JOURNAL OF INFORMATION SCIENCE AND ENGINEERING 40, 27-39 (2024)
DOI: 10.6688/JISE.202401_40(1).0002

Privacy Block-Streaming: A Novel DEX File Loading Scheme Based on Federated Learning*

YICHUAN WANG[1,2], YANHUA FENG[1], YEQIU XIAO[1,+], XIAOXUE LIU[1] AND XINHONG HEI[1,2]
[1]*School of Computer Science and Engineering*
[2]*Shaanxi Key Laboratory for Network Computing and Security*
Xi'an University of Technology
Xi'an, 710048 P.R. China
E-mail: chuan@xaut.edu.cn; yhfeng@stu.xaut.edu.cn; xiaoyeqiu@xaut.edu.cn+; liuxiaoxue@xaut.edu.cn; heixinhong@xaut.edu.cn

With the technologies of wireless mobile networks and wireless devices, users have faced a severe issue of data overload in application usage. Preference recommendation of applications is viewed as an effective method to solve such issues, which mainly relies on users' interests in an application. However, preference recommendation of applications ignores what functions users like to utilize. This paper aims to address the data overload issue by analyzing users' preference of functional classes for devices in wireless mobile networks and proposes a privacy scheme of block-stream service for DEX files based on federated learning. We first propose a formalized model of application functions to provide insight into applications. We calculate correlation weights of functions in mobile applications by using users' operation behavior data. The homomorphic encryption technologies are also utilized to prevent privacy leakage of model parameters in this paper. Finally, models involved in the proposed scheme are stored in the blockchain to support trusted storage and traceability services. Experiment results are presented to verify the effectiveness of our proposed scheme. Compared with other block-streaming loading schemes, the proposed scheme in this paper has a pretty good performance in saving storage space.

Keywords: federated learning, homomorphic encryption, block-streaming service loading, blockchain, DEX file

1. INTRODUCTION

With the development of wireless mobile networks technology and the explosive growth of application usage of wireless mobile terminals, applications (Apps) have been deeply involved in human being's life, which results in the fact that people are surrounded by massive volumes of data [1]. It is significant for Apps to analyze and recommend items that users are potentially interested in from such complex information. Thus, recommendation systems for Apps have been attracting extensive attention in recent years [2]. Traditionally, a recommendation system provides users' preferences by collecting history data of users and analyzing them in a server, which always contains sensitive information of users. Unfortunately, different kinds of malicious attacks exist and bring various threats to data privacy [3]. Therefore, how to protect data privacy in recommendation systems serves as a critical issue and needs to be carefully considered.

Federated learning (FL) is a novel framework of distributed machine learning, which has a good performance on privacy protection of data. Its fundamental principle is to transmit trained data ($e.g.$, local model parameters) instead of original ones from the user's terminal to the server. Nowadays, technologies of federated learning have been widely applied to various recommendation systems. Literature [4] gave a federated recommendation system based on user embedding features and updates parameters with the time decay factor relying on user historical data. In [5], authors proposed a federated

Received September 30, 2022; revised October 31, 2022; accepted December 13, 2022.
Communicated by Xiaohong Jiang.
+ Corresponding author.
* The preliminary version of this work has been presented in NaNA conference.

collaborative filtering method for personalized recommendations, where vectors of user factor are updated by the user's local feedback data. Although systems based on federated learning reduce the probability that events of privacy leakage occur, there may exist unpredictable attacks when local model parameters of a terminal are transmitted, such as eavesdropping attacks. Thus, traditional approaches based on federated learning are not sufficient to ensure data security.

To improve privacy of data in recommendation systems, researchers have been studying the protection of model parameters, which are mainly achieved by data perturbation or cryptography. The federal recommendation algorithm (FedRec) was put forward by [6] for privacy protection, which uses mixed filling and user average algorithm to generate a pseudo-interaction set for data disturbance, and it improves privacy protection ability. In [7], authors studied the collaborative filtering recommendation system based on federated learning, which adds Laplacian noise to gradient parameters uploaded by end-users to reduce likelihood of privacy leakage. The framework of multi-party deep learning was designed by [8], where the method completed a strategy of dynamical allocation for privacy budget in each stage of federated learning.

In aforementioned works, most of them provide privacy for model parameters via losing the transmission rate or the accuracy of predicting users' preference. However, such indexes about quality of service are also important for an App, since they have heavy effects on users' experiences and the profit of App. Motivated by the above observations, in this paper, we try to propose a lightweight recommendation system based on federated learning for Apps, which aims at achieving accuracy of preference prediction and privacy of data. Particularly, we will design a recommendation scheme based on DEX file to improve the probability of successful preference prediction. It is notable that existing research mainly focuses on App recommendation based on user behavior [9, 10], while our work has an insight into recommendation method based on user's preference for functional class of App.

The contributions of this paper are three-fold:

(1) We propose a novel DEX file loading scheme based on federated learning, which measures users' preference for functional classes with their actual operation behaviors and provides priority loading of block-streaming service in wireless mobile networks. Our scheme will dynamically update the description of relationships among different functional classes to improve the accuracy of preference recommendations.

(2) Consider interactions between terminal users and edge servers that may suffer attacks. We apply additive homomorphic encryption and authorized blockchain to guarantee the privacy and secrecy of data. Furthermore, trusted storage and backtracking services are also realized for the scheme in this paper.

