$

- *Global Model Accuracy* Acc_g. We test the global model accuracy Acc_g of each method on both datasets.

B. Analysis of E_{avg}^{comp} on the Simulation Platform

On the simulation platform, Fig. 4 shows the average computational energy E_{avg}^{comp} of HAF-Edge, FedProx, HDHRFL, and the proposed FedALT on the MNIST and CIFAR-10 datasets under three agent heterogeneity $f = \{1, 2, 3\}$ and the highest data heterogeneity $\alpha = 0.1$. We find that FedALT can achieve the lowest E_{avg}^{comp} on both datasets.

As shown in Fig. 4, FedALT achieves the lowest average computational energy consumption $E_{avg}^{comp} = 10.61$ J, 18.37 J, and 40.16 J on the MNIST dataset. At the same time, on the CIFAR-10 dataset, FedALT achieves the lowest average computational energy consumption $E_{avg}^{comp} = 9.04$ KJ, 18.58 KJ, and 35.04 KJ. Under the low agent heterogeneity $f = 1$, due to the low CPU frequency of agents and the small differences in CPU frequency, all four methods achieve low E_{avg}^{comp}. The E_{avg}^{comp} of FedALT on the MNIST dataset is 13.81%, 21.99%, and 31.59% lower than that of HDHRFL, HAF-Edge, and FedProx, respectively. The E_{avg}^{comp} of FedALT on the CIFAR-10 dataset is 16.82%, 32.68%, and 44.36% lower than that of HDHRFL, HAF-Edge, and FedProx, respectively. This indicates that FedALT can demonstrate superior performance and save more computational energy when training with more complex datasets. Under agent heterogeneity $f = 2$, the differences in CPU frequency across agents and the range of CPU frequencies both expand. Although the E_{avg}^{comp} of all four methods rises, FedALT can still achieve the lowest $E_{avg}^{comp} = 18.37$ J on the MNIST dataset, which is 14.32%, 24.90%, and 33.20% lower than that of HDHRFL, HAF-Edge, and FedProx, respectively. Meanwhile, the E_{avg}^{comp} of FedALT on the CIFAR-10 dataset is 18.95%, 29.53%, and 42.35% lower than that of HDHRFL, HAF-Edge, and FedProx, respectively. Under the highest agent heterogeneity $f = 3$, compared with HDHRFL, HAF-Edge, and FedProx, the E_{avg}^{comp} of FedALT on the MNIST dataset decreases by approximately 16.47%, 24.15%, and 30.41%, respectively, which still significantly outperforms the suboptimal HDHRFL. Similar results also appear on the CIFAR-10 dataset, compared with HDHRFL, HAF-Edge, and FedProx, the E_{avg}^{comp} of FedALT decreases by approximately 21.08%, 34.73%, and 42.07%, respectively. We can see that the reduction in the E_{avg}^{comp} of FedALT on the CIFAR-10 dataset is significantly greater than the MNIST dataset, and the reduction in the E_{avg}^{comp} of FedALT under $f = 3$ is significantly greater than $f = 1$. This is because the other three baselines are sensitive to dataset complexity and agent heterogeneity, and their performance significantly degrades on the CIFAR-10 dataset.

Since FedProx only adjusts the number of local training epochs and does not consider reducing the complexity of the local models, which leads to the highest E_{avg}^{comp}. Compared to FedProx, HAF-Edge can achieve lower computational energy, which indicates that only adjusting local training epochs cannot save computational energy. However, the static model-assignment strategy in HAF-Edge cannot adapt to the changes in agent computational capacity. As the agent heterogeneity increases, HDHRFL further reduces the complexity of local models based on heterogeneous model architectures. Thus, HDHRFL can save more computational energy than FedProx and HAF-Edge. Beyond the idea of HDHRFL, FedALT takes the dynamic computational capacities of agents into consideration and dynamically selects the most suitable model architecture for each agent based on its local data distribution. Furthermore, FedALT can achieve finer-grained savings of on-agent computational energy through the cooperation between the adaptive model-selection mechanism and the dynamic local epoch-adjustment mechanism. Therefore, compared with

TABLE III
GLOBAL MODEL ACCURACY Acc_g (%) OF DIFFERENT METHODS ON THE SIMULATION PLATFORM UNDER HIGH AGENT HETEROGENEITY $f = 3$.

