PPGloVe: Privacy-Preserving GloVe for Training Word Vectors in the Dark

Zhongyun Hua, *Senior Member, IEEE*, Yan Tong, Yifeng Zheng, Yuhong Li, and Yushu Zhang, *Senior Member, IEEE*

***Abstract*— Words are treated as atomic units in natural language processing tasks and it is a fundamental step to represent them as vectors for supporting subsequent computations. GloVe is a widely used machine learning model to train word vectors. Generally, a large corpus and high computation resources are required to train high-quality word vectors using GloVe, making it difficult for users to train their own word vectors by themselves. A natural choice nowadays is to outsource the training process to the cloud. However, coming with such cloud-based training services are serious privacy concerns, which should be well addressed. In this paper, we design, implement, and evaluate PPGloVe, the first system framework that supports privacy-preserving word vectors training using GloVe over encrypted data of multiple participants. We first decompose the training task and show that previous privacy-preserving machine learning techniques are not practical for this task. We then construct a new secure training strategy to delicately bridge lightweight cryptographic techniques with GloVe in depth to support privacy-preserving GloVe training on the cloud. By design, the corpora of the participants and the trained word vectors are kept private along the whole training process. Extensive experiments over three datasets of different scales demonstrate that PPGloVe produces word vectors with promising quality comparable to plaintext training, with practically affordable overhead.**

***Index Terms*— Privacy preservation, data security, word representation, cloud computing.**

Manuscript received 5 April 2023; revised 27 July 2023, 5 September 2023, and 22 December 2023; accepted 2 February 2024. Date of publication 8 February 2024; date of current version 26 April 2024. This work was supported in part by the National Natural Science Foundation of China under Grant 62071142; in part by the Guangdong Basic and Applied Basic Research Foundation under Grant 2021A1515110027 and Grant 2023A1515010714; in part by the Shenzhen Science and Technology Program under Grant JCYJ20220531095416037, Grant JCYJ20230807094411024, Grant RCBS20210609103056041, and Grant ZDSYS20210623091809029; and in part by the Guangdong Provincial Key Laboratory of Novel Security Intelligence Technologies under Grant 2022B1212010005. The associate editor coordinating the review of this manuscript and approving it for publication was Prof. Edgar Weippl. *(Corresponding author: Yifeng Zheng.)*

Zhongyun Hua is with the School of Computer Science and Technology, Harbin Institute of Technology, Shenzhen, Guangdong 518055, China, and also with the Guangdong Provincial Key Laboratory of Novel Security Intelligence Technologies, Shenzhen, Guangdong 518055, China (e-mail: huazyum@gmail.com).

Yan Tong and Yifeng Zheng are with the School of Computer Science and Technology, Harbin Institute of Technology, Shenzhen, Guangdong 518055, China (e-mail: 594tongyan@gmail.com; yifeng.zheng@hit.edu.cn).

Yuhong Li is with Xiaohongshu Technology Company Ltd., Shanghai 200092, China (e-mail: liyuhong1@xiaohongshu.com).

Yushu Zhang is with the College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing, Jiangsu 210016, China (e-mail: yushu@nuaa.edu.cn).

Digital Object Identifier 10.1109/TIFS.2024.3364080

I. Introduction

WORDS are treated as atomic units in natural language processing (NLP) tasks and it is a fundamental step in these tasks to convert words to vectors so as to support subsequent computations. GloVe [1] is a widely popular word vector model to generate high-quality word vectors and has been widely used in NLP applications [2], [3]. Since a large corpus and high computation resources are required to train high-quality word vectors via GloVe, it is not easy for users to train their own word vectors themselves. Thus, an effective choice for a user is to collaborate with others and outsource the training over their joint data to the cloud. However, this comes with serious security concerns. On one hand, the corpora may contain privacy-sensitive information and the data owners may not be willing to share them with others. On the other hand, users should pay for the computation resources of cloud for the training and they may not want others to access the well-trained word vectors without being authorised. Thus, it is very important to protect the corpora and word vectors when collaboratively training word vectors on the cloud.