(3) We calculate internal behavior data of users about applications by embedded technology and conduct model training based on corresponding data. The effectiveness of our scheme is demonstrated by experiments. Results show that the proposed scheme can not only save storage space for end devices but also it is able to improve the efficiency of service transmission.

The rest of this paper is organized as follows. The second section describes the architecture of the DEX file loading scheme in this paper. The third section presents processes of the preference recommendation based on functional classes and approaches of guaranteeing data privacy in our proposed scheme. Experiments are shown in the fourth section and conclusions are given in the fifth section.

Throughout this paper, we use the following notations. Scalars and matrices are denoted by normal letter and bold letter, respectively. $\{x_1, x_2, \cdots, x_n\}$ represents a set and its elements. The notation $\mathbf{X}[i][j]$ describes the jth element of the ith line of matrix $\mathbf{X}$. We denote $Gragh.\mathbf{X}$ as the adjacency matrix for a graph that can be obtained by $\mathbf{X}$. Let $(\cdot)'$, $Enc(\cdot,\cdot)$, $Dec(\cdot,\cdot)$ denote the ciphertext, encryption and decryption, respectively.

2. SYSTEM ARCHITECTURE

2.1 Network Model

We consider a network model consisting of terminals, edge-end and cloud-end, where the terminal is composed of wireless mobile devices embedded in Android operating system, such as mobile phones and tablets, the edge-end comprises multiple edge servers, and the cloud-end is responsible for global data aggregation as well as large-scale data storage and computing. According to existing work, privacy leakage may be caused by reverse analysis of updating parameters in such networks [11]. In this paper, to protect data privacy of users' operation behavior, the end device is designed to constantly update model parameters and transmit model parameters instead of original behavior data to the edge server.

Consider the fact that transmission of block-streaming service in block-streaming execution platform(e) [BSEP(e)] mode may suffer attacks [12], which results in incorrect services provided by edge servers. In order to solve such problem, a novel loading scheme of block-streaming service based on federated learning for DEX file is proposed, which adopts existing block-streaming service and trusted verification method to realize trusted loading [13].

2.2 System Architecture

As shown in Fig. 1, we design a system consisting of the end device layer, edge server layer, blockchain storage layer and cloud server layer. The terminal layer forms the set U of Android operating system devices, and the edge layer is composed of several edge servers, which are usually distributed in the mobile phone base stations and have the function of interacting with the blockchain network. Based on this system, we build a federated learning-based scheme of block-streaming service for DEX file.

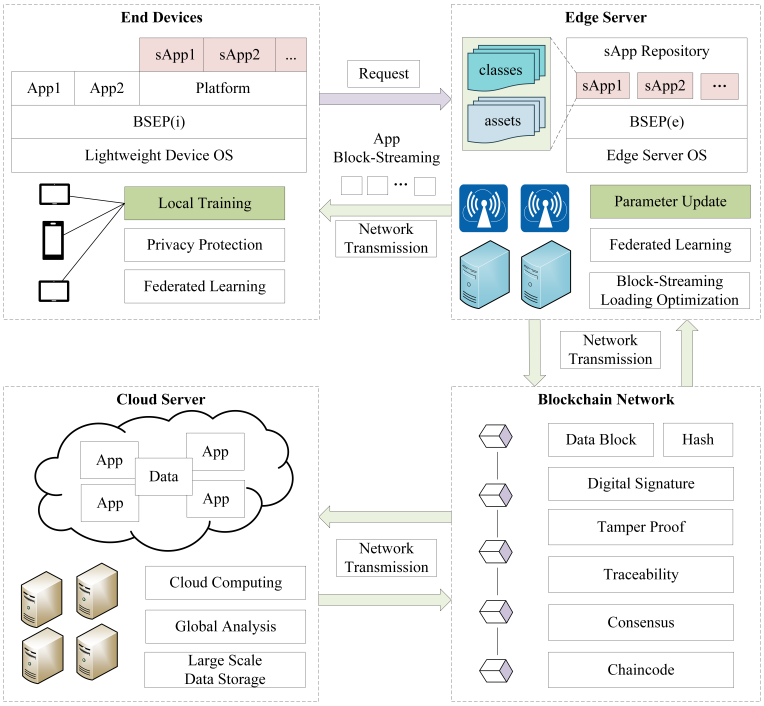

Fig. 1. System architecture.

First, the edge server that is nearest to the end device utilizes DEX file to generate a directed graph based on a specific application, which is denoted by the initial model $\mathbf{M}_0$. Then, the initial model $\mathbf{M}_0$ and digital signature D_s are sent to the end device.

Furthermore, the end device will apply $\mathbf{M}_0$ and sensitive data of users to model training, where local parameter $\mathbf{M}_t^u$ will be obtained and then transmitted to the edge server. After receiving $\mathbf{M}_t^u$, the edge server verifies the digital signature to ensure the reliability of received parameters and begins to train edge nodes. Particularly, once the edge server receives the weight parameters, it will also upload weight parameters in the blockchain network. Furthermore, the weight of the hot spot functional class will be also sent to the blockchain network within a designed time constraint, which can prevent hot spot function's weight from attacks and support the ability of backtracking information on such weight for each transaction.

Otherwise, we also design a cloud server layer in our system to provide data aggregation and information sharing for multiple edge servers. It will dynamically view the weights of hot spot functions on the blockchain network and conduct global model training. The weights of hot spot functions will be also transmitted to the edge server in a designed period, which will provide support to improve strategy of block-streaming service loading.

