The Wonderful Flower of Seven Colors: Self-Evolution Approach for Multi-Terminal Personalized Service Processing

Jia Zhao, *Member, IEEE*, Yafei Zhu, Keqin Li, *Fellow, IEEE*, Ming Hu, *Member, IEEE*, and Yan Ding, *Member, IEEE*

***Abstract*—In mobile cloud computing, traditional frameworks often treat mobile devices as passive data collectors, relying heavily on centralized cloud centers for model processing. This centralized approach limits data-processing accuracy and efficiency due to bandwidth constraints, communication latency, and limited computational utilization of mobile devices. Furthermore, as mobile applications increasingly require real-time, personalized services, these limitations restrict model adaptability and responsiveness, emphasizing the need for a distributed approach that can dynamically adjust to varying service demands and network conditions. This paper proposes FLOM (Framework-Level Self-evolution Optimization Model), a novel framework designed to support autonomous, adaptive model optimization across multi-terminal systems. Unlike traditional methods, FLOM leverages the computational capacity of mobile devices to enable on-site model updates and intelligent data allocation through a collaborative network of cloud and mobile nodes. FLOM incorporates deep reinforcement learning for dynamic model evolution, allowing real-time adaptation across terminals. By utilizing model information from each device, FLOM achieves balanced self-evolution, optimizing processing accuracy and system resilience. The framework introduces three major innovations: (1) Self-adaptive Model Evolution by utilizing mobile terminal resources to locally update models, FLOM reduces dependence on centralized data centers, ensuring faster, context-aware adjustments that enhance real-time service delivery; (2) Intelligent Resource Allocation supporting real-time, task-targeted distribution to enhance fault tolerance and system efficiency, FLOM effectively allocates resources to handle diverse service needs across terminals; and (3) Collaborative Optimization by integrating model characteristics from multiple terminals, FLOM enables unified, system-wide optimization that improves model robustness and adaptability. Experimental results show that FLOM significantly improves data accuracy, reduces processing time, and enhances fault tolerance compared to traditional frameworks, making it a robust and versatile solution for real-time, adaptive mobile cloud applications.**

***Index Terms*—Mobile cloud computing, big data analysis, reinforcement learning, self-evolution approach, service processing.**

I. Introduction

In RECENT years, mobile cloud computing has become a focal point for researchers aiming to expand the capabilities of cloud computing by incorporating mobile devices as active computing resources. While traditional cloud computing primarily leverages centralized cloud servers for data processing and storage, this model often relegates mobile devices to a passive role, limiting them to data collection and remote communication with cloud centers. Current mobile cloud architectures, therefore, typically rely on mobile devices to collect and transmit raw data to centralized cloud data centers, where computationally intensive models are applied to process and analyze the data. However, such a structure introduces multiple constraints. The heavy reliance on centralized processing significantly hampers real-time data analysis and adaptive service delivery, primarily due to limitations such as bandwidth constraints, communication latency, and an under-utilization of the computational resources available on mobile devices [1].

The rapid proliferation of mobile applications requiring real-time, personalized services has further underscored these limitations, emphasizing the need for frameworks that can respond instantaneously to changing conditions and user-specific needs. In fields ranging from healthcare to autonomous driving, applications demand highly adaptive processing capabilities that traditional centralized cloud architectures are ill-equipped to deliver. Network bandwidth limitations and communication delays inherent in these architectures hinder their ability to provide the low-latency, responsive processing essential for real-time applications. Furthermore, while the processing power and storage capabilities of mobile devices have advanced significantly, these resources remain largely untapped in traditional frameworks, resulting in a substantial inefficiency that limits the potential of mobile cloud computing to meet modern application requirements [2].

This paper introduces FLOM (Framework-Level Self-evolution Optimization Model), a novel distributed framework

Received 24 March 2023; revised 30 June 2025; accepted 5 July 2025. Date of publication 24 July 2025; date of current version 10 April 2026. This work was supported by the Outstanding Young Science and Technology Talents Project of Jilin Province Science and Technology Development Plan under Grant 20240602004RC. *(Corresponding author: Yan Ding.)*

Jia Zhao is with the College of Artificial Intelligence Technology & School of Computer Technology and Engineering, Changchun Institute of Technology, Changchun 130012, China, and also with the School of Electronics Engineering and Computer Science, Peking University, Beijing 100871, China.

Yafei Zhu is with the School of Computer Science and Engineering, Changchun University of Technology, Changchun 130012, China.

Keqin Li is with the State University of New York, Albany, NY 12246 USA, and also with Hunan University, Hunan 410012, China.

Ming Hu and Yan Ding are with the College of Artificial Intelligence Technology & School of Computer Technology and Engineering, Changchun Institute of Technology, Changchun 130012, China (e-mail: dingyan@ccit.edu.cn).

Digital Object Identifier 10.1109/TSC.2025.3592421

designed to overcome these limitations by enabling autonomous, adaptive model evolution across multi-terminal systems within a mobile cloud environment. Unlike conventional methods that offload all computational tasks to the cloud, FLOM utilizes deep reinforcement learning to empower mobile devices to conduct on-site model updates, intelligent data allocation, and adaptive processing. By coordinating the computational resources of cloud centers with those of mobile devices, FLOM facilitates real-time, context-aware service delivery that dynamically adapts to diverse user needs and environmental conditions. This distributed approach not only addresses the drawbacks of bandwidth dependency and latency but also maximizes the computing potential of mobile devices, making it suitable for applications requiring high adaptability and responsiveness.

Existing mobile cloud architectures commonly operate in an offline mode, where data is collected on mobile devices but processed centrally on cloud servers. While this approach expands mobile computing capabilities, it disconnects data collection from real-time processing, preventing the framework from adapting models based on live data. In addition, many mobile cloud systems lack a targeted task allocation mechanism, leading to random task assignment and limited ability to handle different service types with varying requirements. As a result, such frameworks face difficulties in improving model accuracy and efficiency in processing dynamic data streams [3], [4], [5]. FLOM, however, addresses these limitations by forming a collaborative network of cloud and mobile nodes, wherein each device performs context-specific updates and resource optimization, resulting in a framework that enhances the precision, adaptability, and scalability of mobile cloud computing environments.

The contributions of this paper are threefold. First, Self-adaptive Model Evolution empowers mobile devices to update models locally, reducing dependence on centralized cloud data centers. This localized update mechanism enables rapid, context-sensitive model adjustments, which are critical for delivering real-time services. Second, Intelligent Resource Allocation ensures that tasks are distributed efficiently across multiple devices based on real-time demand and device availability, thus enhancing system-wide fault tolerance and data-processing efficiency. Finally, Collaborative Optimization facilitates the integration of model parameters across devices, allowing each mobile terminal to contribute to a holistic optimization of the framework. This approach not only improves model robustness but also ensures that FLOM can maintain a high level of accuracy and resilience under varying network conditions.

The primary contributions of this study are detailed as follows:

1) Exploiting mobile device computational power: FLOM harnesses the processing resources of mobile devices for model training and updates, moving beyond the constraints of centralized cloud reliance. This enables real-time adaptation and more efficient use of the full computational spectrum available in mobile cloud environments.
2) Self-evolution of dynamic models: By employing deep reinforcement learning, FLOM autonomously updates and evolves lightweight data models on mobile devices. This dynamic adaptation of models ensures that they remain responsive to new data and evolving service requirements, achieving a seamless model self-evolution process.
3) A distributed framework for multi-terminal processing and optimization: FLOM enables efficient task classification, resource allocation, and data processing across cloud and mobile nodes. This distributed approach addresses the interaction and optimization challenges inherent in mobile cloud computing environments, facilitating a cohesive and scalable system architecture.

The rest of this paper is organized as follows. Section II provides a comprehensive overview of related work and highlights the gaps that FLOM aims to address. Section III presents the research questions and formalizes the specific challenges tackled in this study. Section IV elaborates on the FLOM system architecture and the methodologies developed to address the identified challenges. Section V discusses the experimental setup and presents the results that validate the effectiveness of the proposed framework. Finally, Section VI concludes the paper and outlines potential directions for future research.

II. Related Work

The rapid proliferation of mobile cloud computing (MCC) has catalyzed significant advancements across diverse domains, such as autonomous driving, healthcare, and smart IoT systems. MCC integrates distributed computing paradigms, including mobile edge computing (MEC), which emphasizes processing at the network's edge, closer to data sources, and mobile intelligent computing (MIC), which leverages AI-driven inference within mobile devices. Together, these approaches aim to achieve low-latency, adaptive, and resource-efficient computation by balancing workloads between mobile devices and cloud resources. Despite these advancements, several challenges persist, including the heterogeneity of devices, dynamic network conditions, energy efficiency constraints, and stringent privacy requirements. Researchers have developed a spectrum of strategies, ranging from deep neural network (DNN) partitioning and adaptive inference to federated learning (FL) and joint optimization frameworks, to address these challenges. While these strategies have yielded valuable insights, the need for a unified and scalable framework remains. This section systematically categorizes and evaluates key contributions, limitations, and emerging trends to identify pathways for developing robust and adaptive MEC systems.

