Secure Federated Matrix Factorization

Di Chai*, Leye Wang*, Kai Chen, and Qiang Yang, *Fellow, IEEE*

**equal contribution, ranked alphabetically*

Abstract—To protect user privacy and meet law regulations, federated (machine) learning is obtaining vast interests in recent years. The key principle of federated learning is training a machine learning model without needing to know each user's personal raw private data. In this paper, we propose a secure matrix factorization framework under the federated learning setting, called *FedMF*. First, we design a user-level distributed matrix factorization framework where the model can be learned when each user only uploads the gradient information (instead of the raw preference data) to the server. While gradient information seems secure, we prove that it could still leak users' raw data. To this end, we enhance the distributed matrix factorization framework with homomorphic encryption. We implement the prototype of FedMF and test it with a real movie rating dataset. Results verify the feasibility of FedMF. We also discuss the challenges for applying FedMF in practice for future research.

Index Terms—IEEE Intelligent system, Security and Privacy Protection, Distributed system

1 Introduction

With the prevalence of government regulations and laws on privacy protection in the big data era (e.g., General Data Protection Regulation[1]), privacy-preserving machine learning has obtained rapidly growing interests in both academia and industry. Among various techniques to achieve privacy-preserving machine learning, federated (machine) learning (FL) recently receives high attention. The original idea of FL was proposed by Google [2], which targets at learning a centered model based on the personal information distributed at each user's mobile phone. More importantly, during model training, no user's raw personal information is transferred to the central server, thus ensuring privacy protection. Also, the learned privacy-preserving model can be proved to hold almost similar predictive power compared to the traditional model learned on users' raw data. This highlights the practicality of FL as little predictive accuracy is sacrificed, especially compared to other accuracy-lossy privacy-preserving mechanisms such as differential privacy.

Starting from the original Google paper, many researchers have been devoted to this promising and critical area. Recently, a nice survey paper on FL has been published [10]. While many research works have been done on FL, a popular machine learning technique, *matrix factorization* (MF) [5], is still under-investigated in FL. Since MF is one of the prominent techniques widely employed in various applications such as item recommendation [5] and environment monitoring [8], we highly believe that studying MF under FL is urgently required. This work is one of the pioneering research efforts toward this direction.

Taking recommendation systems as an example, two types of users' private information are leaked in traditional MF [5]: (i) *users' raw preference data*, and (ii) *users' learned latent feature vectors*. As revealed by previous studies, either raw preference data or latent features can leak users' sensitive attributes, e.g., age, relationship status, political views, and sex orientations [9]. This highlights the importance of protecting users' private information during MF. Prior studies have studied privacy-preserving MF in two main types:

(1) **Obfuscation-based methods** obfuscate users' preference raw data before releasing it to the central server so as to ensure a certain level of privacy protection (e.g., differential privacy) [3]. The pitfall is that obfuscation inevitably leads to the loss of predictive power of the learned latent feature vectors. Hence, these methods usually need to make a trade-off between privacy protection and model performance.

(2) **Encryption-based methods** use advanced encryption schemes such as homomorphic encryption for implementing privacy-preserving MF [4]. While they usually do not need to sacrifice predictive power for privacy protection, they commonly require a third-party crypto-service provider. This makes the system implementation complicated as such a provider is not easy to find in practice.[3] Moreover, if the crypto-service provider colludes with the recommendation server, then no user privacy protection can be preserved [4], [6].

This research proposes a novel FL-based MF framework, called **FedMF** (Federated Matrix Factorization). FedMF employs distributed machine learning and homomorphic encryption schemes. In brief, FedMF lets each user compute the gradients of his/her own rating information locally and then upload the gradients (instead of the raw data) to the server for training. To further enhance security, each user can encrypt the gradients with homomorphic encryption. With FedMF, two shortcomings of the traditional obfuscation- or encryption-based methods can be addressed: (i) no predictive accuracy is lost as we do not obfuscate data; (ii) no third-party crypto-service provider is required as each user's device can handle the secure gradient computing task.

1. https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX:32016R0679R(02)
2. https://ai.googleblog.com/2017/04/federated-learning-collaborative.html
3. Some studies suggest that governments can perform this role, but this is still not the case in reality now [4].

In summary, we have the following contributions:

(1) To the best of our knowledge, we design the first FL-based secure MF framework, called *FedMF*, which can overcome the shortcomings of traditional obfuscation- or encryption-based mechanisms as aforementioned.

(2) To implement FedMF, first, we design a user-level distributed matrix factorization framework where the model can be learned when each user only uploads the gradient information (instead of the raw preference data) to the server. Though gradient information seems secure, we prove that it could still leak users' raw data to some extent[4]. Then, we enhance the distributed matrix factorization framework with homomorphic encryption to increase security.

(3) We implement a prototype of FedMF (the code will be published in https://github.com/Di-Chai/FedMF). We test the prototype on a real movie rating dataset. Results verify the feasibility of FedMF.

It is worth noting that, similar to FedMF, [1] tried to develop a federated collaborative filtering system. However, [1] directly let users upload their gradient information to the server. As we will prove in this paper, gradients can still reveal users' original preference data. Hence, a more secure system like FedMF is required to rigorously protect users' privacy.

2 Preliminaries

In this section, we briefly introduce two techniques closely related to our research, *horizontal federated learning* and *additively homomorphic encryption*.