2.3 Scheme Design

A novel loading scheme of block-streaming service based on federated learning for DEX file (RS-FLBS) includes six steps, which are initialization, downloading global model, uploading local parameter information, model aggregation, uploading data to blockchain network and the block-streaming service loading application. The design of the scheme is shown in Fig. 2.

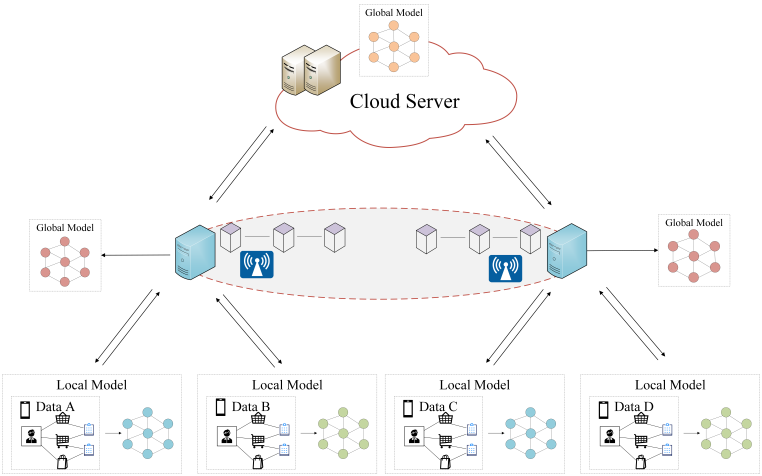

Fig. 2. Scheme design.

Initialization: The DEX file is first processed on the edge server, generating a directed graph of block-streaming service based on DEX file for a specific application program. The weight of the reference relationship between the directed graph nodes is randomized, so the initial model $\mathbf{M}_0$ is generated. Secondly, blockchain network is constructed, we use each edge server as a blockchain node, and peer-to-peer network is built by connecting each edge server. The edge server broadcasts the initial model $\mathbf{M}_0$ and digital signature D_s to neighboring nodes.

Download global model: The end-user devices download the initial model $\mathbf{M}_0$ from the edge server node that is nearest to themselves.

Upload local parameters: We will first calculate the duration of user access, click volume and user behavior type depending on the user's behavior. Then, the local operation data and the global model issued by the edge server are utilized to determine the local weights of parameters in the process of local training. Furthermore, obtained local model will be updated depending on the above results, while local weights of parameters will be uploaded to the edge server with technologies of homomorphic encryption.

Model aggregation: The edge server aggregates the local models uploaded by multiple users by federal average algorithm, which updates the aggregation parameters to the global model.

Data upload to blockchain network: Considering that users' preferences for past operations will remain relatively stable in the future, the edge server will send local model parameters and global model parameters to the blockchain network within a certain period of time, where the local model parameters are a matrix, which indicates the weights of the reference relationships between nodes in the directed graph (CRG) of block-streaming service loading based on DEX file for a specific application.

Let introduce CRG, $CRG\langle V, E\rangle$ which consists of a vertex set V and an edge set E. In this paper, $V = \{V_1, V_2, ..., V_n\}$, which represents the set of classes in the DEX file and $\langle V_i, V_j\rangle$ indicates that there is a direct reference relationship between V_i and V_j. Moreover, the reference relationship between classes is depicted by the set E, where $E = \{E_1, E_2, ..., E_n\}$ and $E_i = \{\langle V_i, V_j\rangle \,|V_i \in V \; and \; V_j \in V\}$. Adjacency matrix $Graph.\mathbf{A}$ is used to store the weighted directed graph (CRG).

Block-streaming service load application: According to the weight of reference relation in the directed graph (CRG) updated by the edge server gives priority to loading the functional class with a larger weight. We use BSEP(e) method to transmit block-streaming service through wireless networks and carry out the trusted loading and verification. Finally, loading is efficiently completed in the process of application service request, service calculation and service transmission.

3. PROCESS OF RS-FLBS SCHEME

3.1 Generating Directed Graph (CRG) of Block-Streaming Service Loading for DEX File

To obtain the loading weighted directed graph (CRG) of block-streaming service based on DEX file, an algorithm of class information extraction of DEX file is designed to get the class information as well as the reference relationship between classes by first analyzing DEX file, as Algorithm 1 presents [14].

Algorithm 1: The method of class information extraction (classExtraction)

input : classes.dex //DEX file
output: Information set of class

```
mapOff = DexHeader.mapOff;
DexMapList = DexHeader[mapOff];
for i ← 0 to n do
    DexMapItem=list[i];
    if DexMapItem.type.kDexTypeClassDefItem == DexMapItem.type then
        DexClassDef = DexMapItem [offset];
        className = DexMapItem[classIdx];
        preClassName = DexMapItem[superclassIdx];
    end
end
```

We can get the directed graph (CRG) of block-streaming service loading based on DEX file for Android application by the class information extraction algorithm and jar package obtained by reverse analysis of the application. CRG can well describe the class information of DEX file as well as the reference relationship between classes, and the weight means the value of calculation of the reference relationship between classes based on user behavior. The weighted directed graph is stored by adjacency matrix $Graph.\mathbf{A}$.