Static DNN Partitioning and Resource Allocation: Static DNN partitioning has long been foundational in addressing computational constraints by distributing tasks between resource-constrained mobile devices and edge servers. Ye et al. [6] explored cooperative edge inference through an end-to-end delay minimization approach, combining DNN partitioning with resource allocation to achieve low-latency, multi-device optimization. Wang et al. [7] effectively enhances task allocation and execution efficiency in mobile edge computing, while significantly reducing delay and energy consumption with data privacy preserved. Chen et al. [8] proposed a Personalized Federated Deep Reinforcement Learning-based computation offloading and resource allocation method (PFR-OA), which adapts to the

dynamic MEC environment and personalized demands in smart communities, significantly improving task execution success rate while reducing delay and energy consumption. These static approaches provide a solid foundation for distributed computation but struggle in dynamic, heterogeneous environments where device capabilities and network conditions vary unpredictably. This limitation highlights the need for dynamic inference systems that adapt to real-time changes, ensuring consistent performance in complex, rapidly evolving MEC scenarios.

Adaptive and Dynamic Inference: Dynamic inference frameworks have emerged as a vital evolution of static methodologies, offering real-time adaptability to address diverse environmental changes. Hu et al. [9] introduced a digital-twin-assisted framework for dynamic collaboration between devices and edge servers, optimizing DNN inference workflows in AIoT systems. Yang et al. [10] focused on accelerating inference for directed acyclic graph (DAG)-based DNNs through edge-cloud collaboration, enhancing the deployment of computationally intensive models in distributed environments. Wang et al. [11] proposed AdaEvo, a system enabling continuous DNN evolution by dynamically adapting models to changing data distributions. Liu et al. [12] presented MoEI, a mobility-aware inference framework integrating model partitioning and service migration, ensuring low-latency performance during user mobility. Li et al. [13] introduced an adaptive and resilient model-distributed inference system that dynamically adjusts to varying network conditions and device capabilities. Agarwal et al. [14] optimized user scheduling, task offloading ratio, and flight parameters in UAV-assisted mobile edge computing networks under large task scenarios. Peng et al. [15] optimized latency, energy consumption, and task overtime issues in smart factory tasks within the Industrial Internet of Things while ensuring data privacy and communication security. Wang et al. [16] improved training efficiency by incorporating prior knowledge to enable reward-free resource allocation, significantly enhancing the generalization and interpretability of intelligent models in vehicular networks. Zhou et al. [17] effectively improved scheduling performance in energy consumption, load balancing, and rejection rate by optimizing server deployment at user cluster centers and enhancing the network architecture of the soft actor-critic algorithm. Yang et al. [18] explored selfish acceleration techniques for on-demand DNN partitioning, enhancing resource utilization efficiency. Wu et al. [19] addressed intermittently operating devices by proposing joint optimization techniques that dynamically adjust model partitioning and resource allocation. These studies highlight the potential of dynamic inference to meet the demands of modern MEC systems. However, they often focus on isolated device optimization, underscoring the need for unified frameworks that coordinate adaptations across distributed nodes for enhanced scalability and system-wide optimization.

Federated Learning and Distributed Intelligence: Federated learning (FL) has gained prominence for enabling privacy-preserving distributed intelligence by facilitating collaborative model training across multiple devices. Gao et al. [20] introduced latency-aware FL strategies for industrial IoT applications, optimizing resource allocation to improve training efficiency. Zhao et al. [21] applied FL to satellite-assisted MEC through a federated deep recurrent Q-learning model, addressing task partitioning and resource allocation in resource-constrained remote environments. Li et al. [22] utilized Stackelberg game theory in UAV-enabled MEC, balancing user incentives with resource efficiency to enhance device participation. Hasan et al. [23] tackled challenges in vehicular edge computing by leveraging FL for robust task offloading and resource management in high-mobility scenarios. Huang et al. [24] focused on optimizing FL convergence through efficient topology management, achieving faster model synchronization across heterogeneous devices. Mendez et al. [25] reviewed FL architectures, identifying computational bottlenecks and offering insights into distributed optimization strategies. Zhao et al. [26] explored strategies to mitigate communication overhead in vehicular FL, enhancing its applicability to real-time scenarios. While FL has made significant strides, challenges such as high synchronization latency and managing device heterogeneity persist, emphasizing the need for robust cross-device coordination to support large-scale, dynamic MEC environments.

Joint Optimization Frameworks and Multi-Agent Systems: Joint optimization frameworks and multi-agent systems have emerged as key solutions to the complex interplay of resource allocation, computation offloading, and energy efficiency in MEC. Deng et al. [27] combined UAV trajectory optimization with resource allocation to enhance service quality in MEC-enabled AI applications. Xiao et al. [28] developed a multi-agent reinforcement learning (MARL) framework, optimizing energy efficiency and resource utilization in heterogeneous MEC environments. Pang et al. [29] extended MARL methodologies to ultra-dense MEC networks, achieving reliable task offloading under fluctuating conditions. Zhang et al. [30] proposed a multi-edge collaborative active video acquisition framework that uses reinforcement learning to select high-quality and non-redundant video data for continual learning, effectively reducing training data volume while maintaining model accuracy. Zou et al. [31] reduced cold starts by intelligently reusing containers and leveraging the advantages of cloud and edge server resources to effectively enhance multi-job processing capability, significantly shortening job completion time. These frameworks underscore the importance of collaborative optimization but expose challenges in achieving seamless coordination and scalability in large-scale MEC deployments, particularly under dynamic network and workload conditions.

Energy-Efficient Design for MEC: Energy efficiency is pivotal to the long-term sustainability of MEC systems. Tan et al. [32] proposed an energy-efficient joint task offloading scheme for OFDMA-based MEC, optimizing both computation and communication resources to minimize energy consumption. Wang et al. [33] achieved adaptive scheduling of heterogeneous IoT applications by capturing long-term dependencies and using prioritized experience replay, significantly reducing response time, energy consumption, and costs. Gill et al. [34] highlighted critical design principles for developing energy-efficient edge AI systems, focusing on minimizing power consumption in distributed environments. Despite these contributions, achieving universal energy efficiency across diverse MEC scenarios remains challenging. The heterogeneity of MEC systems and the

need for seamless adaptability further complicate this landscape. Addressing these challenges demands innovative frameworks that dynamically optimize energy consumption while maintaining system-wide coherence in balancing performance trade-offs.

Toward Unified, Adaptive, and Sustainable MEC Systems: Collectively, the reviewed works underscore substantial progress in DNN partitioning, resource allocation, federated learning, and adaptive inference. However, persistent challenges remain in achieving real-time adaptability, seamless multi-device coordination, and energy efficiency. Future MEC systems must transcend isolated solutions, embracing self-optimizing frameworks that integrate adaptability, distributed intelligence, and energy-efficient design. By advancing beyond incremental improvements, MEC can evolve into an autonomous and resilient ecosystem, dynamically responding to environmental changes and fostering seamless collaboration between devices and cloud resources. This transformative shift promises to redefine computational connectivity, meeting the demands of next-generation applications with unparalleled scalability, efficiency, and robustness.

III. Proposed Problem and Its Formulation

A. Proposed Problem

In mobile cloud computing, mobile devices traditionally act as passive participants, simply applying a fixed model trained in the cloud. However, due to limitations in data diversity, volume, and changing contexts, the basic machine learning models deployed on mobile devices often suffer from limited generalization and poor adaptability. This issue is particularly prominent as mobile applications increasingly demand real-time, context-aware responses and personalized service processing. The initial model, once distributed from the cloud data center, remains static and unable to adapt or improve with additional data during actual deployment, thereby limiting its accuracy and effectiveness. To illustrate, the results of model-driven service processing on mobile devices primarily depend on the model's initial training in the cloud. This dependency implies that model performance does not improve with additional local data collected by the device during use. Moreover, factors such as single-source data features and small training datasets can negatively impact model accuracy, making it challenging to generalize across varied real-world contexts encountered on mobile devices. Additionally, traditional mobile cloud frameworks struggle to support model adaptability in real time due to limited computational capabilities of individual devices and latency in communicating with cloud servers. Given these challenges, it is necessary to design an approach that enables model self-evolution within the mobile cloud computing environment. Such an approach should address the following critical issues:

1) *Localized Model Evolution:* When a mobile device operates independently from the cloud, it should be able to leverage its own collected data to locally update and refine the algorithmic model. This update process must be efficient enough to operate within the constraints of the device's computational resources while improving model accuracy based on real-time data.
2) *Cyclic Optimization Between Cloud and Devices:* When connected to the cloud, each mobile device can share updated data with it. The cloud can aggregate this information across multiple devices, facilitating a cyclical reoptimization process that enhances the base model's robustness and adaptability. This process ensures that the cloud-distributed model remains aligned with the latest real-world scenarios encountered by mobile devices, thus maintaining model relevance and effectiveness.
3) *Dynamic Task Allocation and Resource Optimization:* Beyond model evolution, mobile devices must effectively manage and allocate resources to handle diverse service requirements, particularly in dynamic and heterogeneous environments. This includes the ability to intelligently allocate computational resources and prioritize tasks based on real-time demand and device availability, optimizing both accuracy and efficiency.

By addressing these problems, the proposed FLOM framework seeks to establish a collaborative and self-optimizing ecosystem that integrates cloud and mobile devices. Leveraging deep reinforcement learning, FLOM empowers mobile devices to autonomously update and refine models, allocate resources adaptively, and facilitate coordinated optimization across devices. This distributed approach enables FLOM to overcome traditional limitations in mobile cloud computing, making it particularly suited for real-time, adaptive applications requiring continuous model evolution and efficient resource management.