Privacy-preserving machine learning (ML) has received wide attentions in recent years [4], [5], [6], [7], [8], [9]. The existing privacy-preserving ML techniques can be divided into two categories in general. The first category aims to securely train a ML model in the dark so that the training data and trained model can be well protected [9], [10], [11], while the second category focuses on protecting user data and model parameters during the model inference stage [8], [12], [13], [14], [15]. One can use the model to obtain desired output with his/her data being protected, but cannot know any useful information about the model parameters. Both categories can well protect the confidentiality of user data and model parameters.

Along the fast growing trend in privacy-preserving ML, a line of work has been particularly focused on the NLP domain [16], [17], [18], [19], [20]. For example, the work in [16] utilized secure multi-party computation techniques to build a secure sequence-to-sequence model that can be applied in neural machine translation applications, and the works in [18] and [19] implemented privacy-preserving text classification tasks using private feature extraction. Most of these works focus on protecting user data and model parameters during the inference stage. The works in [17] and [20] utilize homomorphic encryption [21], [22] to protect the Word2Vec model [23] and fastText model [24], respectively, during model training. However, homomorphic cryptosystems typically encrypt data using heavy cryptographic operations,

incurring significant computation overhead and data size expansion. As reported in [17], the largest size of the training dataset being evaluated is limited to only 80 MB, which is clearly insufficient for many practical applications.

Although there has been good progress in privacy-preserving ML in recent years, it is yet still challenging to securely and efficiently train word vectors using GloVe due to the following reasons. Firstly, a new secure mechanism should be designed to efficiently compute the natural logarithm in the GloVe model. Previous privacy-preserving model training works [6], [10], [11], [25] primarily focus on the secure training of neural networks and realize the secure computation of some relatively simple nonlinear functions such as ReLU, sigmoid, and maxpool. Several secure natural logarithm computation methods have been developed in prior works [26], [27], [28]. However, these methods either support only a small effective domain [26] or realize secure computation using heavyweight techniques such as garbled circuit in [27] and homomorphic encryption in [28], which are far from sufficient to support practical applications (more detailed discussions are given in Section III-C-Q1). Secondly, GloVe needs to be trained over a large amount of data for high utility. It will lead to low efficiency if we directly follow the plaintext-domain training strategy for ciphertext-domain training. Thus, we should bridge cryptographic techniques with the GloVe model in depth to achieve privacy preservation and high efficiency.

In light of the above, in this paper, we propose, implement, and evaluate PPGloVe, the first system framework to collaboratively train word vectors using GloVe with privacy preservation on the cloud. Since it requires a large corpus to train high-quality word vectors, we build on a lightweight cryptographic technique, additive secret sharing [29], to protect the training data and word vectors. To adapt to the ciphertext-domain training, we re-organize the workflow of GloVe and have three stages in PPGloVe: system initialization, data processing and secure word vectors training. All the entities prepare for training in the first stage. We separate the time-consuming operations and securely compute them in advance in the second stage so that they need to be computed only once. We securely train the word vectors in the third stage using the previous obtained results. To reduce the communication cost, we use a more efficient learning rate updating strategy without communication cost to replace the original one used in GloVe. Since the previous secure natural logarithm computation methods [26], [27], [28] are inefficient for our task, we design a new secure natural logarithm computation method which is more than $10^4\times$ faster than prior methods. Besides, we also optimize secure multiplications which share the same multiplier and reduce the communication overhead by nearly half. We implement our protocols and conduct evaluations on three datasets of different scales. The experiment results show that PPGloVe has modest performance overhead and can generate word vectors with comparable quality to plaintext training. We summarize our contributions as follows.