Methods	MNIST			CIFAR-10			Average
	$Dir(0.1)$	$Dir(1.0)$	$Dir(10)$	$Dir(0.1)$	$Dir(1.0)$	$Dir(10)$	
HAF-Edge	88.92	90.59	95.21	70.38	74.91	80.25	83.38
FedProx	89.74	91.73	95.28	71.47	75.64	**80.41**	84.05
HDHRFL	90.65	91.86	95.25	72.56	75.42	80.31	84.34
FedALT (Ours)	**91.18**	**92.04**	**95.32**	**73.98**	**76.13**	80.37	**84.84**

the suboptimal HDHRFL, FedALT can significantly reduce the E_{avg}^{comp} on both datasets. All experiments in this section demonstrate that even under more complex datasets and higher agent heterogeneity, FedALT can maintain a better performance and save more computational energy on agents.

C. Analysis of Acc_g on the Simulation Platform

On the simulation platform, Table III reports the global model accuracy Acc_g of HAF-Edge, FedProx, HDHRFL, and the proposed FedALT on the MNIST and CIFAR-10 datasets under three data heterogeneity $\alpha = \{0.1, 1.0, 10\}$ and the highest agent heterogeneity $f = 3$. We find that FedALT can achieve the highest average Acc_g on both datasets.

As reported in Table III, under high data heterogeneity $\alpha = 0.1$, FedALT achieves the highest $Acc_g = 91.18\%$ on the MNIST dataset, which is 2.26%, 1.44%, and 0.53% higher than that of HAF-Edge, FedProx, and HDHRFL, respectively. At the same time, on the CIFAR-10 dataset, FedALT can achieve the highest $Acc_g = 73.98\%$, which is 3.60%, 2.51%, and 1.42% higher than that of HAF-Edge, FedProx, and HDHRFL, respectively. Under low data heterogeneity $\alpha = 1.0$, the Acc_g of all four methods increases. FedALT achieves the highest $Acc_g = 92.04\%$ on the MNIST dataset, and HDHRFL achieves the suboptimal $Acc_g = 91.86\%$, which is 0.18% lower than that of FedALT. Meanwhile, on the CIFAR-10 dataset, FedALT also achieves the highest $Acc_g = 76.13\%$, and FedProx achieves the suboptimal $Acc_g = 75.42\%$, which is 0.71% lower than that of FedALT. As the data heterogeneity increases, the gap among the Acc_g of these four methods is getting smaller. Under the lowest data heterogeneity $\alpha = 10$, the Acc_g of all four methods are close. FedALT still achieves the highest $Acc_g = 95.32\%$ on the MNIST dataset, which outperforms HAF-Edge, FedProx, and HDHRFL by 0.11%, 0.04%, and 0.07%, respectively. However, on the CIFAR-10 dataset, FedProx achieves the highest $Acc_g = 80.41\%$, which outperforms HAF-Edge, HDHRFL, and FedALT by 0.16%, 0.10%, and 0.04%, respectively. It can be seen that the improvement in the Acc_g of FedALT on the CIFAR-10 dataset is significantly greater than the MNIST dataset, which indicates that FedALT can achieve higher Acc_g when training on more complex datasets. After averaging the Acc_g across both datasets, we find that FedALT achieves the highest average $Acc_g = 84.84\%$, which outperforms HAF-Edge, FedProx, and HDHRFL by 1.46%, 0.79%, and 0.50%, respectively.