2.1 Horizontal Federated Learning

Federated learning is a method that enables a group of data owners to train a machine learning model on their joint data and nobody can learn the data of the participants. Federated learning can be categorized based on the distribution characteristics of the data [10]. One category of federated learning is *horizontal federated learning*. Horizontal federated learning is introduced in the scenarios when the data from different contributors share the same feature space but vary in samples. In our secure matrix factorization recommendation system, the rating information distributed on each user's device has exactly the same feature space, while different users are exactly different samples. So our matrix factorization in federated learning can be categorized as horizontal federated learning. Following the typical assumption of horizontal federated learning [10], we assume **all the users are honest and the system is designed to be secure against an honest-but-curious server**[5].

2.2 Additively Homomorphic Encryption

Homomorphic encryption (HE) is often used in federated learning to protect user's privacy by providing encrypted parameter exchange. HE is a special kind of encryption scheme that allows any third party to operate on the encrypted data without decrypting it in advance. An encryption scheme is called homomorphic over an operation '$\star$' if the following equation holds:

$$E(m_1) \star E(m_2) = E(m_1 \star m_2),\ \forall m_1, m_2 \in M \qquad (1)$$

where E is an encryption algorithm and M is the set of all possible messages.

Addictively homomorphic encryption is homomorphic over addition. Typically, it consists of the following functions:

- $KeyGen \rightarrow (pk, sk)$: key generation process, where pk is the public key and sk is the secret key.
- $Enc(m, pk) \rightarrow c$: encryption process, where m is the message to be encrypted and c is the ciphertext.
- $Dec(c, sk) \rightarrow m$: decryption process
- $Add(c_1, c_2) \rightarrow c_a(i.e.\ Enc(c_1 + c_2))$: add operation on ciphertext, c_a is the ciphertext of plaintexts' addition.
- $DecAdd(c_a, sk) \rightarrow m_a$: decrypt c_a, getting the addition of plaintexts.

3 User-level Distributed Matrix Factorization

We firstly introduce the matrix factorization optimization method used in our paper, stochastic gradient descent [5]. Based on it, we design a user-level distributed matrix factorization framework.

3.1 Stochastic Gradient Descent

Suppose we have n users, m items and each user rated a subset of m items. For $[n] := \{1, 2..., n\}$ as the set of users and $[m] := \{1, 2, ..., m\}$ as the set of items, we denote $\mathcal{M} \in [n] \times [m]$ as user-item rating pairs which a rating has been generated, $M = |\mathcal{M}|$ as the total number of ratings and $r_{i,j}$ represents the rating generated by user i for item j.

Given the rating information $r_{ij} : (i, j) \in \mathcal{M}$, the recommendation systems are expected to predict the rating values of all the items for all the users. Matrix factorization formulates this problem as fitting a bi-linear model on the existing ratings. In particular, user profile matrix $U \in \mathbf{R}^{n \times d}$ and item profile matrix $V \in \mathbf{R}^{m \times d}$ are computed, where d is the dimension of the profile vector, the resulting profile matrices are used to predict user i's rating on item j, which is $\langle u_i, v_j \rangle$. The computing process of U and V can be done by solving the following regularized least squares minimization:

$$\min_{U,V} \frac{1}{M}(r_{i,j} - \langle u_i, v_j \rangle)^2 + \lambda ||U||_2^2 + \mu ||V||_2^2 \qquad (2)$$

where λ and μ are small positive values to rescale the penalizer. Stochastic gradient descent iteratively updates U and V with the following equations [5]:

$$u_i^t = u_i^{t-1} - \gamma \bigtriangledown_{u_i} F(U^{t-1}, V^{t-1}) \qquad (3)$$

$$v_i^t = v_i^{t-1} - \gamma \bigtriangledown_{v_i} F(U^{t-1}, V^{t-1}) \qquad (4)$$

where

$$\bigtriangledown_{u_i} F(U, V) = -2 \sum_{j:(i,j)} v_j (r_{ij} - \langle u_i, v_j \rangle) + 2\lambda u_i \qquad (5)$$

4. Similar work that proves gradients leak information also attracts many other researchers' interests, such as [2] proves gradients leak information in image deep learning domain. Here we prove the privacy leakage from gradients in matrix factorization process.

5. An honest-but-curious server means that the server is a legitimate participant in the system who will not deviate from the defined protocol but will attempt to learn all possible information from legitimately received messages

Algorithm 1 User-level Distributed Matrix Factorization

```
Init: Server initializes item profile matrix V
Init: User initializes user profile matrix U
Output: Converged U and V
  Server keeps latest item-profile for all users' download
  User local update:
    Download $V$ from server, peform local updates:
    $u_i^t = u_i^{t-1} - \gamma \bigtriangledown_{u_i} F(U^{t-1}, V^{t-1})$
    $Gradient_i = \gamma \bigtriangledown_{v_i} F(U^{t-1}, V^{t-1})$
    Send $Gradient_i$ to server
  Server update:
    Receive $Gradient_i$ from user-i
    Perform update : $v_i^t = v_i^{t-1} - Gradient_i$
```

$$\bigtriangledown_{v_j} F(U,V) = -2 \sum_{i:(i,j)} u_i(r_{ij} - \langle u_i, v_j\rangle) + 2\lambda v_j \quad (6)$$

The number of iterations relies on the stopping criteria. A typical criteria is to set a small threshold ε, such that the training stops when the gradient $\bigtriangledown_{u_i} F$ and $\bigtriangledown_{v_j} F$ (or one of them) are smaller than ε.