In the model, adjacency matrix $\mathbf{A}_{n\times n}$ for a weighted directed graph, which is applied to store the CRG, while $Graph.\mathbf{A}_{ij}$ in the matrix indicates the value of calculation of the direct reference relationship from V_i to V_j. The weight of the reference relationship

between classes is randomized in the initialization phase, in which the initial model $\mathbf{M}_0$ is generated. The time complexity of Algorithm 1 is analyzed below. Algorithm 1 describes the class information extraction process. It needs to traverse the *DexMapList*, so the time complexity of Algorithm 1 is $O(n)$.

3.2 Model Training

The edge server sends the initial model $\mathbf{M}_0$ to the terminal, and the terminal performs local training on the initialized model $\mathbf{M}_0$ according to the actual operation behavior data of the user. When the training is finished, the adjacent matrix model parameters calculated by the training are uploaded.

It is the key to loading block-streaming services by locally calculating the preference value of end-users for application access functions. Based on the preference value of user behavior data, a local model (UAFCM) for the user accesses functional class is constructed as

$$\text{UAFCM} = \{(c_0, c_0, P_{00}), (c_0, c_1, P_{01}), (c_0, c_2, P_{02}), \cdots, (c_i, c_j, P_{ij})\}, \tag{1}$$

where c_i, c_j and P_{ij} represent the functional class that current user is operating now, the functional class that current user may operate next and the preference value of the path that c_i accesses c_j, respectively.

The user behavior data model mainly calculates the weight of each user's operation behavior data and outputs weight relation matrix of the user's block-streaming service for a specific application. We consider that the application loads the functional classes of larger weights as much as possible according to the value of the dynamic update matrix $Graph.\mathbf{A}_{ij}$. The elements of block-streaming service loading model based on user behavior are mainly composed of θ, τ, υ and ϑ, which in turn represent the class name, the access time of the class, click volume of user events and the event handling class. Thus, block-streaming service loading model based on user behavior (UBSLM) can be given as

$$\text{UBSLM} = \{\theta, \tau, \upsilon, \vartheta\}, \tag{2}$$

where θ and ϑ form the path from c_i to c_j. Furthermore, υ marks click volume of user events, where the value is incremented by 1 once user event is triggered. The click volume is indicated by the autoincrement, υ_i and υ_j represent the click volume of i.class and j.class respectively. The preference value of the path from c_i to c_j is calculated by

$$P_{ij} = \frac{\sum_{k=1}^{K} GW_k \cdot \upsilon_j + \tau_j}{\upsilon_i}, \tag{3}$$

where $\tau_j = \tau_s - \tau_e$, GW_k is the weight of user operation behavior k, and τ_j represents the time when the user accesses the c_j class. Moreover, it is notable that user operation behaviors in Android applications generally include user-triggered events, user browsing and so on, in which the weight of comments is greater than the weight of other actions. The terminal x receives the global model $\mathbf{M}_t$ (initial value of t is zero), and then the local model will be updated by

$$\mathbf{M}_{t+1}^{x}[i][j] = Graph.\mathbf{A}_{ij} = \frac{P_{ij} + \mathbf{M}_t[i][j]}{2}, \tag{4}$$

where $\mathbf{M}_{t+1}^{x}$ is marked as

$$\mathbf{M}_{t+1}^{x} = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{bmatrix}_{n \times n}. \tag{5}$$

The model parameter $\mathbf{M}_{t+1}^{x}$ will be stored in the blockchain network after each iteration, which can support backtracking, and the local training model can be obtained.

The same model training process is repeated for U local end devices on the same LAN, and homomorphic encryption technology is used to upload the local model parameter $\mathbf{M}_{t+1}^{x}$. When the edge server receives the local model parameter $\mathbf{M}_{t+1}^{x}$, the global aggregation parameter is updated by [15]

$$\mathbf{M}_{t+1} = \frac{\sum_{u=1}^{U} \mathbf{M}_{t+1}^{x}[i][j]}{U}. \tag{6}$$

3.3 Parameter Encryption

Homomorphic encryption technology has the characteristics of supporting addition and multiplication under data encryption as well as low computational complexity. As the model training in this paper only uploads parameters involved in matrix $\mathbf{A}$, our paper will use additive homomorphic encryption technology to encrypt gradient parameters, and its encryption principle is presented by

$$\begin{aligned} &(A_1)' = Enc(pk, A_1)\ (A_2)' = Enc(pk, A_2), \\ &(C)' = (A_1)' + (A_2)', \\ &C = Dec(sk, (C)')\ and\ C = A_1 + A_2, \end{aligned} \tag{7}$$

where A_1 and A_2 represent adjacency matrix parameters uploaded by two terminals. Furthermore, pk, sk, $Enc()$ and $Dec()$ are public key, private key, encryption and decryption, respectively. The public key can encrypt the gradient parameters and send the encrypted and masked results to the edge server. After the edge server performs secure aggregation operation and sends that result to the end device. When the terminal receives the parameters, it decrypts them with the private key and updates the local model parameters with obtained information.

3.4 Algorithm

The data collection of user's actual operation behavior is completed by embedding technology. The embedding technology is that developers add processing code to the original complex logic in the application. With embedding technology, we can get the access time, click volume and type value of user behavior for the current class. Utilizing those coefficients, we can describe weights between different classes.