The regularization term in FedProx can excessively constrain local updates under high heterogeneity and reduce the global model accuracy. The aggregation bias in HAF-Edge can degrade the global model accuracy. To improve model accuracy, HDHRFL transfers knowledge among heterogeneous models, so that the Acc_g of HDHRFL is higher than that of FedProx and HAF-Edge. The adaptive model-selection mechanism in FedALT can ensure that each local model can fully learn its local data characteristics, while the dynamic epoch-adjustment mechanism can maintain the effectiveness of local training. To further improve global model accuracy, FedALT conducts the dual knowledge-transfer mechanism to reduce update drifts among local models. Thanks to the dual knowledge-transfer mechanism on the edge server, FedALT can better transfer the knowledge among heterogeneous models and improve the model accuracy by sharing partial model layers and knowledge distillation, which is the key mechanism for improving the model accuracy. All experiments in this section demonstrate that even under more complex datasets and higher data heterogeneity, FedALT can maintain a better performance and achieve a higher global model accuracy.

VI. HARDWARE PLATFORM EVALUATION

A. Experimental Settings

Implementation Environment. Our physical experiments run on the hardware platform equipped with 2 NVIDIA *A100* as the cloud server, 2 NVIDIA *RTX3090* as the edge servers, and 5 *Jetson Orin Nano* boards together with 5 *Jetson Orin NX* boards as the mobile agents. Note that hardware nodes communicate over a local area network, and the jtop tool is used to monitor the computational energy on the boards. The configurations of the hardware platform are as follows.

- The *A100* server is equipped with an Intel(R) Xeon(R) Gold 6326 CPU @2.90GHz, 256GB of memory, and 2×80GB GPU memory.
- The *RTX3090* edge server is equipped with an Intel(R) Xeon(R) Silver 4116 CPU @2.10GHz, 92GB of memory, and 2×24GB GPU memory.
- The *Jetson Orin Nano* board is equipped with a 6-core Arm Cortex-A78AE CPU, an NVIDIA Ampere architecture GPU with 128 CUDA cores, and 8GB of GPU memory.
- The *Jetson Orin NX* board is equipped with a 6-core NVIDIA Carmel ARM CPU, an NVIDIA Volta architecture GPU with 384 CUDA cores, and 8GB of GPU memory.

Platform Settings. On the hardware platform, we set the total number of mobile agents $N = 10$ and the total number of edge servers $S = 2$. Other settings, including heterogeneous computational capacity on agents, Non-IID datasets, generator model, target models, hyperparameter settings, and baselines, are the same as those of the simulation platform.

Metrics. Global model accuracy Acc_g and the following total computational energy E_{total}^{comp} are used to evaluate FedALT.

- *Total Computational Energy* E_{total}^{comp}. According to (9), we sum the $E_{t_d}^{comp}$ to get the total computational energy E_{total}^{comp} on the hardware platform, which is given by

$$E_{total}^{comp} = \sum\nolimits_{t_d=1}^{T_d} E_{t_d}^{comp}. \tag{51}$$

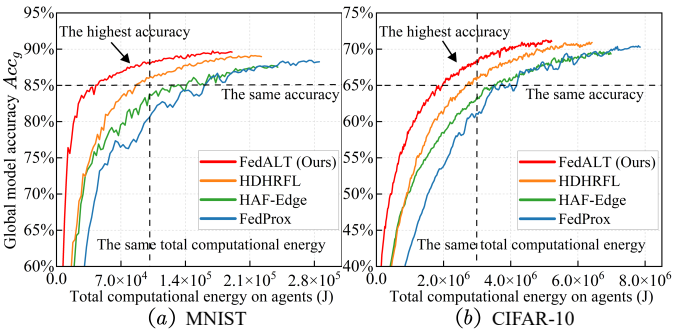


Fig. 5. Total computational energy E_{total}^{comp} consumed by agents on the hardware platform under high data heterogeneity $\alpha = 0.1$.