B. Problem Formulation

In this study, we consider a mobile cloud computing environment consisting of multiple mobile devices and a centralized cloud data center. Let N represent the total number of mobile devices in the system, and let $n = N + 1$ denote the total number of entities, including both the mobile devices and the cloud data center. This setup enables collaborative processing between the cloud and mobile devices, aimed at optimizing computational efficiency and adaptability for real-time applications. Each mobile device $Device_i$ is represented as a tuple, $Device_i = \{Capacity_i, ComputePower_i, SupportedTasks_i, Resources_i\}$, where $Capacity_i$ defines the maximum workload that mobile device i can handle based on its computational power and available resources. $ComputePower_i$ specifies the processing capacity of mobile device i, determined by factors such as CPU speed, memory, and energy reserves. $SupportedTasks_i$ indicates the specific types of services or applications that device i is optimized to process. $Resources_i$ represents the available resources on device i, including network bandwidth, battery level, and current load.

The cloud data center, denoted C, acts as the central processing unit and model distributor in this environment. It provides the core machine learning model M and the training data $P = \{P_1^\alpha, P_2^\alpha, \ldots, P_N^\alpha\}$, where each P_i^α represents the original data used to train base model components tailored for specific service types and tasks. Once trained, the cloud distributes the model $M = \{m_1, m_2, \ldots, m_k\}$ to each mobile device, where

each m_k corresponds to a component optimized for different tasks on the mobile terminals, with k being determined by the number of model segments needed across devices.

1) Task Categorization and Service Allocation: When a pending service PendingServiceCategory arrives at the cloud data center, it is classified based on its likelihood of belonging to one of several service categories. Let PendingServiceCategory $= \{\text{CategoryProbability}_1, \ldots, \text{CategoryProbability}_w\}$, where w represents the number of possible service categories and $\text{CategoryProbability}_\sigma$ denotes the probability of the most likely category. The cloud data center uses this probability to determine the best-suited device for handling the service and then assigns it accordingly.

Given a service to be processed, the system identifies the highest probability category through cloud analysis and then transfers the service to the mobile device optimized for that category. Since each device has a segment of the core model M, it can immediately process the pending service locally, achieving local model adaptation LocalUpdate_{m_i} for the device i responsible for the task.

2) Self-Evolving Model Process: As mobile devices process services, they generate temporary data $P_{i,l}^{\beta}$, where i indexes the current mobile device and β_l represents the volume of real-time data generated. The set $P_i^{\beta} = \{P_{i,1}^{\beta}, P_{i,2}^{\beta}, \ldots, P_{i,l}^{\beta}\}$ includes all data generated by device i during model application. This data allows each device to iteratively refine the model component it uses. These updates are then synchronized back to the cloud when the device reconnects, enabling a cyclical re-optimization of the core model M across the entire system.

The primary goal of this formulation is to enable dynamic task allocation, local model adaptation, and resource sharing between the cloud and mobile devices, allowing the system to adaptively optimize processing tasks based on each device' s current state and the overall system conditions. We formulate this as a self-evolution optimization problem, aiming to achieve the following objectives:

1) *Workload Capacity Compliance:* Each device Device_i must handle service requests without exceeding its designated capacity Capacity_i. This constraint can be formulated as:

$$\sum_{j=1}^{m} \text{Load}_j \leq \text{Capacity}_i. \tag{1}$$

where m represents the number of service requests allocated to device i, and Load_j denotes the demand of each request within the device' s processing limit.

2) *Processing Efficiency:* To maintain low latency, tasks assigned to each mobile device Device_i or the cloud should be processed efficiently within ComputePower_i or ComputePower_c. The aim is to minimize the ProcessingTime for each service, a metric influenced by both the computational power of the device and the task' s complexity.

3) *Real-Time Resource Adaptability:* The system should dynamically adjust task allocations based on each device' s real-time Resources_i, adapting to variations in network bandwidth, battery levels, and current load. This adaptability ensures tasks are directed to the most capable and resource-available device, optimizing both performance and energy efficiency.

4) *Quality of Service (QoS) Assurance:* Each service has specific QoS parameters-such as latency, reliability, and accuracy-that must be met. Task assignments should consider and aim to maximize the QoS metric $\text{QoS}_{s,i}$ for each service processed by a device or cloud, ensuring the system remains responsive to service demands.

The objective function for this optimization model, therefore, can be defined as follows:

$$\max \sum_{i=1}^{n} \text{QoS}_{s,i} - \alpha \cdot \text{ProcessingTime} - \beta \cdot \sum_{i=1}^{n} \text{Resources}_i. \tag{2}$$

where:

• $\text{QoS}_{s,i} \in [0, 1]$ represents the normalized quality of service achieved for service s on device i calculated as: $QoS_{s,i} = waccuracy * Accuracy_{s,i} + wlatency * (1 - Latency_{s,i}/Latencymax) + wreliability * Reliability_{s,i}$ with weights $waccuracy = 0.5$, $wlatency = 0.3$, $wreliability = 0.2$.

• ProcessingTime $\in [0, T_{\max}]$ is the total processing time across all devices, measured in milliseconds, where $T_{\max} = 5000$ ms represents the maximum acceptable latency.

• $Resources_i \in [0, 1]$ denotes the normalized resource utilization ratio for device *i*.

• $\alpha \in [0.1, 1.0]$ is the processing time weight coefficient, empirically set to $\alpha = 0.4$ to balance QoS and latency.

• $\beta \in [0.05, 0.5]$ is the resource consumption weight coefficient, set to $\beta = 0.2$ to prioritize QoS while maintaining resource efficiency.

The coefficients α and β were determined through grid search optimization over the validation dataset, with the selected values providing the best trade-off between service quality and system efficiency.

By integrating these components, the proposed FLOM framework achieves a self-evolving allocation model that dynamically adapts to changes in device capacity, network conditions, and task requirements. This adaptive design allows the system to deliver efficient, low-latency processing, effectively overcoming conventional limitations in mobile cloud computing environments and optimizing it for real-time, adaptive applications.

As illustrated in Fig. 1, the proposed FLOM approach leverages the distinctive advantages of a mobile cloud computing environment to achieve real-time, self-adaptive processing. FLOM performs probability-based classification for incoming tasks, directing each to the most suitable mobile device based on the maximum likelihood of successful processing within its category. This targeted distribution aligns tasks with optimized device capabilities, improving efficiency and accuracy across the system.

The architecture employs a two-tiered model: a core model managed centrally in the cloud and adaptable sub-models, or "leaf" models, on individual mobile devices. This leaf structure

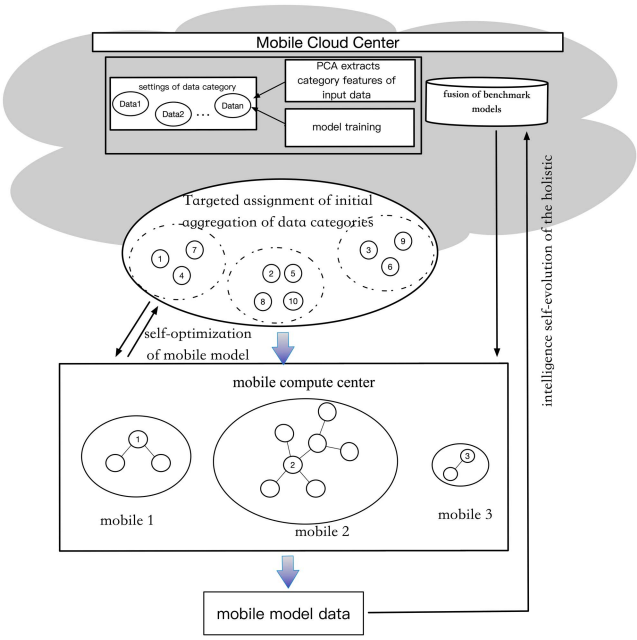


Fig. 1. Framework-Level Self-evolution Optimization Model (FLOM) Architecture (Pending services are classified by probability; tasks are then directed to optimized mobile devices with local model updates, and periodic cloud synchronization ensures continuous system-wide adaptation.).

enables mobile devices to conduct localized model updates based on incoming real-time data, enhancing the system's responsiveness to specific service requirements. FLOM's continuous self-evolution process is supported by periodic synchronization of local updates with the cloud, which refines the core model to maintain coherence across all devices.

To manage dynamically changing conditions, FLOM integrates deep reinforcement learning for resource adaptability. This approach allows FLOM to balance local processing and centralized model updates efficiently, even with limited data and varying network conditions. Through this combination of localized updates and centralized refinement, FLOM achieves high accuracy and personalized processing capabilities, sustaining performance under diverse, fluctuating conditions. The following sections provide an in-depth discussion of FLOM's core methodologies and components.

IV. FLOM: An Intelligent Self-Evolution Approach in Mobile Cloud Computing

A. Architecture Design for FLOM

Fig. 2 illustrates the proposed FLOM architecture within a mobile cloud computing environment, highlighting the interactions between the cloud data center and mobile devices. The architecture is organized into three stages: orientation recognition, mobile device self-evolution, and group self-evolution. And the pseudocode of FLOM's algorithm has been given as follows.

Initially, after receiving input data, the cloud data center extracts relevant information to classify the data and distributes it to the appropriate mobile devices for further processing. In the cloud data center, a foundational deep reinforcement learning (DRL) model is trained on basic data to establish a general-purpose model. This baseline model is then distributed to each mobile device, enabling efficient service processing without requiring task-specific model retraining for each data type encountered.

As each mobile device processes tasks, it generates temporary data. The device uses this data to update its model locally, recording the processing information through a deep neural network to achieve self-adaptation. Periodically, each device uploads its updated model to the cloud data center. The cloud data center consolidates and analyzes the models, using differential information obtained from these updates and its own stored data to retrain the baseline model. This cyclical process enables continuous refinement of the benchmark model, ultimately achieving group-level self-evolution.