Algorithm 2: Federated learning algorithm (E-FedAvg)

Input : $\mathbf{M}_t$, Candidate U
Output: Global model $\mathbf{M}_{t+1}$ after decryption

initialize $\mathbf{M}_0$;
for $t \leftarrow 0$ **to** T **do**
  $\{U_x\}$ = Select the devices from the end device set U;
  Edge-server sends $\mathbf{M}_t$ to each device in $\{U_x\}$;
  for $u \in \{U_x\}$ ***in parallel*** **do**
    $(\mathbf{M}_{t+1}^{u})' = LocalUpdate(u, \mathbf{M}_t)$;
    Device u sends $(\mathbf{M}_{t+1}^{u})'$ back to server;
    $\mathbf{M}_{t+1}^{u} = Dec(pk, (\mathbf{M}_{t+1}^{u})')$;
  end
  Server aggregates the globle model $\mathbf{M}_{t+1}$ as $\mathbf{M}_{t+1} = \frac{\sum_{u=1}^{U} \mathbf{M}_{t+1}^{u}}{U}$;
end

We calculate operation data of user behavior, which the local model is trained by combining with the initialization model $\mathbf{M}_0$, and the training result parameters are uploaded to the edge server. Since privacy leakage may occur when uploaded parameters are transmitted among different devices or servers, homomorphic encryption will be adopted

for model training. First, each end device generates a public/private keypair, and $devicex$ and $devicey$ respectively train local model to get parameters $\mathbf{M}_{t+1}^{x}$ and $\mathbf{M}_{t+1}^{y}$. After encrypting $\mathbf{M}_{t+1}^{x}$ and $\mathbf{M}_{t+1}^{y}$ with the public key, $(\mathbf{M}_{t+1}^{x})'$ and $(\mathbf{M}_{t+1}^{y})'$ are sent to the edge server. When the edge server receives the parameters, it aggregates them with formula (6) after decrypting $(\mathbf{M}_{t+1}^{x})'$ and $(\mathbf{M}_{t+1}^{y})'$. The edge server encrypts the global model using the same process and sends it to the end device. After the end device decrypts the global parameters, the local model is trained and updated by combining with operation data of the local user. Then, the end device sends them to the edge server with technologies of homomorphic encryption. We repeat above process until the model converges, as Algorithm 2 presents. The time complexity of Algorithm 2 is analyzed below. It takes $O(n)$ for t rounds of global iteration and $O(\log_2 n)$ for parallel loop training of terminal equipment. Therefore, the time complexity of the federated learning algorithm based on homomorphic encryption is $O(n \log_2 n)$.

The $LocalUpdate(u, \mathbf{M}_t)$ in the sixth line of Algorithm 2 is further given by Algorithm 3. Since it needs $O(n)$ for e round local iteration, the time complexity of Algorithm 3 is $O(n)$.

Algorithm 3: Local model update (LocalUpdate)

input : $\mathbf{M}_t$, Candidate U
output: Encrypted local parameters $(\mathbf{M}_{t+1}^{u})'$

for *epoch* $e \in E$ **do**
  Training local weight parameter;
  $P_{ij} = \frac{\sum_{k=1}^{K} GW_k \cdot v_j + \tau_j}{v_i}$;
  $Graph.\mathbf{A}_{ij} = \frac{P_{ij} + \mathbf{M}_t[i][j]}{2}$;
  $\mathbf{M}_{t+1}^{u}[i][j] = Graph.\mathbf{A}_{ij}$;
  $\mathbf{M}_{t+1}^{u}$ **parameter homomorphic encryption**;
  $(\mathbf{M}_{t+1}^{u})' = Enc(pk, \mathbf{M}_{t+1}^{u})$
end
return $(\mathbf{M}_{t+1}^{u})'$;

Through the above steps, the global model parameters can be obtained, which is the weight of the reference relationship in the directed graph based on block-streaming service loading for DEX file. According to the weight, the application can be loaded through the block-streaming service, and the service with the larger weight can be loaded preferentially as much as possible, where loading of the application can be efficiently completed.

4. EXPERIMENT

The experimental environment used in RS-FLBS scheme is configured as follows: The terminal operating system: android 12; Storage space: 10GB; RAM: 64KB. The edge server device operating system: CentOS Linux Release 7.2.1511; Storage space: 40GB; RAM: 4GB.

4.1 Initialization Stage

To validate our work, we first developed a lightweight application named House Project, which contains three functional modules, namely terminal monitoring module, user management module and system setting module, and the House Project is installed on Android devices. Then, based on the application, classes.dex is converted into classes-dex2jar.jar with the help of dex2jar tool and the classes-dex2jar.jar file is dragged into jd-gui tool, following which class file can be viewed. By analyzing the .class files, we can get the dependency relationship between different classes in House Project. The class

names are shown in Table 1, where Not. is the abbreviation of Notion. Fig. 3 shows directed graph of block-streaming service that loading for DEX file.

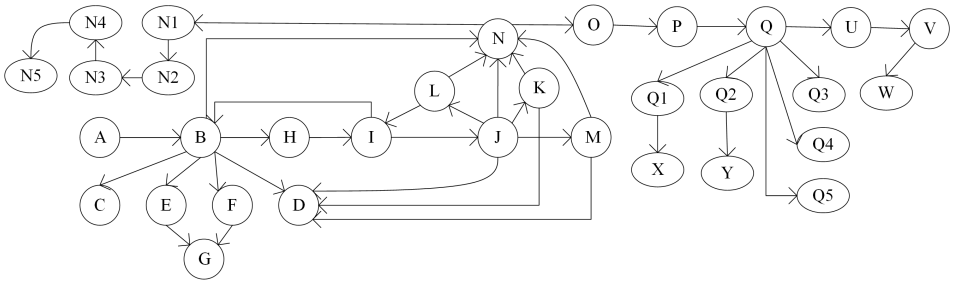

Fig. 3. The directed graph of the House Project based on block-streaming service loading of DEX file.