B. Analysis of E_{total}^{comp} and Acc_g on the Hardware Platform

On the hardware platform, we calculate the total computational energy E_{total}^{comp} when the global model reaches the target Acc_g. Fig. 5 shows the energy-accuracy curve of HAF-Edge, FedProx, HDHRFL, and the proposed FedALT on the MNIST and CIFAR-10 datasets under the highest data heterogeneity $\alpha = 0.1$. We find that when the E_{total}^{comp} is equal, FedALT can achieve the highest Acc_g, and when the target Acc_g is equal, FedALT can achieve the lowest E_{total}^{comp}.

As shown in Fig. 5, under the highest data heterogeneity $\alpha = 0.1$, FedALT can converge faster than the other three baselines and achieve the highest global accuracy $Acc_g = 89.60\%$ and $Acc_g = 71.09\%$ on the MNIST and CIFAR-10 datasets, respectively. When FedALT reaches approximately $E_{total}^{comp} = 1.39 \times 10^5$ J, the global model begins to converge on the MNIST dataset, while HDHRFL achieves the suboptimal $Acc_g = 88.97\%$ and HAF-Edge achieves the lowest $Acc_g = 87.51\%$. Similar results also appear on the CIFAR-10 dataset, HDHRFL also achieves the suboptimal $Acc_g = 70.91\%$ and HAF-Edge achieves the lowest $Acc_g = 69.33\%$. It can be seen that the Acc_g achieved on the hardware platform is slightly lower than those of the simulation platform. When we fix a line on the curve for $E_{total}^{comp} = 1.0 \times 10^5$ J on the MNIST dataset, we can see that FedALT achieves the highest $Acc_g = 87.98\%$ under the same E_{total}^{comp} budget, which is 2.01%, 4.19%, and 7.42% higher than that of HDHRFL, HAF-Edge, and FedProx, respectively. At the same time, when we fix a line on the curve for $E_{total}^{comp} = 3.0 \times 10^6$ J on the CIFAR-10 dataset, we can see that FedALT achieves the highest $Acc_g = 68.31\%$ under the same E_{total}^{comp} budget, which is 2.12%, 4.92%, and 7.31% higher than that of HDHRFL, HAF-Edge, and FedProx, respectively. Furthermore, when we fix a line on the curve for $Acc_g = 85.00\%$ on the MNIST dataset, FedALT achieves the lowest $E_{total}^{comp} = 4.29 \times 10^4$ J, which is approximately 50.55%, 69.74%, and 73.73% lower than that of HDHRFL, HAF-Edge, and FedProx, respectively. Meanwhile, when we fix a line on the curve for $Acc_g = 65.00\%$ on the CIFAR-10 dataset, FedALT achieves the lowest $E_{total}^{comp} = 1.97 \times 10^6$ J, which is approximately 25.98%, 44.06%, and 53.09% lower than that of HDHRFL, HAF-Edge, and FedProx, respectively.

Although HDHRFL achieves the suboptimal Acc_g and converges faster than HAF-Edge and FedProx, FedALT can reduce more E_{total}^{comp} than HDHRFL. For example, when the global model converges, the E_{total}^{comp} of FedALT on the MNIST dataset is about 22.80% lower than HDHRFL, while the final Acc_g of FedALT is approximately 0.63% higher than HDHRFL. We can find similar results on the CIFAR-10 dataset, FedALT can also reduce more E_{total}^{comp} and achieve higher Acc_g than taht of HDHRFL. This is because HDHRFL does not consider whether each heterogeneous model matches the local dataset on each agent. Superior to HDHRFL, the adaptive model-selection mechanism in FedALT is built to select the most suitable model for each agent, which can fully leverage the computational capacity of agents while reducing computational energy and improving the global model accuracy. All experiments in this section demonstrate that, even on the hardware platform, FedALT can achieve higher global model accuracy while saving more computational energy.