B. Modelling the FLOM Framework

In this section, we provide a detailed description of the modeling process for the proposed FLOM approach. The modeling of the FLOM framework includes defining its key components, specifying the dynamic interactions between the cloud data center and mobile devices, and establishing the optimization objectives that guide the framework's self-evolution process. The overall workflow of the FLOM framework model is structured to enable adaptive, real-time processing and efficient resource allocation across diverse mobile cloud environments.

1) Data Preprocessing Module Based on PCA: To ensure the input data is effectively prepared for processing within the FLOM framework, a data preprocessing stage is applied. This stage is important in the initial phase of the mobile cloud computing center, where the core model requires an adaptable and generalizable baseline without specific preprocessing. By applying Principal Component Analysis (PCA), we aim to reduce the dimensionality of the data while retaining as much of the original information as possible. This approach provides a versatile benchmark model with high generalizability, ensuring it can accommodate a wide range of tasks across various devices in the mobile cloud environment.

The data processing module consists of two main stages:

(1) *Baseline Model Initialization:* In the uninitialized phase of the mobile cloud computing center, the core model is trained on raw input data without any specific preprocessing, thus enhancing its general applicability. This approach prioritizes model adaptability, ensuring the baseline model can handle diverse data types across devices.

(2) *Data Dimensionality Reduction Using PCA:* Once the mobile cloud computing system is operational, the incoming data is organized into a sample dataset D, comprising input vectors Z and their corresponding features N. PCA is applied to transform this dataset from a high-dimensional space to a lower-dimensional one, encapsulated in a data matrix ZX. This dimensionality reduction captures the essential structure of the data, enhancing the efficiency of subsequent classification tasks within the FLOM framework.

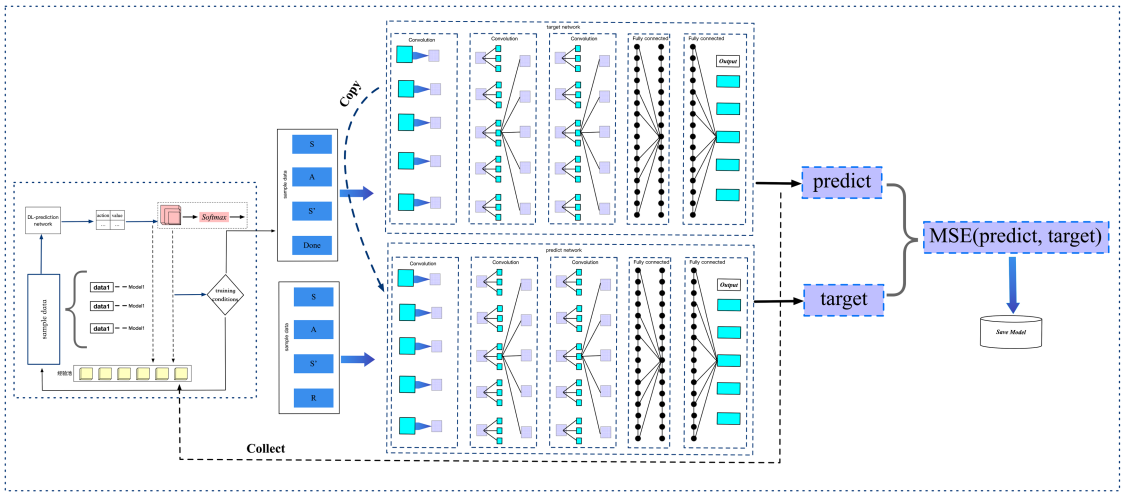


Fig. 2. Overview of the FLOM architecture in a mobile cloud computing environment.

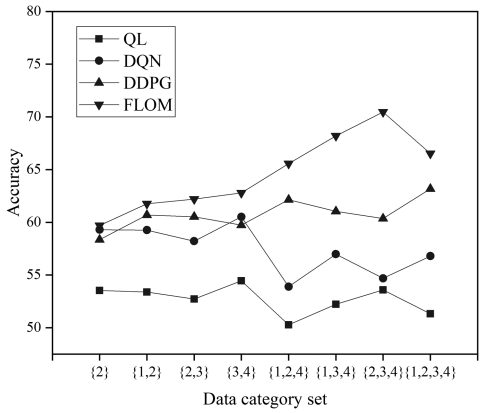


Fig. 3. Data are distributed to the corresponding mobile device according to the experimental process.

(a) First, after calculating the mean of each feature *X*, we perform the mean-removal operation on all samples; thus, we obtain the processed matrix *ZX'*.

(b) The covariance matrix *C* of *Z* samples is calculated for the *N'* dimension feature, and the formula is as follows.

$$C = Cov\left(ZX\right) = ZX' \times ZX'^{T}. \tag{3}$$

where $C \in \mathbb{R}^{N' \times N'}$ is a symmetric positive semi-definite covariance matrix, and $ZX' \in R, Z \times N'$ is the mean-centered data matrix with Z samples and N' features.

Then, the eigenvector *u* corresponding to the eigenvalues of the covariance matrix *C* is calculated by solving the characteristic equation:

$$Cu = \lambda u. \tag{4}$$

where $C \in \mathbb{R}^{N' \times N'}$ is the covariance matrix from (3); $u \in \mathbb{R}^{N'}$ is an eigenvector with $\mathrm{L2}(\mathbf{u}) = 1$; $\lambda \in \mathbb{R}$ is the corresponding non-negative eigenvalue. The eigenvalues $\lambda_1, \lambda_2, \ldots, \lambda_{N'}$ are ordered such that $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_{N'} \geq 0$.

We select the top *k* eigenvectors corresponding to the *k* largest eigenvalues, where *k* is determined by the cumulative explained variance ratio:

$$\frac{\sum_{i=1}^{k} \lambda_i}{\sum_{i=1}^{N'} \lambda_i} \geq 0.95. \tag{5}$$

This ensures that 95% of the data variance is preserved while achieving dimensionality reduction.

These λ values are sorted in descending order, and finally, we remove the corresponding *k* eigenvectors. Then, we can obtain a set of feature groups $\{(\gamma_1, \mu_1), (\gamma_2, \mu_2), \ldots, (\gamma_k, \mu_k)\}$, where the eigenvectors $\{\gamma_1, \gamma_2, \ldots \gamma_k\}$ form the eigenvector matrix *V*. Then, we project the original feature $(x_1^i, x_2^i, .., x_n^i)^T$ through this feature group to obtain the new feature $(y_1^i, y_2^i, .., y_n^i)^T$, and its calculation formula is as follows:

$$Y = ZX' \times P. \tag{6}$$

By using the PCA technology in FLOM to preprocess the input data, the main feature information for data classification is obtained, which lays the foundation for the subsequent directional assignment to mobile devices of specific categories.

2) Model Self-Evolution Mechanism Via Deep Reinforcement Learning: To adapt dynamically to evolving conditions in the mobile cloud computing environment, the FLOM framework employs a self-evolution mechanism based on deep reinforcement learning (DRL). This approach enables continuous model updates and optimizations, leveraging real-time data from mobile devices. By utilizing DRL, the system can autonomously adjust to fluctuations in network conditions, resource availability, and task demands, ensuring that both local and global models remain relevant and efficient across all devices in the network.

The proposed DRL-based approach also governs the assignment of Quality Value *(Q)* for each service task, which serves as a priority or quality metric that helps guide task allocation to suitable mobile devices. In this context, *Q* values are dynamically assigned based on each device's current state and capabilities, and they reflect the importance or urgency of processing each

task. This process can be formulated as a Markov Decision Process (MDP), where each mobile device's state, action, and reward contribute to determining the optimal *Q* value for incoming tasks.

The MDP formulation for the self-evolution process is as follows:

- *State (S):* Represents the current status of each mobile device, including attributes such as processing power, resource availability, and recent task performance.

- *Action (A):* Refers to the allocation of specific tasks with defined *Q* values to mobile devices, where each action influences the overall processing efficiency.

- *Reward (R):* Defined as a measure of task processing performance, which increases when a task with a high *Q* value is completed with low latency and high accuracy.

The transition probability equation for the state transitions in the MDP is given by:

$$P_{dd'}^{a} = P[D'_{t+1} = d' | D_t = d, A_t = a]. \tag{7}$$

where D_t represents the current accuracy level or performance metric of each mobile device, and D'_{t+1} denotes the updated state after an action a (task allocation with a specific *Q* value) is taken. This formulation allows the FLOM framework to learn optimal policies for Q value assignment, ensuring that tasks are prioritized and allocated based on device capabilities and task demands.

By integrating this *Q* value assignment process within the DRL-based self-evolution mechanism, the FLOM framework achieves adaptive resource allocation that aligns with both real-time device states and evolving task requirements. This enables the system to dynamically distribute high-priority tasks to capable devices, thereby maximizing processing efficiency and maintaining quality of service.

Then, to judge the pros and cons of the obtained current state, we can introduce the state parameter reward *R* to represent the pros and cons of state *t* at a certain moment. The formula is as follows:

$$M_t = R_{t+1} + \delta R_{t+2} + \ldots = \sum_{k=0}^{\infty} \delta^k R_{t+k+1}. \tag{8}$$

where δ represents the influencing factor participating in the calculation, and its value is generally set to be less than 1. The greater the feedback degree of the current state and the longer the time is, the smaller the value, where M_t represents the sum of the state reward values possessed by each state at time *t*.