Table 1. House project class names.

Not.	Class Name	Not.	Class Name	Not.	Class Name
A	Launcher	B	LoginActivity	Y	ComponentCallbacks
C	Person	L	OtherActivity	Q2	ComponentCallbacks2
D	Intent	M	SettingActivity	Q3	onCreateContextMenuListener
E	Button	N	AppCompatActivity	Q4	android.view.keyEvent
F	EditText	J	MainActivity	Q5	android.view.windows
G	TextView	K	ControlActivity	U	ContextThemeWrapper
H	view	N3	LayoutInflater	N5	AppCompatTextView
Q	Activity	V	contextWrapper	N1	AppCompatDelegateImpl
Q1	Factory2	O	FragmentActivity	N2	LayoutInflaterCompat
X	Factory	W	context	N4	AppCompatViewInflater
I	Toast	P	ComponentActivity		

Table 2. Initialize class dependency weights.

	Values		Values		Values		Values		Values
$a_{2\ 3}$	9	$a_{2\ 4}$	66	$a_{2\ 5}$	17	$a_{2\ 6}$	5	$a_{2\ 8}$	78
$a_{2\ 17}$	28	$a_{9\ 16}$	89	$a_{9\ 2}$	30	$a_{16\ 13}$	63	$a_{16\ 4}$	3
$a_{16\ 12}$	8	$a_{16\ 14}$	74	$a_{12\ 4}$	84	$a_{14\ 4}$	72	$a_{13\ 9}$	56
$a_{12\ 17}$	91	$a_{14\ 17}$	77	$a_{13\ 17}$	85				

The directed graph is stored in the adjacency matrix $Graph.\mathbf{A}$, and the dependency weights between initialization classes are shown in Table 2. The initialization rule is a one-to-one dependency with a weight of ∞, and the rest of the dependencies are randomly set to integers in the interval $[1, 100]$.

4.2 Model Training

In order to verify the effectiveness of our scheme, we apply data based on House Project to experiments. With technologies of SDK embedding, weights of user-defined classes in House Project are calculated and updated, which is further utilized to be the test data in the following experiments. Consider different user behavior involved in this study, namely the user-triggered behavior, the user browsing, the user comment, the user favorites and the user collection. The respective weight of above user behavior, denoted by GW_k in Section 3, is set to be 25, 15, 40, 5 and 15. Otherwise, with an analysis of the Android operating system source code, our approaches in this work are also available to extract system classes and compute their weights.

Utilizing formulas (3) and (4), we update values of $a_{16\ 12}$, $a_{9\ 16}$, $a_{16\ 14}$ and $a_{16\ 13}$ to be 17, 84 ,41 and 37 for the local model, respectively, which is based on the test data and weights of user operation behavior. With similar steps, operation data from 10 Android users are collected in local area network and the global model parameters can be obtained through the E-FedAvg algorithm. Adopting formula (6), values of $a_{16\ 12}$, $a_{9\ 16}$, $a_{16\ 14}$, $a_{16\ 13}$ are updated to be 15, 114 ,41 and 37, respectively.

Then, we will use the global model parameters obtained above to dynamically load the block-streaming service, giving priority to the services with larger weights. There are 10 new android users. When users utilize the application for House Project, they will load the service by BSEP(e) method according to the global model parameters. Here, local model training is carried out according to the global model and local sensitive data. Theoretically, the service predicted by 10 users is $MainActivity$. However, due to experiments in this paper, services that users prefer to load are $MainActivity$, $MainActivity$, $MainActivity$, $MainActivity$, $LoginActivity$, $LoginActivity$, $MainActivity$, $MainActivity$, $MainActivity$, $MainActivity$, respectively. It means the likelihood achieves eighty percent, where we are successful in predicting the preference of block-streaming service.

4.3 Cost Analysis

It is known that new versions of Android operating system will implement new system function modules into libraries, which are part of applications. Some classes provided by the operating system may be not packaged into DEX file. To verify the effectiveness of our proposed scheme, time and space required in the whole application running process are analyzed. In order to avoid the influence of network environment, we complete experiment 10 times and show their average results in this section.

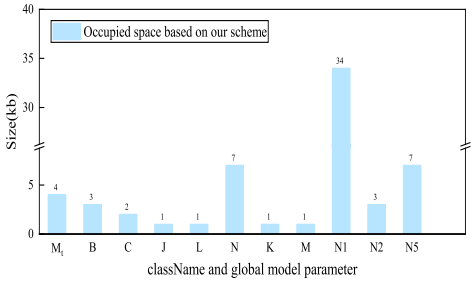

Fig. 4. Space overhead of our scheme.

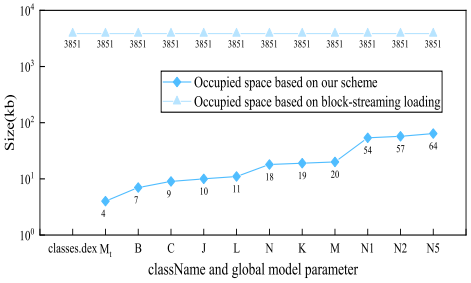

Fig. 5. Cumulative space overhead of our scheme.