VII. Conclusion

Edge mobile agents with limited computational capacity and battery life may drop out of FL training prematurely due to low battery, which can degrade the convergence speed and accuracy of the global model. To this end, we propose a hierarchical Federated learning framework based on Adaptive Local Training (FedALT). Specifically, FedALT builds the adaptive model-selection mechanism on the cloud server to reduce on-agent computational energy. To save on-agent computational energy in a fine-grained manner, FedALT designs the dynamic epoch-adjustment mechanism on each agent. Furthermore, FedALT conducts the dual knowledge-transfer mechanism on each edge server to reduce update drifts among local models. Experiments on both the simulation platform and the hardware platform demonstrate that, compared to baselines, FedALT can achieve the lowest computational energy, the highest average global model accuracy, and the fastest global model convergence. These improvements indicate that FedALT is a promising approach that can significantly reduce computational energy for edge mobile agents in FL.

References

[1] S. Zhang *et al.*, "Large Models for Aerial Edges: An Edge-Cloud Model Evolution and Communication Paradigm," in *IEEE Journal on Selected Areas in Communications*, vol. 43, no. 1, pp. 21-35, Jan. 2025.

[2] X. Cheng, R. Meng, X. Xu, H. Gao, P. Zhang and D. Niyato, "APEG: Adaptive Physical Layer Authentication With Channel Extrapolation and Generative AI," in *IEEE Transactions on Information Forensics and Security*, vol. 21, pp. 1257-1272, 2026.

[3] D. Thakur, A. Guzzo, G. Fortino, and F. Piccialli, "Green Federated Learning: A New Era of Green Aware AI," *ACM Comput. Surv.*, vol. 57, no. 8, p. 194:1-194:36, Mar. 2025.

[4] Y. Liu *et al.*, "Hierarchical Micro-Segmentations for Zero-Trust Services via Large Language Model-Enhanced Graph Diffusion," in *IEEE Transactions on Networking*, vol. 34, pp. 3029-3044, 2026.

[5] S. Zhang, Z. Zheng, F. Wu, B. Li, Y. Shao and G. Chen, "MIDDLE: A Mobility-Driven agent-Edge-Cloud Federated Learning Framework," in *IEEE Transactions on Mobile Computing*, vol. 24, no. 6, pp. 4589-4606, June 2025.

[6] R. Zhang *et al.*, "Generative AI Agents With Large Language Model for Satellite Networks via a Mixture of Experts Transmission," in *IEEE Journal on Selected Areas in Communications*, vol. 42, no. 12, pp. 3581-3596, Dec. 2024.

[7] M. Rahmati, "Energy-Aware Federated Learning for Secure Edge Computing in 5G-Enabled IoT Networks," *Journal of Electrical Systems and Inf Technol*, vol. 12, no. 1, p. 13, May 2025.

[8] T. Pang *et al.*, "Efficient LLM Deployment in Consumer Electronics Applications via Resource-Aware and Salience-Guided Quantization," in *IEEE Transactions on Consumer Electronics*, vol. 72, no. 1, pp. 2122-2134, Feb. 2026.

[9] Y. Li, X. Wang, H. Li, P. K. Donta, M. Huang, and S. Dustdar, "Communication-Efficient Federated Learning for Heterogeneous Clients," *ACM Trans. Internet Technol.*, vol. 25, no. 2, pp. 1–37, May 2025.

[10] C. Jin, T. Du, and X. Chen, "Energy-Efficient Model Decoupling for Personalized Federated Learning on Cloud-Edge Computing Networks," *ACM Transactions on Emerging Telecommunications Technologies*, vol. 36, no. 7, p. e70203, 2025.

[11] M. Kim, W. Saad, M. Mozaffari and M. Debbah, "Green, Quantized Federated Learning Over Wireless Networks: An Energy-Efficient Design," in *IEEE Transactions on Wireless Communications*, vol. 23, no. 2, pp. 1386-1402, Feb. 2024.

[12] L. Lei et al., "Energy Optimization and Lightweight Design for Efficient Federated Learning in Wireless Edge Systems," in *IEEE Transactions on Vehicular Technology*, vol. 73, no. 9, pp. 13542-13557, Sept. 2024.

[13] Z. Lin et al., "Hierarchical Split Federated Learning: Convergence Analysis and System Optimization," in *IEEE Transactions on Mobile Computing*, vol. 24, no. 10, pp. 9352-9367, Oct. 2025.