3.2 Distributed Matrix Factorization

In the distributed matrix factorization scenario, users hold their rating information locally and the model is trained on their joint data. To achieve this goal, we leverage a distributed matrix factorization method, which decomposes the iterative updating process into two parts that are performed on the user side and the server side, respectively. In particular, equation (3) is executed on user i's device, namely *user update*, and equation (4) is performed on the server, called *server update*. This decomposition prevents the server from directly knowing users' raw preference data or learned profiles.

Algorithm 1 shows our user-level distributed matrix factorization method. The server keeps providing the latest item profile matrix V for all the users to download. Having the latest downloaded V and his/her own rating information, each user i performs local updates and computes $Gradient_i$, which will be sent back to the server to update item profiles.

4 Gradients Leak Information

With the enacting of many privacy-preserving regularizations (e.g., GDPR), dealing with personal data is sensitive and critical. Because GDPR has strict restrictions on collecting and using personal data, e.g., companies cannot give users' data to other companies without users' explicit permission. Currently, few companies can meet the requirements, and the violation may face a hefty fine. According to the definition in GDPR, users' rating data belongs to personal data [6] because we can link the ratings to natural person even without identity information (e.g., using linkage attack), thus it needs to be carefully treated. In FL, users keep the rating data locally and only upload certain gradients. Thus, GDPR can be satisfied as long as the uploaded information leaks no private data. Algorithm 1 shows a framework that allows the server to build a matrix factorization recommendation system on a distributed dataset, i.e., users keep rating information locally. In this framework, users iteratively send $Gradient$ information to the server in plain text. Next, we are going to prove that such a distributed matrix factorization system cannot protect users' rating information against the server, i.e., the server can deduce users' rating data using the $Gradient$. Thus, a new solution (i.e., homomorphic encryption) is required to avoid private data leakage from gradients and guarantee that the federated matrix factorization can satisfy the GDPR regulation.

For user-vector u_i, suppose the user rated item-set is M_i. We will have the following equation at time t

$$u_i^t(r_{ij} - \langle u_i^t, v_j^t\rangle) = G_j^t,\ j \in M_i \quad (7)$$

where G is the gradient to be uploaded from the user i to the server for updating item-profiles. Similarly, at time $t+1$,

$$u_i^{t+1}(r_{ij} - \langle u_i^{t+1}, v_j^{t+1}\rangle) = G_j^{t+1},\ j \in M_i \quad (8)$$

From equation (3) and (5), the correlation between U_t and U_{t+1} is :

$$(1-2\lambda\gamma)u_i^t - u_i^{t+1} = -2\gamma \sum_{j\in M_i} v_j^t * (r_{ij} - < u_i^t, v_j^t >) \quad (9)$$

Take a close look at the element-wise calculation of equation (7)–(9), where we denote the latent user and item vectors (u_i and v_j) are D dimension:

$$\begin{cases} u_{i1}^t(r_{ij} - \sum_{m=1}^D u_{im}^t v_{jm}^t) = G_{j1}^t \\ \vdots \\ u_{ik}^t(r_{ij} - \sum_{m=1}^D u_{im}^t v_{jm}^t) = G_{jk}^t \\ \vdots \\ u_{iD}^t(r_{ij} - \sum_{m=1}^D u_{im}^t v_{jm}^t) = G_{jD}^t \end{cases} \quad (10)$$

$$\begin{cases} u_{i1}^{t+1}(r_{ij} - \sum_{m=1}^D u_{im}^{t+1} v_{jm}^{t+1}) = G_{j1}^{t+1} \\ \vdots \\ u_{ik}^{t+1}(r_{ij} - \sum_{m=1}^D u_{im}^{t+1} v_{jm}^{t+1}) = G_{jk}^{t+1} \\ \vdots \\ u_{iD}^{t+1}(r_{ij} - \sum_{m=1}^D u_{im}^{t+1} v_{jm}^{t+1}) = G_{jD}^{t+1} \end{cases} \quad (11)$$

$$\begin{cases} (1-2\lambda\gamma)u_{i1}^t - u_{i1}^{t+1} = -2\gamma \sum_{n=1}^N v_{n1}^t(r_{in} - \sum_{m=1}^D u_{im}^t v_{nm}^t) \\ \vdots \\ (1-2\lambda\gamma)u_{ik}^t - u_{ik}^{t+1} = -2\gamma \sum_{n=1}^N v_{nk}^t(r_{in} - \sum_{m=1}^D u_{im}^t v_{nm}^t) \\ \vdots \\ (1-2\lambda\gamma)u_{iD}^t - u_{iD}^{t+1} = -2\gamma \sum_{n=1}^N v_{nD}^t(r_{in} - \sum_{m=1}^D u_{im}^t v_{nm}^t) \end{cases} \quad (12)$$

6. Personal data means any information relating to an identified or identifiable natural person; an identifiable natural person is one who can be identified, directly or indirectly (more information can be found at https://gdpr.eu/eu-gdpr-personal-data).