We investigate terminal equipment needed for storage space in a novel loading scheme based on federated learning for DEX file as shown in Fig. 4. The bar chart represents the global model parameter $\mathbf{M}_t$ and the amount of space occupied by the application of House Project to complete the on-demand loading of each .class file, where the abscissa is denoted by $Not.$, and the specific class name is shown in Table 1. We consider terminal equipment needed for accumulated storage space as shown in Fig. 5. The bar colored by dark blue shows the global model parameter $\mathbf{M}_t$ and the rest uses $Not.$ to represent the cumulative amount of space occupied by the House application to complete the on-demand loading of each .class file. The traditional block-streaming service scheme requires 3851KB to load the classes.dex file, which colors light blue. It is notable that there is no data with respect to the classes.dex under our proposed scheme, since our method loads classes in DEX file instead of DEX file itself. Obviously, we can see from Fig. 5 that the block-streaming loading method for DEX file requires about 60 times storage space of our proposed scheme in this paper, which illustrates that the RS-FLBS scheme has a good performance in saving storage space for users.

The average time cost of local model training is shown in the bar chart in Fig. 6. We consider respective time cost of downloading global model parameters, decrypting global model parameters, storing global model parameters into database, training local model by combining local data with global model parameters, and encrypting updated local model parameters. The line chart in the Fig. 6 represents the accumulated time cost of each stage. The bar chart in Fig. 7 presents the average time cost of global model training, which include time costs of decrypting local model parameters, calculating global model parameters, storing global model parameters in notepad file and encrypting global model parameters. The accumulated time cost of each stage is shown in the line chart in Fig. 7.

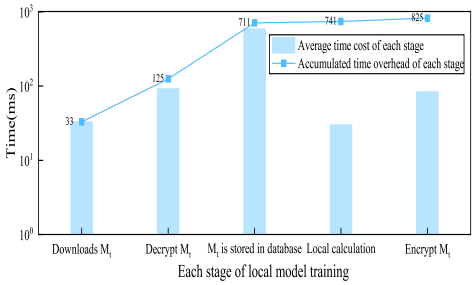

Fig. 6. Local model training time overhead.

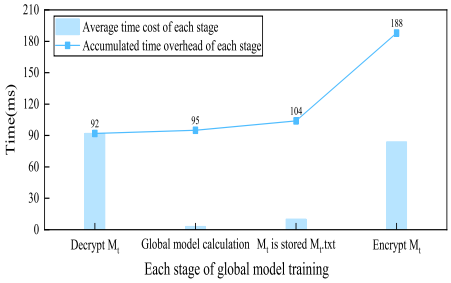

Fig. 7. Global model training time overhead.

We study the average time overhead required in the DEX file loading scheme from a federated learning perspective. In Fig. 8, $Local$ describes the total time of local model training, $Global$ represents the total time of global model training and the rest uses $Not.$ to represent time costs required by application on-demand loading of .class files in the House Project. The total time cost required by our method is shown in Fig. 9. The blue line with five-pointed star indicates that the total time required for the whole processes is 1444 ms, which includes loading .class file, local model training and global model training. Furthermore, time cost of traditional block-streaming service scheme to load classes.dex file is about 27363 ms, which is denoted by blue square. It should highlight that our method only loads classes in DEX file but not DEX file so that Fig. 9 does not show the data about classes.dex under our proposed scheme. It can be seen from Fig. 9 that compared with the block-streaming loading method for dex file, efficiency will be greatly increased if the RS-FLBS scheme is applied, where the cost of loading time for block-streaming method of DEX file is about 18 times higher than our proposed scheme.

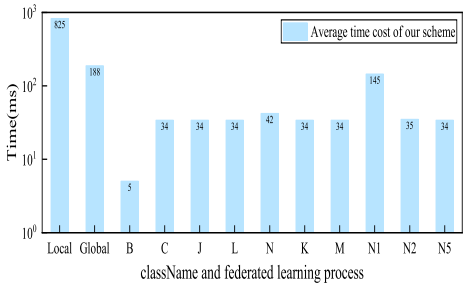

Fig. 8. Time overhead of our scheme.

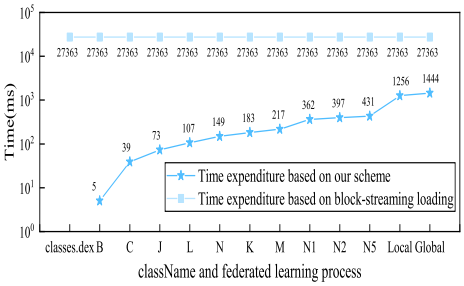

Fig. 9. Cumulative time overhead of our scheme.

Obviously, with our proposed scheme, both time and space costs are greatly reduced, which illustrates that our scheme is effective to recommend the user's interested preference functional classes according to the user's actual operation behavior.

5. CONCLUSIONS

This paper focuses on the problem of data overload in wireless mobile networks, which may influence the efficiency of communication among terminals and servers. Combining technologies of federated learning and blockchain, we propose a novel loading scheme based on a recommendation system for DEX file to guarantee the privacy of users' operation data involved in training. Moreover, in order to provide secrecy for data transmission of model training parameters, the technology of homomorphic encryption is further introduced to the proposed scheme. In our work, the weights based on user-defined classes are measured by their behavior and then utilized to find the functional class that the user prefers, which will affect what block-streaming services should be loaded. The validity of our model involved in this paper is verified by an application called House Project on Android devices. The results show that the proposed scheme has a better per-

formance than traditional block-streaming loading schemes in both storage cost and time cost. It illustrates that our work seems to be helpful for solving the data overload problem of wireless mobile networks.