[14] K. Yang, B. Hu and M. Zhao, "Coordinating Computational Capacity for Adaptive Federated Learning in Heterogeneous Edge Computing Systems," in *IEEE Transactions on Parallel and Distributed Systems*, vol. 36, no. 8, pp. 1509-1523, Aug. 2025.

[15] W. Gao, O. Tavallaie, S. Chen, and A. Zomaya, "Federated Learning as A Service for Hierarchical Edge Networks with Heterogeneous Models," in *International Conference on Service-Oriented Computing*. Springer, 2024, pp. 85–99.

[16] S. Wang et al., "Adaptive Federated Learning in Resource Constrained Edge Computing Systems," in *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 6, pp. 1205-1221, June 2019.

[17] T. Li, A. K. Sahu, M. Zaheer, M. Sanjabi, A. Talwalkar, and V. Smith, "Federated Optimization in Heterogeneous Networks," *Proceedings of Machine Learning and Systems*, vol. 2, pp. 429–450, Mar. 2020.

[18] L. Shen, Y. Sun, Z. Yu, L. Ding, X. Tian, and D. Tao, "On Efficient Training of Large-Scale Deep Learning Models," *ACM Comput. Surv.*, vol. 57, no. 3, p. 57:1-57:36, Nov. 2024.

[19] S. Rajput, T. Widmayer, Z. Shang, M. Kechagia, F. Sarro, and T. Sharma, "Enhancing Energy-Awareness in Deep Learning through Fine-Grained Energy Measurement," *ACM Trans. Softw. Eng. Methodol.*, vol. 33, no. 8, p. 211:1-211:34, Dec. 2024.

[20] R. Zhang, Yinqiu Liu, Shunpu Tang, Jiacheng Wang, Dusit Niyato, Geng Sun, Yonghui Li, Sumei Sun, "Covert Prompt Transmission for Secure Large Language Model Services," in *IEEE Journal on Selected Areas in Communications*, vol. 44, pp. 1589-1603, 2026.

[21] X. Pang, J. Hu, P. Sun, J. Ren and Z. Wang, "When Federated Learning Meets Knowledge Distillation," in *IEEE Wireless Communications*, vol. 31, no. 5, pp. 208-214, October 2024.

[22] L. Luo and X. Zhang, "Federated Split Learning via Mutual Knowledge Distillation," in *IEEE Transactions on Network Science and Engineering*, vol. 11, no. 3, pp. 2729-2741, May-June 2024.

[23] D. Li and J. Wang, "FedMD: Heterogeneous Federated Learning via Model Distillation," in *Proc. Int. Conf. Neural Inf. Process. Syst. Workshop*, 2019, pp. 1–8.

[24] Y. Jiang, D. Wang, B. Song, and S. Luo, "HDHRFL: A Hierarchical Robust Federated Learning Framework for Dual-Heterogeneous and Noisy Clients," *Future Generation Computer Systems*, vol. 160, pp. 185–196, Nov. 2024.

[25] Z. Zhu, J. Hong, and J. Zhou, "Data-Free Knowledge Distillation for Heterogeneous Federated Learning," in *Proceedings of the 38th International Conference on Machine Learning*, PMLR, July 2021, pp. 12878–12889.

[26] H. Yuan and T. Ma, "Federated Accelerated Stochastic Gradient Descent," in *Advances in Neural Information Processing Systems*, Curran Associates, Inc., 2020, pp. 5332–5344.

[27] R. Zhang, K. Xiong, Y. Lu, P. Fan, D. W. K. Ng and K. B. Letaief, "Energy Efficiency Maximization in RIS-Assisted SWIPT Networks With RSMA: A PPO-Based Approach," in *IEEE Journal on Selected Areas in Communications*, vol. 41, no. 5, pp. 1413-1430, May 2023.