- We present PPGloVe, the first system framework that allows users to collaboratively train word vectors based on the GloVe model at the cloud, while protecting user data and word vectors during the whole training process.
- We delicately bridge lightweight cryptographic techniques with GloVe training in depth to support privacy preservation, and design an efficient secure natural logarithm computation method that is more than $10^4\times$ faster than existing methods.
- We analyze the complexity of PPGloVe and formally justify its security. Extensive experiments on datasets of different scales demonstrate that PPGloVe can produce word vectors with quality comparable to the plaintext-domain training, with practically affordable overhead.

The rest of this paper is organized as follows. Section II introduces some preliminaries. We give the problem statement in Section III. Then, we present the design of PPGloVe in Section IV. We analyze the complexity of PPGloVe and prove its security in Section V. We then show our experiment results in Section VI, and discuss the related work in Section VII. Finally, we conclude this paper in Section VIII.

TABLE I
KEY NOTATIONS

Notation	Description
ℓ	Bit length
m	Dimension of word vector
$v(i)$	The i-th element in vector v
$\mathbf{M}_i$	The co-occurrence matrix constructed by the i-th user
v_i, d_i	The word vector for word i and the bias for v_i
$\langle x\rangle^{(b)}$	Arithmetic share of value x held by server b
$[\![x]\!]^{(b)}$	Boolean share of value x held by server b
$\langle x\rangle^{(b)} \pm \langle y\rangle^{(b)}$	Addition/subtraction over arithmetic shares
$\langle x\rangle \cdot \langle y\rangle$	Multiplication over arithmetic sharing
$[\![x]\!]^{(b)} \oplus [\![y]\!]^{(b)}$	Bitwise XOR over Boolean shares
$[\![x]\!] \wedge [\![y]\!]$	Bitwise AND over Boolean sharing

II. PRELIMINARIES

In this section we introduce some necessary preliminaries. And in Table I we list some key notations used in this paper.

A. Global Vector

The Global Vector (GloVe) model is an unsupervised learning algorithm for obtaining vector representations of words [1]. It uses sliding window to traverse the corpus and updates the co-occurrence matrix in the sliding window, combining the advantage of the global matrix and local context window. GloVe is one of the most popular models for training word vectors, and has been widely used in many NLP tasks [3]. GloVe takes the co-occurrence statistics of words as the primary information for learning word representations. Let $\mathbf{M}$ denote the word-to-word co-occurrence matrix, and its element $M_{j,k}$ denotes the times that the word k occurs in a same sliding window with the word j. Specifically, the GloVe model builds the matrix $\mathbf{M}$ using a decreasing weighting function. Therefore, the word pair (j,k) with r words distance in the sliding window contributes $1/r$ to $M_{j,k}$. The element $M_j = \sum_k M_{j,k}$ denotes the times that any word appears in the context of the word j. We also use $P_{j,k} = P(k|j) = M_{j,k}/M_j$ to represent the probability that the word k appears in the context of the word j.

The starting point for learning word vectors is the ratios of the co-occurrence probabilities, rather than the probabilities

themselves. Specifically, given two words h and j in a particular aspect, the ratio of their co-occurrence probabilities with probe words k is used to estimate the relationship between the two words. For example, let $h = ice$ and $j = steam$, the ratio $P_{h,k}/P_{j,k}$ is much larger than 1 if k has closer relationship with the word h than with the word j (e.g., $k = solid$). Otherwise, the ratio $P_{h,k}/P_{j,k}$ is much smaller than 1 if k has closer relationship with the word j than with the word h. Besides, the ratio $P_{h,k}/P_{j,k}$ approaches to 1 if k has a similar relationship with words h and j (e.g., $k = water$ or $fashion$).