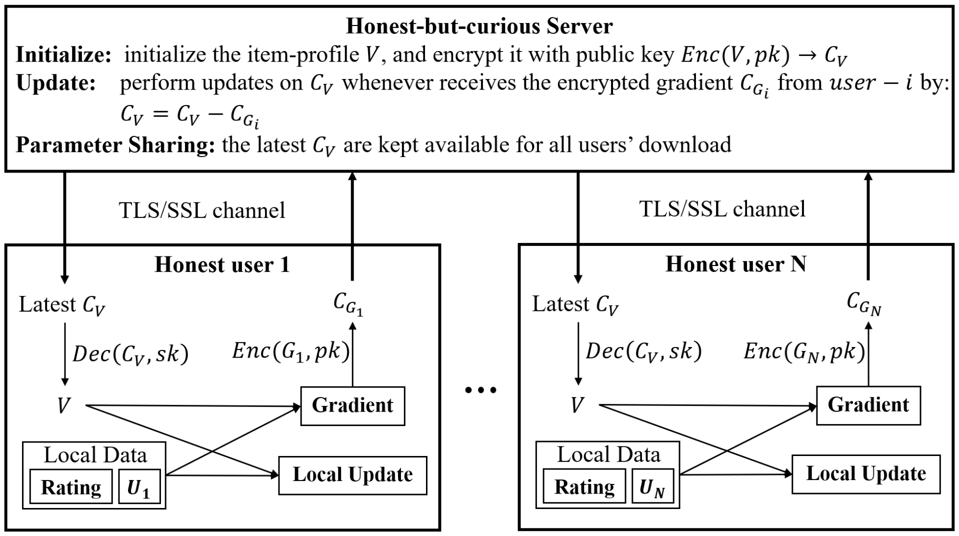

Honest-but-curious Server
Initialize: initialize the item-profile V, and encrypt it with public key $Enc(V, pk) \to C_V$
Update: perform updates on C_V whenever receives the encrypted gradient C_{G_i} from $user - i$ by: $C_V = C_V - C_{G_i}$
Parameter Sharing: the latest C_V are kept available for all users' download
TLS/SSL channel
TLS/SSL channel
Honest user 1
Latest C_V
C_{G_1}
$Dec(C_V, sk)$
$Enc(G_1, pk)$
V
Gradient
Local Data
Rating
U_1
Local Update
…
Honest user N
Latest C_V
C_{G_N}
$Dec(C_V, sk)$
$Enc(G_N, pk)$
V
Gradient
Local Data
Rating
U_N
Local Update

Fig. 1: Overview of FedMF

Now we turn to analyze the k-th entry of u_i, u_{ik}. From equation (10), we have :

$$\frac{u_{ik}^t}{u_{i(k+1)}^t} = \frac{G_{ik}^t}{G_{i(k+1)}^t} \tag{13}$$

$$r_{ij} - \sum_{m=1}^{D} u_{im}^t v_{jm}^t = \frac{G_{jk}^t}{u_{ik}^t} \tag{14}$$

Plug equation (13) into (12), we will have

$$(1-2\lambda\gamma)u_{ik}^t - u_{ik}^{t+1} = -2\gamma \frac{1}{u_{ik}^t} \sum_{n=1}^{N} v_{nk}^t G_{nk}^t \tag{15}$$

Thus we can represent u_{ik}^{t+1} using u_{ik}^t as:

$$u_{ik}^{t+1} = (1-2\lambda\gamma)u_{ik}^t + 2\gamma \frac{1}{u_{ik}^t} \sum_{n=1}^{N} v_{nk}^t G_{nk}^t \tag{16}$$

From equation (10) and (11) we have:

$$\frac{G_{jk}^t}{u_{ik}^t} + \sum_{m=1}^{D} u_{im}^t v_{jm}^t = \frac{G_{jk}^{t+1}}{u_{ik}^{t+1}} + \sum_{m=1}^{D} u_{im}^{t+1} v_{jm}^{t+1} \tag{17}$$

Plug equation (16) into (17):

$$\frac{G_{jk}^t}{u_{ik}^t} - \frac{G_{jk}^{t+1}}{(1-2\lambda\gamma)u_{ik}^t + 2\gamma\frac{1}{u_{ik}^t}\sum_{n=1}^{N} v_{nk}^t G_{nk}^t} = \sum_{m=1}^{D}[((1-2\lambda\gamma)u_{im}^t + 2\gamma\frac{1}{u_{im}^t}\sum_{n=1}^{N} v_{nm}^t G_{nm}^t)v_{jm}^{t+1} - u_{im}^t v_{jm}^t] \tag{18}$$

Let $\alpha_k = 2\gamma\sum_{n=0}^{N-1} v_{nk}^t G_{nk}^t$,

$$\frac{G_{jk}^t}{u_{ik}^t} - \frac{G_{jk}^{t+1}}{u_{ik}^t + \frac{\alpha_k}{u_{ik}^t}} = \sum_{m=1}^{D}[((1-2\lambda\gamma)v_{jm}^{t+1} - v_{jm}^t)u_{im}^t + \frac{\alpha_m v_{jm}^{t+1}}{u_{im}^t}] \tag{19}$$