However, it is impossible for us to obtain all the state reward values in the current state, so we introduce the concept of a value function and obtain the state reward value M_t by substituting M_t into the value function equation. The expectation of the future reward value in the state S_t is calculated as follows:

$$v(s) = E[M_t | S_t = s]. \tag{9}$$

From formulas (7) and (8), we can obtain the final form of the value function for the sub-Markov process, which can be obtained from the expectation of the immediate reward of the current state and the probability distribution of the state at the next moment. We use s' to represent any possible state of s at the next moment, which can be expressed as:

$$v(s) = R_s + \delta \sum_{s' \in S} P_{ss'v(s')}. \tag{10}$$

where $s \in S$ represents the current state from the finite state space *S*; $R_s \in [-R\max, R\max]$ is the immediate reward for state *s*, bounded by $R_{\max} = 100$; $\delta \in (0, 1)$ is the discount factor, set to $\delta = 0.8$in our experiments; $Pss' \in [0, 1]$ is the state transition probability from state s to state s', satisfying $\sum(s' \in s)Pss' = 1$; $v(s')$ is the value of successor state s'.

The reward function R_s is specifically defined as:

$R_s = wreward \cdot QoS_s - wpenalty \cdot (Delay_s / DelayR_{\max} + ResourceUtilization_s)$ where *wreward* = 10, *wpenalty* = 5, $DelayR_{\max} = 1000$ ms, ensuring that high QoS states receive positive rewards while states with excessive delays or resource consumption are penalized.

The value function converges when $|v(k+1)(s) - v(k)(s)| < \varepsilon$ for all $s \in S$, where $\varepsilon = 10^{-4}$ is the convergence threshold.

To solve the value of the value function in each sub-Markov process, we can use a matrix to represent the size of the feedback degree in different states.

$$\begin{bmatrix} v(1) \\ \vdots \\ v(n) \end{bmatrix} = \begin{bmatrix} R_1 \\ \vdots \\ R_n \end{bmatrix} + \delta \begin{bmatrix} P_{11} & \ldots & P_{1n} \\ \vdots & \ddots & \vdots \\ P_{n1} & \ldots & P_{nn} \end{bmatrix}. \tag{11}$$

where *v* is a column vector representing an entire state, and its mathematical formula can be abstracted as follows:

$$v = R + \delta P_v. \tag{12}$$

Combined with the Markov state-value function in the above formula, to obtain the optimal strategy in the process of model self-evolution, we can use the value iteration approach to determine the optimal solution. Then, the optimal value iteration equation based on the Bellman equation can be expressed as follows:

$$\begin{aligned} v_{k+1}(s) &= \max_x E\left[R_{t+1} + \delta v_k\left(S_{t+1}\right) | S_t = s, A_t = a\right] \\ &= \max_a \sum_{s',r} p(s', r | s, a)\left[r + \delta v_k\left(s'\right)\right]. \end{aligned} \tag{13}$$

We use the value iteration approach to follow the new value, and the final value is obtained by convergence, where v_k. As long as the model converges, we can obtain the optimal policy because we use greedy thinking for action selection, and each step obtains feedback from the current state.

In the mobile cloud computing environment, to achieve the optimal effect of the algorithm model, we can use the optimization strategy π with parameters for calculation to determine what action value a to take at different times *t*, where the neural network parameter value θ can adjust the weights, biases and other parameters in the neural network for optimization. We can use formula (12) to express the random strategy adopted in the optimization process:

$$a_t \tilde{} \pi_\theta(\cdot | s_t). \tag{14}$$

Because the system can preform a variety of different actions when it is in the same state, its feedback values must be different. To achieve self-evolution of the algorithm model, we can use a greedy strategy for action selection and select the action with the largest feedback value each time. According to our definition above, the action value function A can be obtained as follows:

$$A^{\pi} = E_{s'}[r + \delta A^{\pi}(s', a') | s, a]. \tag{15}$$

where π represents the action value under the current strategy, indicating that each action a we take in state s is generated by the strategy according to the current state.

We can obtain the optimal action value function by combining the formula of the above value function and according to formula (14), thus ensuring the uniqueness of the value and achieving the goal of solving the sub-MDP process and realizing the optimal solution of the local value.

$$A^{*}(s, a) = E_{s'}[r + \delta \max_{a'} A^{*}(s', a') | s, a]. \tag{16}$$

Since our input data are high-dimensional data, if we want to simulate a series of information, such as all states, actions, and feedback values, from input to output, we will inevitably encounter the curse of dimensionality. Therefore, we can use a deep neural network to approximate the value of the value function, and the formula is as follows:

$$A(s, a) \approx f(s, a, \omega). \tag{17}$$

where ω represents a distribution of the approximate state-action values obtained after the data are output by the neural network, i.e., a vector representing the state-action values $[A(s, a_1), A(s, a_2), \ldots, A(s, a_n)]$, so the above formula can be transformed into $\mathrm{A}(\mathrm{s}, \mathrm{a}; \theta) \approx A^{*}(s, a)$, where θ represents the model parameters. In the process of real-time interaction with the outside world through the algorithm model, the iterative and long-term trial-and-error process can achieve the optimization of the revenue function.

Then, we can transform the self-evolution problem of the model into the optimization problem of the neural network and adjust the deviation between the label and the network output by using the gradient descent approach through backpropagation to minimize the loss function, so we can obtain the neural network. The loss function gradient formula of the network is:

$$\frac{\partial \mathrm{L}(\mathrm{w})}{\partial \mathrm{w}} = \mathrm{E}\left[\left(\mathrm{r} + \delta \max_{a'} A(s', a') - A(s, a)\right) \frac{\partial \mathrm{A}(\mathrm{s}, \mathrm{a}, \mathrm{w})}{\partial \mathrm{w}}\right]. \tag{18}$$

Thus far, we can obtain the preformula for the self-evolution of the algorithm model in the mobile cloud computing environment as follows:

$$\mathrm{A}(\mathrm{s}, \mathrm{a},) \leftarrow \mathrm{A}(\mathrm{s}, \mathrm{a}) + \alpha\left[\mathrm{r} + \delta \max_{a'} A(s', a') - A(s, a)\right]. \tag{19}$$

where α is the learning rate between 0 and 1. Since a deep neural network is used for the calculation of the A value, we can obtain detailed information on the parameter θ of the neural network for the *kth* sample value, where the bias parameter is $[\theta_{q1}, \theta_{q2}, \ldots, \theta_{qX(lay+1)}]$, the weight parameter is $[\theta_{x1}, \theta_{x2}, \ldots, \theta_{xX(lay+1)}]$, and $X(lay+1)$ is the neural network. The number of hidden units in the middle layer, $\tau(s, a)$, is the activation function matrix of the neural network, so the following formula can be obtained:

$$\begin{aligned} Q_{\theta}(s, a) &= \theta^{T} \cdot \tau(s, a) \\ &= \sum_{x=1}^{X(lay)} \sum_{y=1}^{X(lay+1)} \left(\theta_{xy \cdot mx}(s, a) + \theta_{qy}\right). \end{aligned} \tag{20}$$

To update the network parameter θ, we randomly extract the number of samples from the memory network to update the policy, where the value is *Y*, and the rules are as follows:

$$\theta \leftarrow \theta - \varepsilon_{\theta} \sum_{m=1}^{Y} \delta_m, TD \nabla_{\theta} Q_{m,\theta}(s, a). \tag{21}$$

where ε_{θ} is the step size set by stochastic gradient descent. The memory network and the target network have the same network structure and different parameters. We can obtain $\mathrm{A}(\mathrm{s}, \mathrm{a}, \theta)$ by formula (20) in the gradient-specific steps in the descent process.

$$\begin{aligned} \nabla_{\theta_i} L_i(\theta_i) &= E_{s, a \sim \rho(.); s' \sim \varepsilon} \\ &\left[\left(r + \delta \max_{a'} A(s, a; \theta_i)\right) \nabla_{\theta_i} Q(s, a; \theta_i)\right]. \end{aligned} \tag{22}$$

When the algorithm model starts training, the deep neural network generates the value *A*, and then the SGD algorithm works at the same time. The data generated by the neural network are not independent and identically distributed, so we cannot use it immediately but need to store it in the experience pool first and disrupt the correlation between the generated data by random sampling. Furthermore, the risk of model oscillation and divergence caused by the neural network that generates data labels and the neural network that performs value calculation is avoided. We need to construct an independent and novel neural network based on the value computing network. Therefore, in the FLOM framework, the self-evolution of the algorithm model is completed.

The specific process of the FLOM self-evolution workflow for adaptive resource management is as follows.

C. Discussion

The FLOM framework advances mobile edge computing by enabling adaptive, efficient task distribution and dynamic system optimization. A key feature of FLOM is its ability to minimize computational overhead through dimensionality reduction techniques while ensuring optimal resource utilization. This balance enhances both performance and scalability across diverse, dynamic environments. FLOM employs dual-layer optimization to address the challenges of heterogeneity and resource constraints. At the local level, edge devices dynamically refine task execution to reduce latency, while at the global level, the cloud synthesizes updates to maintain system coherence. This hierarchical strategy ensures robust performance under varying network conditions and device capabilities. During our experimental evaluation, key parameters were calibrated to optimize the trade-offs between responsiveness, energy efficiency, and accuracy. Specifically, *m*

Algorithm 1: Self-Evolution Workflow of FLOM for Adaptive Resource Management.