Drawing upon the above observation, a general form of the model can be constructed as

$$F(v_h, v_j, \tilde{v}_k) = \frac{P_{h,k}}{P_{j,k}}, \tag{1}$$

where $v \in \mathbb{R}^n$ are word vectors and $\tilde{v} \in \mathbb{R}^n$ are separate context word vectors. They are initialized randomly and trained equivalently. The function F is recommended as the exponential function. Then vectors should satisfy that

$$v_j^T \tilde{v}_k + d_j + \tilde{d}_k = \ln(M_{j,k}), \tag{2}$$

where d_j and $\tilde{d}_k$ denote the bias for v_j and $\tilde{v}_k$. To avoid treating all the co-occurrences of words equally, a weighting function $f(x)$ is introduced in the loss function and defined as

$$f(x) = \begin{cases} (x/x_{\max})^\alpha & \text{if } x < x_{\max}; \\ 1 & \text{otherwise}, \end{cases} \tag{3}$$

where x_{max} and α are hyperparameters and α is set to 3/4 for empirical motivation in GloVe. Finally, the total loss function is obtained as

$$\boldsymbol{J} = \frac{1}{2} \sum_{j,k \in W} f\left(M_{j,k}\right) \left(v_j^T \tilde{v}_k + d_j + \tilde{d}_k - \ln(M_{j,k})\right)^2, \tag{4}$$

where W is the corpus vocabulary. It can be seen that the v_j, $\tilde{v}_k$, d_j, and $\tilde{d}_k$ need to be updated during training. To reduce noise for a better performance, the word i is finally represented as $(v_i + \tilde{v}_i)/2$, rather than only v_i. For simplicity, we refer to both v and $\tilde{v}$ as word vectors in this paper. We use $\mathbf{V}$ to denote the set of all the word vectors v and $\tilde{v}$, and use $\mathbf{D}$ to denote the set of all the biases d and $\tilde{d}$.

B. Cryptographic Primitives

1) Arithmetic Secret Sharing: Given an ℓ-bit private value x in ring $\mathbb{Z}_{2^\ell}$, the arithmetic secret sharing [29] can protect it by splitting it into two secret shares $\langle x \rangle^{(1)}, \langle x \rangle^{(2)} \in \mathbb{Z}_{2^\ell}$ such that $\langle x \rangle^{(1)} + \langle x \rangle^{(2)} \equiv x \pmod{2^\ell}$. Each share is a uniformly distributed random value in $\mathbb{Z}_{2^\ell}$ and reveals no information about x. We denote the arithmetic secret sharing of x as $\langle x \rangle$ for short. Suppose that the two secret shares of x and y are held by two parties $P_{(1)}$ and $P_{(2)}$, respectively. Then some computations among the two parties can be supported. Firstly, the secure addition ($SecAdd$) and secure subtraction ($SecSub$) over shares ($\langle z \rangle^{(b)} = \langle x \rangle^{(b)} \pm \langle y \rangle^{(b)}, b \in \{1, 2\}$), and the secure multiplication over a share and a public value γ ($\langle z \rangle^{(b)} = \gamma \cdot \langle x \rangle^{(b)}$) can be computed locally at each party. Secondly, the secure multiplication ($SecMul$) between two secret-shared values ($\langle z \rangle = \langle x \rangle \cdot \langle y \rangle$) requires one round of communication. Specifically, a Beaver triple ($\langle l \rangle, \langle o \rangle, \langle p \rangle$) is needed to assist in the computation [30], where $p = l \cdot o$. Such triples can be generated offline by a third party [31]. The $SecMul$ is realized as follows. (1) Each party $P_{(b)}$ ($b \in \{1, 2\}$) computes $\langle \beta \rangle^{(b)} = \langle x \rangle^{(b)} - \langle l \rangle^{(b)}$ and $\langle \delta \rangle^{(b)} = \langle y \rangle^{(b)} - \langle o \rangle^{(b)}$; (2) The two parties collaboratively reconstruct β and δ; (3) Each party computes $\langle z \rangle^{(b)} = (b-1) \cdot \beta \cdot \delta + \delta \cdot \langle l \rangle^{(b)} + \beta \cdot \langle o \rangle^{(b)} + \langle p \rangle^{(b)}$. The communication overhead for $SecMul$ is 4ℓ bits [32].