From equation (13), we can have:

$$u_{im}^t = \frac{G_{jm}^t}{G_{jk}^t} u_{ik}^t \tag{20}$$

Plug equation (20) into (19):

$$\frac{G_{jk}^t}{u_{ik}^t} - \frac{G_{jk}^{t+1}}{u_{ik}^t + \frac{\alpha_k}{u_{ik}^t}} = \frac{u_{ik}^t}{G_{jk}^t}\sum_{m=1}^{D}[((1-2\lambda\gamma)v_{jm}^{t+1} - v_{jm}^t)G_{jm}^t] + \frac{G_{jk}^t}{u_{ik}^t}\sum_{m=1}^{D}[\frac{\alpha_m v_{jm}^{t+1}}{G_{jm}^t}] \tag{21}$$

Denote β_j and γ_j as follow:

$$\begin{cases} \beta_j = \sum_{m=1}^{D}[((1-2\lambda\gamma)v_{jm}^{t+1} - v_{jm}^t)G_{jm}^t] \\ \gamma_j = \sum_{m=1}^{D}[\frac{\alpha_m v_{jm}^{t+1}}{G_{jm}^t}] \end{cases} \tag{22}$$

We will have:

$$\frac{G_{jk}^t}{u_{ik}^t} - \frac{G_{jk}^{t+1}}{u_{ik}^t + \frac{\alpha_k}{u_{ik}^t}} = \frac{u_{ik}^t}{G_{jk}^t}\beta_j + \frac{G_{jk}^t}{u_{ik}^t}\gamma_j \tag{23}$$

Since we know there must be one real scalar of u_{ik}^t that satisfies equation (23). We can use some iterative methods to compute a numeric solution of (23), e.g., Newton's method.

After getting u_i^t, we can use equation (10) to compute r_i, which can be written as:

$$r_{ij} = \frac{G_{jk}^t}{u_{ik}^t} + \sum_{m=1}^{D} u_{im}^t v_{jm}^t \tag{24}$$

In summary, knowing the gradients of a user uploaded in two continuous steps, we can infer this user's rating information. Thus, we propose a secure matrix factorization framework based on homomorphic encryption, which will be elaborated in the next section.

5 FedMF: Federated Matrix Factorization

To overcome this information leakage problem, we propose to encode the gradients such that the server cannot inverse the encoding process. Then, the encoded data leaks no information. Meanwhile, the server should still be able to perform updates using the encoded gradients. One way to achieve such a goal is by using homomorphic encryption.

Figure 1 shows a framework of our method, called *FedMF* (Federated Matrix Factorization). Two types of participants are involved in this framework, the server and the users. As previously illustrated in Sec. 2.1, we assume that the server is honest-but-curious, the users are honest, and the privacy of the users is protected against the server.

Key Generation: As the typical functions involved in homomorphic encryption (Sec. 2.2), we first generate the *public key* and *secret key*. The key generation process is carried out on one of the users. The *public key* is known to all the participants including the server. And the *secret key* is only shared between users and needs to be protected against the server. After the keys are generated, different TLS/SSL secure channels will be established for sending the *public key* and *secret key* to the corresponding participants.

Parameter Initialization: Before starting the matrix factorization process, some parameters need to be initialized. The item profile matrix is initialized at the server side while the user profile matrix is initialized by each user locally.

Matrix Factorization: Major steps include,

1) The server encrypts item profile V using *public key*, getting the ciphertext C_V. From now on, the latest C_V is prepared for all users' download.
2) Each user downloads the latest C_V from the server, and decrypts it using *secret key*, getting the plaintext of V. V is used to perform local update and compute the gradient G. Then G is encrypted using *public key*, getting ciphertext C_V. Then a TLS/SSL secure channel is built, C_V is sent back to the server via this secure channel.
3) After receiving a user's encrypted gradient, the server updates the item profile using this ciphertext : $C_V^{t+1} = C_V^t - C_G$. Afterward, the latest C_V is prepared for users' downloading.
4) Step 2 and 3 are iteratively executed until convergence.

Security against Server: As shown in Fig. 1, only ciphertext is sent to the server in FedMF. So no bit of information will be leaked to the server as long as our homomorphic encryption system ensures ciphertext indistinguishability against chosen plaintext attacks [2].

No Accuracy Decline: We also claim that FedMF is accuracy equivalent to the user-level distributed matrix factorization. This is because the parameter updating process is the same as the distributed matrix factorization (Sec. 3) if the homomorphic encryption part is removed.

6 Prototype and Evaluation

In this section, we choose Paillier encryption method [7] to instantiate a prototype of our system and use a real movie rating dataset to evaluate the prototype.

6.1 Prototype Implementation

We use Paillier encryption [7] to build a prototype of FedMF. Paillier encryption is a probabilistic encryption schema based on composite residuosity problem. Given *publick key* and encrypted plaintext, paillier encryption has the following homomorphic property operations:

op1. $E(m_1) \cdot E(m_2)\ (mode\ n^2) = E(m_1 + m_2\ (mode\ n))$
op2. $E(m_1) \cdot g^{m_2}\ (mode\ n^2) = E(m_1 + m_2\ (mode\ n))$
op3. $E(m_1)^{m_2}\ (mode\ n^2) = E(m_1 m_2\ (mode\ n))$

Typically, paillier encryption requires the plaintext to be a positive integer. But in our system, the data are all in the form of floating point numbers and some of them might be negative. Thus we need to extend the encryption method to support our system.