Input: Initial dataset P, baseline model M, mobile devices $\{Device_i\}$, DRL policy parameters π
Output: Optimized global model M and adaptive resource allocation policy π
Stage 1: Initialization and Model Distribution;
Train the central model M on initial dataset P to establish a baseline;
Segment model M into $\{m_k\}$ and distribute segments to each mobile device $\{Device_i\}$ according to task types $\{SupportedTasks_i\}$;
Stage 2: Task Categorization and Allocation Based on Priority;
foreach *new incoming task* **do**
    Compute task category probabilities $\{CategoryProbability_w\}$;
    Assign Quality Value (Q) to task based on urgency and processing priority;
    Select the mobile device $Device_i$ best suited to handle the task's category and Q value;
    Dispatch task to $Device_i$ for processing based on allocation results;
Stage 3: Local Processing and Adaptive Model Updating on $Device_i$;
foreach *task received by* $Device_i$ **do**
    Observe current state S of $Device_i$ (includes resources, load, and recent performance);
    Apply DRL policy π to determine optimal action A for task execution;
    Execute action A, perform task processing and update local model segment m_i;
    Store real-time data $P_{i,l}^{\beta}$ generated from processing on $Device_i$;
    Save incremental updates of m_i for synchronization with cloud;
Stage 4: Cloud Synchronization and Central Model Refinement;
if *synchronization interval is reached* **then**
    foreach $Device_i$ **do**
        Transmit local model updates m_i to cloud data center;
    Aggregate local updates $\{m_i\}$ and analyze differential information with M;
    Perform weighted retraining of M to reflect global optimizations from collective updates;
Stage 5: Self-Evolution Policy Refinement via DRL;
Update DRL policy π using cumulative reinforcement signals based on task completion metrics;
foreach *task* **do**
    Reward R derived from Q values, task latency, and output accuracy;
    Update policy π to prioritize high-Q tasks and adapt resource allocation dynamically;
Loop Continuation;
Repeat steps 5-27 for continuous adaptation in response to new tasks and changing device states;
return *Refined global model M and DRL-based adaptive allocation policy π*;

was set to 2, t to 4, k to 0.6, and γ to 0.8. These settings reflect a balance between computational intensity and system efficiency, ensuring that FLOM remains adaptable to diverse application scenarios. Adjustments to these parameters can further tailor the framework to specific use cases, demonstrating its flexibility and real-world applicability. Overall, FLOM represents a unified, scalable solution for next-generation MEC systems. Its ability to adapt dynamically while maintaining efficiency highlights its potential to address emerging challenges in mobile edge computing.

The robustness of FLOM's performance across varying parameter settings is crucial for its practical deployment. Through comprehensive sensitivity analysis, we observed that the framework maintains stable performance even when key parameters deviate from their optimal values. Specifically, when the trade-off coefficients α and β vary within 20% of their calibrated values ($\alpha = 0.4$ and $\beta = 0.2$), the overall system performance degradation remains below 5%, demonstrating the framework's resilience to parameter variations. Similarly, the QoS weight distribution shows comparable stability, with performance metrics remaining consistent when individual weights fluctuate within reasonable bounds. This robustness is particularly important in real-world mobile cloud environments where precise parameter tuning may be challenging due to dynamic network conditions and heterogeneous device capabilities. The discount factor δ in our reinforcement learning formulation exhibits optimal convergence behavior across the range [0.75, 0.85], with our chosen value of 0.8 providing the best balance between convergence speed and solution quality. These findings confirm that FLOM is not merely optimized for specific parameter configurations but rather exhibits inherent stability that makes it suitable for diverse deployment scenarios without extensive recalibration.

V. Performance Evaluation and Analysis

The FLOM approach is proposed to address the self-evolution of the model in the mobile cloud computing environment, and the optimized model plays an important role in processing specific types of data in this field. We evaluate FLOM through the following aspects: (1) The accuracy of the FLOM processing data under normal circumstances; (2) Different sample categories are not assigned to the designated mobile terminal for processing; (3) Overall self-optimization performance analysis; (4) Comparison of time performance under the number of sample categories.

To verify the performance of our proposed self-evolutionary model in the mobile cloud computing environment, we need to collect different service request data for processing, and after preprocessing, we identify and assign the data to a mobile terminal with the corresponding service equipment. We use simulation tools to generate multiple request data in parallel, and different service resources are continuously added to the

TABLE I
PREDICTION ACCURACY FOR DIFFERENT QUANTITY CLASSES

Accuracy				
Category number collection	QL	DQN	DDPG	FLOM
2	53.52%	59.29%	58.34%	59.69%
1,2	53.38%	59.25%	60.70%	61.75%
2,3	52.72%	58.20%	60.52%	62.20%
3,4	54.45%	60.52%	59.73%	62.78%
1,2,4	50.28%	53.89%	62.15%	65.57%
1,3,4	52.23%	56.97%	61.04%	68.19%
2,3,4	53.58%	54.68%	60.36%	70.46%
1,2,3,4	51.32%	56.79%	63.18%	66.52%

cloud data centre and mobile cloud computing centre to form the data set required for self-evolving tasks. This paper divides these data into training and test data. It is worth noting that they provide a data model with high universality in the benchmark model generation stage and generate a self-optimized data model that conforms to this category according to the data in the mobile terminal application stage. The hyperparameters of all compared approaches were tuned by cross-validation. In the construction of the deep reinforcement learning model mentioned in the paper, the deep learning framework, TensorFlow, is used. Based on the proposed FLOM approach, the resource information and status of the cloud computing centre and the mobile cloud computing centre can be regularly obtained. This paper evaluates and tests the performance of FLOM through 4 different experiments. The final results show that the proposed approach has high accuracy and stability in dealing with the self-evolution problem of a model in the mobile cloud computing environment. The effect is especially obvious when dealing with specific categories under the domain.

A. Experimental Setup

To realize the intelligent self-evolution process of FLOM, we use the data collection script to obtain stock data required for the experiment and set the data update period to 1 h. The host configuration used in our cloud computing centre is (an Intel Dual-core CPU (3.7 GHZ), 4 G main memory, 1 T hard disk), each mobile cloud computing centre is a Google nexus mobile phone. Furthermore, the MATLAB simulation platform is used to obtain satisfactory experimental results and good simulation performance. Different types of service requests are generated, FLOM is applied, and the model's performance and resource status are monitored regularly.

B. Accuracy of FLOM Processing Data Under Normal Circumstances

In this experiment, we compare the performance of four approaches in handling different types of data, verifying the accuracy of the proposed FLOM approach across various data categories in the mobile cloud computing environment. Table I summarizes the model's accuracy for different data categories. First, the required data is selected through a collection script, divided into multiple domains by category, and the accuracies

TABLE II
MODEL ACCURACY FOR DATA ANOMALY ASSIGNMENT

Accuracy					
Data error distribution	1 mistake	2 mistakes	3 mistakes	4 mistakes	FLOM
20%	65.35%	64.68%	64.32%	63.24%	65.31%
40%	66.72%	65.98%	66.31%	64.59%	67.65%
60%	65.76%	68.34%	67.98%	65.32%	70.47%
80%	67.46%	68.93%	68.76%	67.63%	72.95%
100%	68.27%	69.21%	69.86%	68.92%	71.65%

of FLOM and other algorithms are compared for both homogeneous and heterogeneous data types. As shown in Fig. 3, the accuracy difference between DQN and FLOM is less than 0.5% only in the single-class data domain. DQN performs well here, outperforming DDPG, which struggles with continuous data processing. As the number of data domains increases, DDPG's accuracy improves slightly but remains lower than FLOM's, while the gap between FLOM and QL/DQN widens. As more data categories are introduced, the performance of all approaches improves, though the gains for QL and DQN diminish and stabilize. FLOM continues to improve as long as the data domain size does not reach the upper limit of mobile terminal capacity. Notably, the performance improvement for each approach decreases with each additional data domain, but FLOM consistently maintains its advantage as long as the number of categories does not exceed the capacity of the mobile cloud computing centers. When processing single-category data, all approaches show similar accuracy, but as the multi-category data domain expands, QL, DQN, and DDPG exhibit distinct but relatively similar accuracies. FLOM consistently outperforms all, confirming that its performance improves across different data categories in the same domain. Overall, FLOM yields the best results in every experimental scenario, with other approaches showing varying performance for single-class and multi-class data. This experiment demonstrates FLOM's superior ability to process diverse data types within the same domain.

C. Misassignment of Each Sample Category

In this set of experiments, we verify the changes in the accuracy of the model when different sample categories are not assigned to the mobile terminal that handles the corresponding tasks. In this experiment, to determine whether the sample data are correctly assigned being processed by FLOM, in the process of increasing the number of sample categories, a control experiment with service data correctly assigned to the corresponding mobile terminal and incorrectly assigned service data is carried out. Since there are many cases of incorrectly assigned data in the domain, we average the obtained accuracies, observe only the variation trend of the experimental results and ignore their data detail performance. The detailed numerical results are summarized in Table II. Then, the results of each group of experiments are compared with the results of the FLOM approach to obtain the performance changes of different categories of data dislocation in the same field. As shown in Fig. 4, when the number of sample categories is 1, after the sample data are obtained by the mobile terminal, because the data category domain is

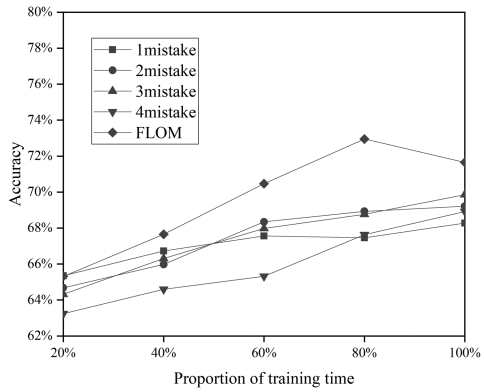


Fig. 4. Sample class error distribution trial procedure.