[28] J. Kaplan, S. McCandlish, T. Henighan, T. B. Brown, B. Chess, R. Child, S. Gray, A. Radford, J. Wu, and D. Amodei, "Scaling Laws for Neural Language Models," arXiv preprint arXiv: 2001.08361, 2020.

[29] J. Feng, F. R. Yu, Q. Pei, J. Du and L. Zhu, "Joint Optimization of Radio and Computational Resources Allocation in Blockchain-Enabled Mobile Edge Computing Systems," in *IEEE Transactions on Wireless Communications*, vol. 19, no. 6, pp. 4321-4334, June 2020.

[30] L. Liu et al., "Blockchain-Enabled Secure Data Sharing Scheme in Mobile-Edge Computing: An Asynchronous Advantage Actor–Critic Learning Approach," in *IEEE Internet of Things Journal*, vol. 8, no. 4, pp. 2342-2353, 15 Feb.15, 2021.

[31] A. Tusha and H. Arslan, "Interference Burden in Wireless Communications: A Comprehensive Survey From PHY Layer Perspective," in *IEEE Communications Surveys & Tutorials*, vol. 27, no. 4, pp. 2204-2246, Aug. 2025.

[32] R. Zhang *et al*., "Toward Agentic AI: Generative Information Retrieval Inspired Intelligent Communications and Networking," in *IEEE Communications Magazine*, vol. 64, no. 1, pp. 197-204, January 2026.

[33] D. Li, W. E. Wong, W. Wang, Y. Yao and M. Chau, "Detection and Mitigation of Label-Flipping Attacks in Federated Learning Systems with KPCA and K-Means," in *Proceedings of 2021 8th International Conference on Dependable Systems and Their Applications (DSA)*, Yinchuan, China, 2021, pp. 551-559.

[34] L. Dai, F. Zeng, H. Kong, J. Cai, H. Jiang and K. Li, "Throughput-Aware Cooperative Task Offloading in Dynamic Mobile Edge Computing Systems," in *IEEE Transactions on Mobile Computing*, vol. 24, no. 12, pp. 13276-13292, Dec. 2025.

[35] Y. Liu *et al*., "LAMeTA: Intent-Aware Agentic Network Optimization via a Large AI Model-Empowered Two-Stage Approach," in *IEEE Journal on Selected Areas in Communications*, vol. 44, pp. 2462-2478, 2026.

[36] F. Sattler *et al*., "CFD: Communication-Efficient Federated Distillation via Soft-Label Quantization and Delta Coding," in *IEEE Transactions on Network Science and Engineering*, vol. 9, no. 4, pp. 2025-2038, 1 July-Aug. 2022.

[37] T. van Erven and P. Harremos, "Rényi Divergence and Kullback-Leibler Divergence," in *IEEE Transactions on Information Theory*, vol. 60, no. 7, pp. 3797-3820, July 2014.

[38] S. Zhang, Shuai Jiang *et al*., "Generative AI on SpectrumNet: An Open Benchmark of Multiband 3-D Radio Maps," in *IEEE Transactions on Cognitive Communications and Networking*, vol. 11, no. 2, pp. 886-901, April 2025, doi: 10.1109/TCCN.2024.3502492.

[39] Y. Shi, Y. Liu, K. Wei, L. Shen, X. Wang, and D. Tao, "Make Landscape Flatter in Differentially Private Federated Learning," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition,* CVPR, 2023, pp. 24552-24562.

[40] H. Wang, Y. Li, W. Xu, R. Li, Y. Zhan, and Z. Zeng, "DaFKD: Domain-Aware Federated Knowledge Distillation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, CVPR, 2023, pp. 20412–20421.

[41] Y. LeCun and C. Cortes, "MNIST Handwritten Digit Database." 2010.

[42] A. Krizhevsky, "Learning Multiple Layers of Features from Tiny Images," *MIT and NYU, Tech. Rep.*, 2009.

[43] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 770–778.

[44] S. H. Hasanpour, M. Rouhani, M. Fayyaz, M. Sabokrou, and E. Adeli, "Towards Principled Design of Deep Convolutional Networks: Introducing Simpnet," arXiv preprint arXiv:1802.06205, 2018.