Float. In brief, a base exponent was multiplied to the decimal, the integer part of the multiplication result I and the base exponent e was treated as the integer representation of the floating point number, i.e. (I, e). In the encryption process, only I is encrypted, the ciphertext will be (C_I, e). Then the ciphertext with the same e can directly conduct operation *op1* to get the encrypted summation of plaintext, ciphertext with different e needed to recalibrate such that e is the same. In practice, we use the same base exponent such that no information will leak from e.

Negative number. A *max number* parameter is set to handle the negative numbers. The *max number* can be set to half of n in *pk*, which means we assume all of our data is smaller than *max number*, such an assumption is easy to satisfy since n is usually set to a very large prime number. Then we perform mode n on all the plaintext, all the positive number have no changes and all the negative numbers become positive numbers greater than *max number*. In the decryption process, if the decrypted plaintext is greater than *max number*, we minus n to get the correct negative plaintext.

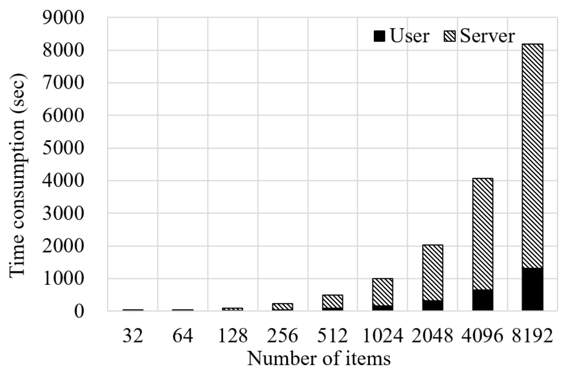
FullText or PartText. Usually the rating or feedback comprises a sparse matrix [5] which means the amount of feedback from a user could be very limited. Therefore, two different settings are implemented in our system. Both of them follow the overall steps of FedMF but are slightly different at the user uploading process. In one setting called *FullText*, users upload gradients for all the items; the gradient is set to 0 if a user does not rate an item. In the other setting called *PartText*, users only upload the gradients of the rated items. They both have advantages and disadvantages, *PartText* leaks information about which items the user has rated but has higher computation efficiency, *FullText* leaks no information but needs more computation time.

We utilize an open source python package, *python-paillier*[7] to accomplish the encryption part in our prototype system.

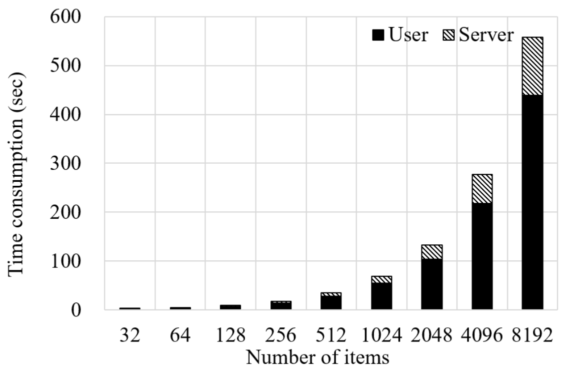
6.2 Evaluation

Dataset: We use two datasets in our experiments. The first one is MovieLens small dataset which is frequently used in other homomorphic-encrypted MF works such as [6] and [4], and it contains 100K ratings made by 610 users on 9724 movies. We also test our method on a large dataset: the MovieLens full dataset which contains 27M ratings made by 280K users on 58K items.

7. https://github.com/n1analytics/python-paillier



(a) FullText



(b) PartText

Fig. 2: User-Server time consumption ratio

Parameters: In Paillier encryption, we set the length of *public key* to 1024. The bandwidth of communication is set to *1 Gb/s*. In the matrix factorization process, we set the dimension of the user and item profile to 100.

Environment: All the test experiments are performed on a server with 3.7GHz 6-core CPU and 32GB RAM, where the operating system is Windows and the programming language is Python. We used a module called *gmpy*[8] to accelerate the homomorphic encryption part in Python such that it is as fast as C++ implementation.

TABLE 1: Time consumption (per iteration) on MovieLens-small dataset (second).

#Item	#Users	#Rating	Regular MF	Distributed MF	FedMF PartText	FedMF FullText
32	610	356	0.00429	0.00321	2.963	15.253
64	610	738	0.0082	0.00604	5.119	36.938
128	610	1420	0.0161	0.0104	9.518	95.021
256	610	2941	0.0308	0.0195	18.286	224.644
512	610	5227	0.0624	0.0378	35.759	487.755
1024	610	10615	0.131	0.0763	69.622	1004.051
2048	610	20758	0.251	0.154	133.113	2035.324
4096	610	42951	0.495	0.312	278.168	4080.073
8192	610	84994	0.997	0.618	560.156	8200.046

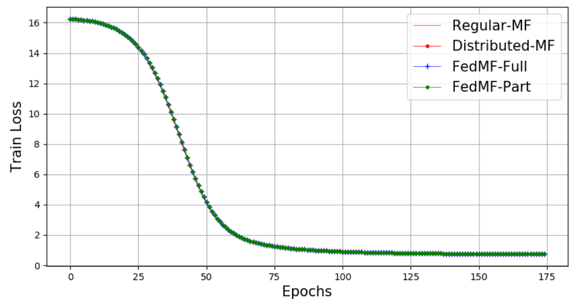


Fig. 3: Comparison of training loss

Performance: Since neither distributed computing or homomorphic encryption mechanisms will affect the computation values, FedMF will output the same user and item profiles as the original MF algorithm. Hence, the major objective of the experiments is testing the computation time of FedMF.