REFERENCES

1. A. Balapour, H. R. Nikkhah, and R. Sabherwal, "Mobile application security: Role of perceived privacy as the predictor of security perceptions," *International Journal of Information Management*, Vol. 52, 2020, pp. 102 063-102 075.
2. X. Ye, M. Chen, and R. Ali, "Personalized recommendation for mobile internet wealth management based on user behavior data analysis," *Scientific Programming*, Vol. 2021, 2021, pp. 9 326 932-9 326 939.
3. L. Melis, C. Song, E. De Cristofaro, and V. Shmatikov, "Exploiting unintended feature leakage in collaborative learning," in *Proceedings of IEEE Symposium on Security and Privacy*, 2019, pp. 691-706.
4. Z. Jie, S. Chen, J. Lai, M. Arif, and Z. He, "Personalized federated recommendation system with historical parameter clustering," *Journal of Ambient Intelligence and Humanized Computing*, Vol. 2022, 2022, pp. 1-8.
5. M. Ammad-ud-din, E. Ivannikova, S. A. Khan, W. Oyomno, Q. Fu, K. E. Tan, and A. Flanagan, "Federated collaborative filtering for privacy-preserving personalized recommendation system," *arXiv Preprint*, 2019, arXiv:1901.09888.
6. G. Lin, F. Liang, W. Pan, and Z. Ming, "FedRec: Federated recommendation with explicit feedback," *IEEE Intelligent Systems*, Vol. 36, 2021, pp. 21-30.
7. Y. Wang, Y. Tian, X. Yin, and X. Hei, "A trusted recommendation scheme for privacy protection based on federated learning," *CCF Transactions on Networking*, Vol. 3, 2020, pp. 218-228.
8. M. Gong, J. Feng, and Y. Xie, "Privacy-enhanced multi-party deep learning," *Neural Networks*, Vol. 121, 2020, pp. 484-496.
9. S. Zhao, Z. Luo, Z. Jiang, H. Wang, F. Xu, S. Li, J. Yin, and G. Pan, "Appusage2vec: Modeling smartphone app usage for prediction," in *Proceedings of IEEE 35th International Conference on Data Engineering*, 2019, pp. 1322-1333.
10. Y. Z. K. Wang, B. Wang, and J. Wang, "Research on app usage prediction algorithm based on smartphone users' behavior habits," *Computer Applications and Software*, Vol. 36, 2019, pp. 82-86.
11. Z. Wang, M. Song, Z. Zhang, Y. Song, Q. Wang, and H. Qi, "Beyond inferring class representatives: User-level privacy leakage from federated learning," in *Proceedings of IEEE INFOCOM – IEEE Conference on Computer Communications*, 2019, pp. 2512-2520.
12. X. Peng, J. Ren, L. She, D. Zhang, J. Li, and Y. Zhang, "Boat: A block-streaming app execution scheme for lightweight iot devices," *IEEE Internet of Things Journal*, Vol. 5, 2018, pp. 1816-1829.
13. Y. Wang, Y. Tian, X. Hei, L. Zhu, and W. Ji, "A novel iov block-streaming service awareness and trusted verification scheme in 6G," *IEEE Transactions on Vehicular Technology*, Vol. 70, 2021, pp. 5197-5210.
14. Y. Wang, Y. Feng, Y. Du, X. Hei, Y. Tian, and X. Cui, "Block-streaming service loading optimization of android dalvik executable file for cloud-end collaboration," in *Proceedings of International Conference on Networking and Network Applications*, 2022, pp. 1-6.
15. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. Y. Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Artificial Intelligence and Statistics*, 2017, pp. 1273-1282.

YiChuan Wang received the Ph.D. degree in Computer System Architecture from Xidian University, Xi'an, China, in 2014. He is currently an Associate Professor with the Shaanxi Key Laboratory of Network Computing and Security Technology, Xi'an University of Technology, Xi'an, China. His research interests include networks security and system vulnerability analysis. He is a Member of ACM and CCF.

YanHua Feng is studying for a master's degree in Xi'an University of Technology. Her main research direction is block-streaming service loading for Android.

YeQiu Xiao received the B. Eng. in Computer Science and Technology, M. Eng. degree in Computer Software and Theory, and Ph.D. degree in Computer System Architecture from Xidian University, Xi'an, China, in 2013, 2016 and 2021, respectively. She is currently a Lecturer in Xi'an University of Technology, Xi'an, China. Her current work concerns the physical layer security of wireless communications and satellite communication systems.

XiaoXue Liu received her MS degree from Shaanxi Normal University and Ph.D. degree from Xidian University in 2021, Xi'an, China. She now is a Lecturer with the School of Computer Science and Engineering, Xi'an University of Technology, Xi'an, China. Her current research interests include cryptography and information security.

XinHong Hei received his BS and MS degrees in Computer Science and Technology from Xi'an University of Technology, Xi'an, China, in 1998 and 2003, respectively, and his Ph.D. degree from Nihon University, Tokyo, Japan, in 2008. He is currently a Professor with the Faculty of Computer Science and Engineering, Xi'an University of Technology, Xi'an, China. His current research interests include intelligent systems, safety-critical system, and train control system.