[45] M. Tan and Q. Le, "EfficientNetV2: Smaller Models and Faster Training," in *Proceedings of the 38th International Conference on Machine Learning*, PMLR, July 2021, pp. 10096–10106.

[46] Z. Yang, M. Chen, W. Saad, C. S. Hong and M. Shikh-Bahaei, "Energy Efficient Federated Learning Over Wireless Communication Networks," in *IEEE Transactions on Wireless Communications*, vol. 20, no. 3, pp. 1935-1949, March 2021.

Benteng Zhang (Student Member, IEEE) received the B.S. degree in software engineering from the College of Computer Science and Technology, Qingdao University, Qingdao, China, in 2021. He is currently pursuing the Ph.D. degree with the College of Computer Science and Software Engineering, Hohai University, Nanjing.

His research interests include distributed machine learning, edge computing, and federated learning.

Jianxin Huang received the M.S. degree in electrical power electronics and drives from Dalian Maritime University, Dalian, China, in 2024. He is currently the Vice President and General Manager of the R&D Center at Suma Information Industry Co., Ltd.

His research interests include server system architecture design and advanced computing.

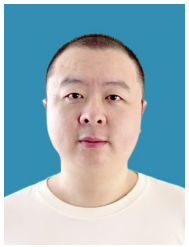
Yingchi Mao (Member, IEEE) received the Ph.D. degree in computer software and theory from the Department of Computer Science and Technology, Nanjing University, Nanjing, China in 2007. She serves with the Key Laboratory of Water Big Data Technology, Ministry of Water Resources, Nanjing, and she is also currently a Professor with the College of Computer Science and Software Engineering, Hohai University, Nanjing. Her main research interests include edge intelligent computing, Internet of Things data analysis, and mobile sensing system.

Prof. Mao is a Senior Member of China Computer Federation and Chinese Association of Automation.

Feng Mao received the Ph.D. degree in thermal engineering from the School of Resources and Environmental Engineering, East China University of Science and Technology, Shanghai, China, in 2021. He serves as an R&D Engineer at Suma Information Industry Co., Ltd. and a Postdoctoral Researcher at the College of Computer Science and Software Engineering, Hohai University, Nanjing, China.

His research interests include advanced computing, deep learning, and fault diagnosis.

Tianfu Pang (Student Member, IEEE) received the M.S. degree in software engineering from the College of Computer Science and Software Engineering, Hohai University, Nanjing, China, in 2025. He is currently pursuing the Ph.D. degree with the College of Computer Science and Software Engineering, Hohai University, Nanjing.



Xiaoming He (Member, IEEE) received the Ph.D. degree in Computer Science and Software Engineering from Hohai University, Nanjing, China, in 2023. He is currently a Lecturer with the College of Internet of Things, Nanjing University of Posts and Telecommunications, Nanjing, China. Prior to work, he was a Visiting Research Fellow in Singapore University of Technology and Design.

His current research interests include edge intelligence and FPGA-based AI accelerator.

Zhenxiang Pan received the B.S. degree in Information Management and Information System from Shanghai University of International Business and Economics, Shanghai, China, in 2021. He is currently working forward the Ph.D. degree in Computer Science and Technology at Hohai University, Nanjing, China.

Jie Wu (Fellow, IEEE) received the Ph.D. degree in computer engineering from Florida Atlantic University, Boca Raton, FL, USA, in 1989. He is the Director of the China Telecom Cloud Computing Research Institute and a Laura H. Carnell Professor with Temple University, Philadelphia, PA, USA, and also serves as the Director of International Affairs, College of Science and Technology. Dr. Wu is the recipient of the 2011 China Computer Federation (CCF) Overseas Outstanding Achievement Award. He was an IEEE Computer Society Distinguished Visitor, an ACM Distinguished Speaker, and the Chair for the IEEE Technical Committee on Distributed Processing. He is a Fellow of the AAAS.