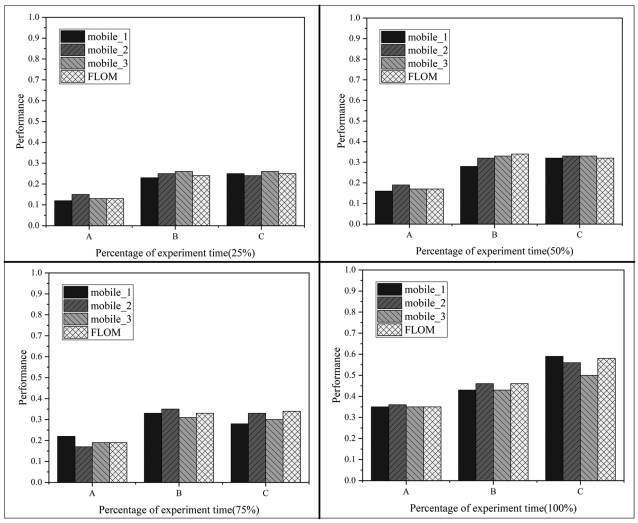


Fig. 5. Performance comparison of the benchmark model after iteration.

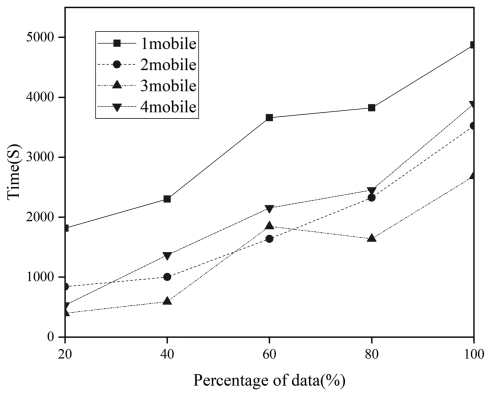


Fig. 6. Comparison of the time performance for each sample category.

different from the data model processed by the mobile cloud computing centre, the sample processing performance is poor in the initial stage, but as the running time of the system increases, the mobile cloud computing centre obtains the characteristics of the sample data. Due to the self-evolution characteristics of FLOM, this type of data can be processed by the mobile terminal, and the performance of the model is significantly improved. Obviously, since the number of sample categories is 2 and 3, there are more allocation situations in the process of allocating mobile terminals. The performance improvement obtained by FLOM is in line with the trend when the number of sample categories is 1, but the time spent increases as the sample size increases. When the number of data sample categories exceeds the number of mobile cloud computing centres in the experiment, the performance of FLOM still has an upwards trend, but it fluctuates greatly during the rising period. Since the FLOM approach of the control group handles the sample data correctly, its model performance is always higher than the case when the data categories are dislocated. However, when the number of categories processed exceeds the upper limit set by the experiment, FLOM also has the ability to automatically adjust the settings. Due to the characteristics of evolution, the reissued benchmark model can fully fit the greatest common divisor of the characteristics of each type of data, and with a partial decrease in performance, the ability to process the input data of the sample is improved. According to the results of this group of experiments, the proposed intelligent self-evolution approach has good experimental results when dealing with dislocated data categories due to the characteristics of the FLOM design.

D. Overall Self-Optimizing Performance Analysis

In this experiment, we compare the performance of the benchmark algorithm in FLOM before and after self-optimization to simulate the proposed framework's performance in a mobile cloud computing environment. We analyze how the benchmark model affects the performance of each mobile terminal and how it evolves over time. As shown in Fig. 5, when a mobile terminal receives data without a benchmark model, it starts training the algorithm model from scratch. Over time, each terminal optimizes local performance when processing its data, showing significant improvement, but the overall performance gain of FLOM is slightly lower than in the other two scenarios. Once each mobile cloud computing centre obtains the distributed benchmark model, the initial data processing accuracy of each terminal is higher than in the first case. However, since the second scenario does not integrate local algorithm models into a self-optimized framework, FLOM's overall self-evolution process remains incomplete. As a result, the third solution, after completing the self-optimization process, yields better performance than the second solution. Experimental results also show that when the benchmark model undergoes self-optimization and is reissued, the performance of each mobile device initially lags behind the other solutions but gradually surpasses them over time. The updated data processing performance of the mobile terminal after FLOM iteration aligns with our expectations.

E. Comparison of Time Performance for Different Numbers of Sample Categories

This experiment verifies the time performance of the proposed FLOM approach in the experimental environment by observing the running time of the system for different number of sample categories as the number of mobile cloud computing processing centres increases in the mobile cloud computing environment,

TABLE III
TIME PERFORMANCE FOR DIFFERENT MOBILE TERMINALS

Accuracy				
Number of mobile terminals	1 mobile terminal	2 mobile terminals	3 mobile terminals	4 mobile terminals
20	1816.738	840.356	397.779	532.726
40	2302.913	1002.436	591.56	1368.2348
60	3659.113	1638.693	1846.862	2153.346
80	3824.446	2324.932	1640.966	2453.382
100	4872.257	3523.207	2686.227	3891.342

as shown in Fig. 6. Because the running time is an important indicator of the performance of the reaction system, the running time should be as short as possible to save computing resources and reduce the application threshold of FLOM to make it easier to popularize and use.

Table III shows the running times of different mobile cloud computing centres for various sample sizes. When the sample size is 20%, Plan 3 takes the least time, while Plan 2 takes the most, followed by Plan 1. This is because, in Plan 3, the mobile terminals match the sample data categories, minimizing processing time, whereas the other solutions require more time to distribute the data due to the absence of corresponding mobile cloud computing centres. As the number of mobile processing centres exceeds the number of input sample categories, the performance of Plan 4 closely follows that of Plan 3, but due to its directional processing setup, it takes slightly more time. As shown in the figure below, as the proportion of sample data increases, system time consumption rises, with all four plans showing an upward trend. Notably, when the number of data categories matches the number of mobile cloud computing centres in Plan 2, the system time consumption is significantly reduced. Plan 3, representing our proposed FLOM approach, demonstrates the best time performance, with minimal time consumption and a gradual growth curve, highlighting the stability of FLOM's performance.

VI. CONCLUSION AND FUTURE WORK

This paper introduces FLOM, an innovative self-evolving framework for mobile cloud computing. By combining deep reinforcement learning with targeted data processing, FLOM optimizes both data allocation and computational efficiency in dynamic mobile cloud environments. Experimental results validate its superior performance in handling limited data while maintaining high accuracy across mobile cloud centers. FLOM represents a significant step towards scalable, adaptive, and resource-efficient mobile cloud computing systems, offering a robust solution for data processing and model optimization.

Future research will focus on extending FLOM to multi-cloud and multi-mobile cloud environments, enabling dynamic coordination between diverse computing centers. Key areas of exploration include reducing training time, optimizing energy consumption, and improving model transfer efficiency. These advancements will enhance FLOM's scalability and applicability in large-scale, real-world cloud computing scenarios, pushing the boundaries of intelligent mobile cloud systems.

REFERENCES

[1] X. Xia, F. Chen, Q. He, J. C. Grundy, M. Abdelrazek, and H. Jin, "Cost-effective app data distribution in edge computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 1, pp. 31–44, Jan. 2021.

[2] X. Tang et al., "Cost-efficient workflow scheduling algorithm for applications with deadline constraint on heterogeneous clouds," *IEEE Trans. Parallel Distrib. Syst.*, vol. 33, no. 9, pp. 2079–2092, Sep. 2022.

[3] H. Chen, W. Tian, P. Wang, F. Wang, L. Xiong, and H. Li, "EPRO-PNP: Generalized end-to-end probabilistic perspective-N-points for monocular object pose estimation," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2023, pp. 2781–2790, doi: 10.48550/arXiv.2303.12787.

[4] S. Starke, I. Mason, and T. Komura, "Deepphase: Periodic autoencoders for learning motion phase manifolds," *ACM Trans. Graph.*, vol. 41, no. 4, pp. 136:1–136:13, 2022.

[5] J. Zeng, D. Ding, K. Kang, H. Xie, and Q. Yin, "Adaptive DRL-based virtual machine consolidation in energy-efficient cloud data center," *IEEE Trans. Parallel Distrib. Syst.*, vol. 33, no. 11, pp. 2991–3002, Nov. 2022.

[6] X. Ye, Y. Sun, D. Wen, G. Pan, and S. Zhang, "End-to-end delay minimization based on joint optimization of DNN partitioning and resource allocation for cooperative edge inference," in *Proc. 98th IEEE Veh. Technol. Conf. (VTC Fall 2023)*, Hong Kong, SAR, China, 2023, pp. 1–7.

[7] J. Tang, S. Wang, S. Yang, Y. Xiang, and Z. Zhou, "Federated learning-assisted task offloading based on feature matching and caching in collaborative device-edge-cloud networks," *IEEE Trans. Mobile Comput.*, vol. 23, no. 12, pp. 12061–12079, Dec. 2024.

[8] Z. Chen, B. Xiong, X. Chen, G. Min, and J. Li, "Joint computation offloading and resource allocation in multi-edge smart communities with personalized federated deep reinforcement learning," *IEEE Trans. Mobile Comput.*, vol. 23, no. 12, pp. 11604–11619, Dec. 2024.

[9] S. Hu, M. Li, J. Gao, C. Zhou, and X. Shen, "Adaptive device-edge collaboration on DNN inference in AIoT: A digital-twin-assisted approach," *IEEE Internet Things J.*, vol. 11, no. 7, pp. 12893–12908, Apr. 2024.