Fixing the number of users to 610, Table 1 shows the time consumption of each iteration of RegularMF, DistributedMF, FedMF-PartText and FedMF-FullText (one iteration means all of the 610 users' uploaded gradients are used to update the item profiles once). We assume that all the users update in parallel and the server starts the aggregation only after receives all the users' gradients. Thus the time consumption bottleneck of users' update is the one who has the most rating data, because it requires the most significant running time. For both *PartText* and *FullText*, the time consumption is quite good when there are not

TABLE 2: Time consumption (per iteration) on MovieLens full dataset.

#Item	#Users	#Rating	Regular MF (second)	Distributed MF (second)	FedMF PartText (hour)	FedMF FullText [*] (hour)
58K	5K	493K	5.833	3.520	0.4	∼110
58K	10K	994K	11.557	5.612	0.6	∼220
58K	20K	1.9M	23.646	9.125	1.0	∼470
58K	40K	3.8M	47.198	17.202	1.9	∼970
58K	80K	7.7M	93.813	31.350	2.8	∼1500
58K	160K	15M	184.706	63.340	8.1	∼4000
58K	283K	27M	324.601	114.592	15.3	∼8400

[*] We note that running FedMF FullText is very time consuming (e.g., one iteration for all the users/items can be up to one year). So, currently we estimate the running time rather than actual running. Particularly, the user's local running time is linearly correlated with the number of ratings, thus, we sample 100 users and calculate the averaged time consumption on one rating and estimate the running time for all the users.

TABLE 3: Comparison of converged results.

	RegularMF	DistributedMF	FedMF PartText	FedMF FullText
RMSE	1.39692[*]	1.39761[*]	1.39654	1.39654

[*] The results of RegularMF and DistributedMF are slightly different because in DistributedMF we make the server updates item-vector only when all the users have their gradient uploaded, and in RegularMF the server updates item-vector immediately after receiving gradient from an arbitrary user.

8. *gmpy* is a c-coded Python extension module that supports multiple-precision arithmetic, https://github.com/aleaxit/gmpy

too many items, and the time efficiency decreases when more items are given. Roughly, the time consumed for each iteration linearly increases with the number of items. Compared with *Fulltext*, *PartText* is more efficient but it leaks some information. Particularly, *PartText* is nearly 15 times faster than the *Fulltext* solution. The results also show that FedMF-PartText is about four orders of magnitude slower compared with DistributedMF because of the significant time consumption overhead caused by HE. However, if not using HE, the security of users' rating data cannot be guaranteed due to the possible gradient leakage. Hence, this is a trade-off between security and efficiency. Table 2 shows the time consumption on MovieLens full dataset. The time consumption of both PartText and FullText setting linearly increases with the number of users. When using the whole dataset (i.e., 280K users), it requires more than 15 hours to train one iteration in FedMF-PartText. This reveals that for applying FedMF or other homomorphic encryption-enhanced federated learning mechanisms to larger datasets, improving computation efficiency is an urgently required future research direction.

Figure 2a and 2b show the ratio of the user and server updating time when the number of items changes. The communication time is dismissed from the figures because it is too small compared with the user and server updating time. For example, 256MB of gradient data needs to be sent to the server when the item number is 8192, and it will cost about 4 seconds using a bandwidth of 1Gb/s. In the FullText setting, the server update costs most of the time. In contrast, the user update consumes most of the time in the PartText setting. The reason is that the server's updating time significantly increases in the FullText setting, while the users' updating time is bearly influenced because of the bottleneck mentioned above.

To verify that our FedMF system has no accuracy decline, we performed experiments on a small-scale dataset [9] which contains 40 movies, 7 users and 130 ratings. For each user, we select three rating records for testing and the other ratings compose the training dataset. Root Mean Square Error (RMSE) is used as metric to evaluate the model's performance. As illustrated in Figure 3 and Table 3, the RMSE of RegularMF/DistributedMF and FedMF are almost the same, showing that FedMF has no accuracy decline. It is worth nothing that the float numbers are transferred to integers using a fixed precision in pailliar encryption, thus the results of pailliar-encryption based methods (e.g., FedMF) will be slightly different from plaintext methods (e.g., DistributedMF).

Real world implementation: In real-world implementation, different from experiment settings, users' devices may not online simultaneously. Such circumstance will not affect FedMF system since the server can update item vector immediately after receiving gradient from one user instead of waiting all users' gradients received. The training process can be carried out when the users' devices are charging and remains extra computing resources.

9. Such a small dataset can help to reduce model convergence time consumption.

7 Conclusion and Future Work

In this paper, we propose a novel secure matrix factorization framework in federated machine learning, called *FedMF*. More specifically, we first prove that a distributed matrix factorization system where users send gradients to the server in forms of plaintext will leak users' rating information. Then, we design a homomorphic encryption based secure matrix factorization framework. We have proved that our system is secure against an honest-but-curious server, and the accuracy is the same as the matrix factorization on users' raw data.