[10] L. Yang, X. Shen, C. Zhong, and Y. Liao, "On-demand inference acceleration for directed acyclic graph neural networks over edge-cloud collaboration," *J. Parallel Distrib. Comput.*, vol. 171, pp. 79–87, 2023.

[11] L. Wang et al., "Adaevo: Edge-assisted continuous and timely DNN model evolution for mobile devices," *IEEE Trans. Mobile Comput.*, vol. 24, no. 4, pp. 2485–2503, Apr. 2025.

[12] Z. Liu, M. Tian, M. Dong, X. Wang, C. Qiu, and C. Zhang, "MoEI: Mobility-aware edge inference based on model partition and service migration," *IEEE Trans. Mobile Comput.*, vol. 23, no. 10, pp. 9437–9450, Oct. 2024.

[13] P. Li, E. Koyuncu, and H. Seferoglu, "Adaptive and resilient model-distributed inference in edge computing systems," *IEEE Open J. Commun. Soc.*, vol. 4, pp. 1263–1273, 2023.

[14] N. Agarwal and S. Joshi, "Federated learning-based task offloading in a UAV-aided cloud computing mobile network," *IEEE Trans. Veh. Technol.*, vol. 73, no. 10, pp. 15751–15756, Oct. 2024.

[15] K. Peng, P. Xiao, S. Wang, and V. C. M. Leung, "SCOF: Security-aware computation offloading using federated reinforcement learning in industrial Internet of Things with edge computing," *IEEE Trans. Serv. Comput.*, vol. 17, no. 4, pp. 1780–1792, Jul./Aug. 2024.

[16] Y. Wang, Y. He, F. R. Yu, K. Wu, and S. Chen, "Intelligence-based reinforcement learning for dynamic resource optimization in edge computing-enabled vehicular networks," *IEEE Trans. Mobile Comput.*, vol. 24, no. 3, pp. 2394–2406, Mar. 2025.

[17] X. Zhou, J. Yang, Y. Li, S. Li, Z. Su, and J. Lu, "EC-TRL: Evolutionary-weighted clustering and transformer-augmented reinforcement learning for dynamic resource scheduling in edge cloud environments," *IEEE Internet Things J.*, vol. 12, no. 6, pp. 7503–7517, Mar. 2025.

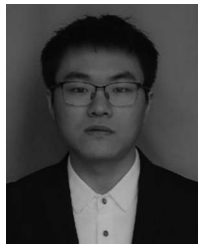
[18] X. Yang et al., "Adaptive DNN surgery for selfish inference acceleration with on-demand edge resource," 2023, *arXiv:2306.12185*.

[19] Q. Wu, Y. Zhang, C. Yang, and J. Sun, "Joint optimization of model partitioning and resource allocation for edge computing with intermittently operating devices," in *Proc. 29th IEEE Int. Conf. Parallel Distrib. Syst.*, Ocean Flower Island, China, 2023, pp. 2186–2193.

[20] W. Gao, Z. Zhao, G. Min, Q. Ni, and Y. Jiang, "Resource allocation for latency-aware federated learning in industrial Internet of Things," *IEEE Trans. Ind. Informat.*, vol. 17, no. 12, pp. 8505–8513, Dec. 2021.

[21] Z. Zhao, M. Feng, C. Ke, Z. Chen, and T. Jiang, "Federated deep recurrent q-learning for task partitioning and resource allocation in satellite mobile-edge-computing-assisted industrial IoT," *IEEE Internet Things J.*, vol. 11, no. 15, pp. 26444–26458, Aug. 2024.
[22] C. Li, M. Song, and Y. Luo, "Federated learning based on stackelberg game in unmanned-aerial-vehicle-enabled mobile edge computing," *Expert Syst. Appl.*, vol. 235, 2024, Art. no. 121023.
[23] M. K. Hasan et al., "Federated learning for computational offloading and resource management of vehicular edge computing in 6G-V2X network," *IEEE Trans. Consum. Electron.*, vol. 70, no. 1, pp. 3827–3847, Feb. 2024.
[24] S. Huang, Z. Zhang, S. Wang, R. Wang, and K. Huang, "Accelerating federated edge learning via topology optimization," *IEEE Internet Things J.*, vol. 10, no. 3, pp. 2056–2070, Feb. 2023.
[25] J. Mendez, K. Bierzynski, M. P. Cuéllar, and D. P. Morales, "Edge intelligence: Concepts, architectures, applications, and future directions," *ACM Trans. Embedded Comput. Syst.*, vol. 21, no. 5, pp. 48:1–48:41, 2022.
[26] J. Zhao, H. Quan, M. Xia, and D. Wang, "Adaptive resource allocation for mobile edge computing in Internet of Vehicles: A deep reinforcement learning approach," *IEEE Trans. Veh. Technol.*, vol. 73, no. 4, pp. 5834–5848, Apr. 2024.
[27] C. Deng, X. Fang, and X. Wang, "UAV-enabled mobile-edge computing for AI applications: Joint model decision, resource allocation, and trajectory optimization," *IEEE Internet Things J.*, vol. 10, no. 7, pp. 5662–5675, Apr. 2023.
[28] Y. Xiao, Y. Song, and J. Liu, "Collaborative multi-agent deep reinforcement learning for energy-efficient resource allocation in heterogeneous mobile edge computing networks," *IEEE Trans. Wireless Commun.*, vol. 23, no. 6, pp. 6653–6668, Jun. 2024.
[29] S. Pang, T. Wang, H. Gui, X. He, and L. Hou, "An intelligent task offloading method based on multi-agent deep reinforcement learning in ultra-dense heterogeneous network with mobile edge computing," *Comput. Netw.*, vol. 250, 2024, Art. no. 110555.
[30] L. Zhang, G. Gao, H. Yin, and H. Zhang, "Multi-edge reinforced collaborative data acquisition for continuous video analytics by prioritizing quality over quantity," in *Proc. AAAI-25, Sponsored Assoc. Advance. Artif. Intell.*, Philadelphia, PA, USA, T. Walsh, J. Shah, and Z. Kolter, Eds., 2025, pp. 1084–1092.
[31] W. Zou, Z. Zhang, N. Wang, X. Tan, and L. Tian, "A time-saving task scheduling algorithm based on deep reinforcement learning for edge cloud collaborative computing," in *Proc. 99th IEEE Veh. Technol. Conf., VTC Spring 2024*, Singapore, 2024, pp. 1–6.
[32] L. Tan, Z. Kuang, L. Zhao, and A. Liu, "Energy-efficient joint task offloading and resource allocation in OFDMA-based collaborative edge computing," *IEEE Trans. Wireless Commun.*, vol. 21, no. 3, pp. 1960–1972, Mar. 2022.
[33] Z. Wang, M. Goudarzi, and R. Buyya, "TF-DDRL: A transformer-enhanced distributed DRL technique for scheduling IoT applications in edge and cloud computing environments," *IEEE Trans. Serv. Comput.*, vol. 18, no. 2, pp. 1039–1053, Mar./Apr. 2025.
[34] S. S. Gill et al., "Edge AI: A taxonomy, systematic review and future directions," *Cluster Comput.*, vol. 28, no. 1, pp. 18:1–18:53, 2025.

Jia Zhao (Member, IEEE) is currently a professor with the College of Artificial Intelligence Technology & School of Computer Technology and Engineering, Changchun Institute of Technology, China. His main research interests include distributed system, mobile cloud computing, artificial intelligence, Big Data, and blockchain. He has led and participated in several projects and authored or coauthored more than 50 journal and conference papers in the above areas.



Yafei Zhu is a graduate student with the School of Computer Science and Engineering, Changchun University of Technology. His main research interests include mobile cloud computing, deep learning, reinforcement learning, Big Data technology analysis and application, image recognition. He participated in the research of several scientific research projects and authored or coauthored articles in the above fields.

Keqin Li (Fellow, IEEE) received the BS degree in computer science from Tsinghua University, in 1985, and the PhD degree in computer science from the University of Houston, in 1990. He is currently a SUNY distinguished professor with the State University of New York, and a National distinguished professor with Hunan University, China. He has authored or coauthored more than 1130 journal articles, book chapters, and refereed conference papers. Since 2020, he has been among the world's top few most influential scientists in parallel and distributed computing regarding single-year impact (ranked #2) and career-long impact (ranked #4) based on a composite indicator of the scopus citation database. He is listed in ScholarGPS Highly Ranked Scholars from 2022 to 2024 and is among the top 0.002% out of over 30 million scholars worldwide based on a composite score of three ranking metrics for research productivity, impact, and quality in the recent five years. He was the recipient of the IEEE TCCLD Research Impact Award from the IEEE CS Technical Committee on Cloud Computing in 2022 and the IEEE TCSVC Research Innovation Award from the IEEE CS Technical Community on Services Computing in 2023. He is a fellow of AAAS, AAIA, ACIS, and AIIA. He is a member of Academia Europaea (Academician of the Academy of Europe).

Ming Hu (Member, IEEE) received the BS degree in computer application from the Changchun University of Technology, in 1985, and the MA and PhD degrees in computer science and technology from Jilin University, in 1998 and 2005, respectively. As a person in charge or a principal participant, he has finished more than 30 national, provincial and ministerial level research projects of China and authored or coauthored more than 30 journal and conference papers.

Yan Ding (Member, IEEE) is currently an associate professor with the School of Computer Technology and Engineering, Changchun Institute of Technology, China. His main research interests include distributed systems, mobile cloud computing, the Internet of Things, artificial intelligence application, Big Data, and blockchain. He has led and participated in several projects and authored or coauthored more than 30 journal and conference papers in the above areas.