Experiments on real-world data show that FedMF's time efficiency is acceptable when the number of items is small. Also note that our system's time consumption linearly increases with the number of items. To make FedMF more practical in reality, we still face several challenges:

More efficient homomorphic encryption. FedMF shows large time consumption overhead because of the low efficiency of the HE algorithm. If we can improve the HE's efficiency when conducting operations on ciphertext, our system's performance will increase.

Between FullText and PartText. Our experiments have shown that *PartText* is much more efficient than *FullText*, but *PartText* reveals the set of items rated by a user. This information, without the exact rating scores, may still leak users' sensitive information [9]. Perhaps we can ask users to upload more gradients than only the rated items, but not all the items, so as to increase efficiency compared to *FullText*, while not leaking the exactly rated item set.

More secure definitions. We currently use a typical horizontal federated learning secure definition, which assumes honest participants and an honest-but-curious server. Such a secure definition is relatively weak because there might be malicious users in real-world applications. Malicious users may collude with the server (i.e., share the private key) to attack other users, causing privacy issues. They also may perform backdoor attacks to the system, which will bring security issues. In the future, we will explore more challenging secure definitions, such as building a secure system where the server is honest-but-curious, and some participants are malicious, and the malicious participants may collude with the server.

Acknowledgment

Leye Wang is partially supported by the NSFC under Grant No.61972008. Qiang Yang is partially supported by the National Key Research and Development Program of China under Grant No.2018AAA0101100.

References

[1] Ammad-ud-din, M., Ivannikova, E., Khan, S. A., Oyomno, W., Fu, Q., Tan, K. E., and Flanagan, A. Federated collaborative filtering for privacy-preserving personalized recommendation system. *CoRR abs/1901.09888* (2019).

[2] Aono, Y., Hayashi, T., Wang, L., Moriai, S., et al. Privacy-preserving deep learning via additively homomorphic encryption. *IEEE Transactions on Information Forensics and Security 13*, 5 (2017), 1333–1345.

[3] Berlioz, A., Friedman, A., Kaafar, M. A., Boreli, R., and Berkovsky, S. Applying differential privacy to matrix factorization. In *Proceedings of the 9th ACM Conference on Recommender Systems* (2015), ACM, pp. 107–114.

[4] KIM, S., KIM, J., KOO, D., KIM, Y., YOON, H., AND SHIN, J. Efficient privacy-preserving matrix factorization via fully homomorphic encryption. In *Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security* (2016), ACM, pp. 617–628.

[5] KOREN, Y., BELL, R., AND VOLINSKY, C. Matrix factorization techniques for recommender systems. *Computer*, 8 (2009), 30–37.

[6] NIKOLAENKO, V., IOANNIDIS, S., WEINSBERG, U., JOYE, M., TAFT, N., AND BONEH, D. Privacy-preserving matrix factorization. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security* (2013), ACM, pp. 801–812.

[7] PAILLIER, P. Public-key cryptosystems based on composite degree residuosity classes. In *International Conference on the Theory and Applications of Cryptographic Techniques* (1999), Springer, pp. 223–238.

[8] WANG, L., ZHANG, D., WANG, Y., CHEN, C., HAN, X., AND M'HAMED, A. Sparse mobile crowdsensing: challenges and opportunities. *IEEE Communications Magazine 54*, 7 (2016), 161–167.

[9] YANG, D., ZHANG, D., QU, B., AND CUDRÉ-MAUROUX, P. Privcheck: privacy-preserving check-in data publishing for personalized location based services. In *Proceedings of the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing* (2016), ACM, pp. 545–556.

[10] YANG, Q., LIU, Y., CHEN, T., AND TONG, Y. Federated machine learning: Concept and applications. *ACM Transactions on Intelligent Systems and Technology (TIST) 10*, 2 (2019), 12.

AUTHOR BIOGRAPHY

Di Chai is a Ph.D student in computer science and engineering at Hong Kong University of Science and Technology. He got his master degree of science from Hong Kong Unviersity of Science and Technology, in 2018. His research interests include federated learning and privacy-preserving machine learning. Contact him at dchai@connect.ust.hk.

Leye Wang is an assistant professor at Key Lab of High Confidence Software Technologies, Peking University, China. His research interests include ubiquitous computing, mobile crowdsensing, and urban computing. Wang received a Ph.D. in computer science from the Institut Telecom SudParis and University Paris 6, France, in 2016, and was a postdoc researcher with Hong Kong University of Science and Technology. Contact him at leyewang@pku.edu.cn.

Kai Chen is an Associate Professor with the Department of Computer Science and Engineering, Hong Kong University of Science and Technology, Hong Kong. He received a Ph.D. degree in computer science from Northwestern University, Evanston, IL in 2012. His research interest includes data center networking, machine learning systems, and privacy-preserving AI. Contact him at kaichen@cse.ust.hk.

Qiang Yang is the chief AI officer in WeBank and a chair professor of engineering at Hong Kong University of Science and Technology. His research interests include data mining and artificial intelligence including machine learning, planning, and case-based reasoning. He received a Ph.D. in computer science from the University of Maryland, College Park, in 1989. He was founding editor in chief of ACM Transactions on Intelligent Systems and Technology and IEEE Transactions of Big Data. He is a Fellow of the IEEE, AAAI, and ACM. Contact him at qyang@cse.ust.hk.