2) Boolean Sharing: The Boolean sharing can be regarded as a special case of arithmetic sharing with $\ell = 1$. The Boolean shares are within the ring $\mathbb{Z}_2$, and the addition and subtraction over shares can be replaced by the XOR ($\oplus$) operation. Besides, the multiplication operation in the Boolean sharing can be replaced by the AND ($\wedge$) operation that needs a Beaver AND triple to assist in computation. To distinguish Boolean sharing from arithmetic sharing, we use $[\![x]\!]^{(1)}$ and $[\![x]\!]^{(2)}$ to represent the two Boolean shares of x, and also use $[\![x]\!]$ to denote Boolean sharing for short.

C. Real Number Representation

The operands in the secret sharing primitives are all integers within the ring $\mathbb{Z}_{2^\ell}$. However, the computation of GloVe in our task contains real numbers. In our design, we use the fixed-point representation to handle the real numbers, following the priors works in [6] and [10]. Specifically, a real number x is scaled and rounded to an integer $\overline{x} = \lfloor x \cdot 2^q \rfloor \pmod{2^\ell}$ within the ring $\mathbb{Z}_{2^\ell}$, where q is the scaling factor used to control the precision. Note that the multiplication of two scaled fixed-point values leads to a scaling factor $2q$, which may exceed the bit length ℓ in the ring $\mathbb{Z}_{2^\ell}$. Therefore, the multiplication result should be scaled down by 2^q before the subsequent operation. Our scaling-down operation follows the local truncation scheme in [6], which simply discards the last q-bit fraction part. This truncation scheme has been proved to incur at most 1-bit error with a high probability when the ring size is large enough.

III. Problem Statement

Training high-quality word vectors with GloVe requires a large corpus and significant computational resources, making it challenging for individual users. Thus, users may try to collaborate with others and outsource the training over their joint data to the cloud. Since the cloud can access the plaintext of user data and word vectors, the training process comes with privacy concerns. To address this, we propose PPGloVe as the first system framework that supports privacy-preserving word vector training using GloVe. PPGloVe ensures that during the training process, both user data and word vectors remain encrypted. Below we first introduce the system architecture of PPGloVe, and then introduce the threat model and design goals. After that, we elaborate on the design challenges.

A. System Architecture

Fig. 1 illustrates the system architecture of PPGloVe. It works under a 2PC model and contains three types of entities: user, cloud server, and key server.

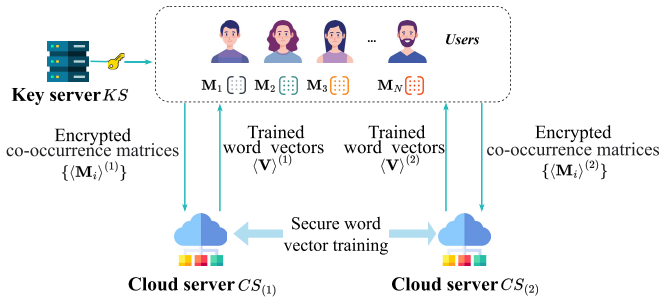


Fig. 1. The system architecture of our PPGloVe.

1) User: The system contains N participating users and each user is denoted as $\mathcal{U}_i$ ($i \in [1, N]$). These users aim to collaboratively train high-quality word vectors. Each user $\mathcal{U}_i$ firstly counts a word-to-word co-occurrence matrix $\mathbf{M}_i$ using his/her own corpus locally. Since $\mathbf{M}_i$ uses word pairs as indices (e.g., the element $M_{I,am}$ means the co-occurrence of the word pair (I, am)), the user $\mathcal{U}_i$ replaces each word appearing in $\mathbf{M}_i$ with a secure token generated by a pseudo-random function (PRF), to protect the words from being known by the servers. Finally, each user $\mathcal{U}_i$ splits $\mathbf{M}_i$ into two shares $\langle \mathbf{M}_i \rangle^{(1)}$ and $\langle \mathbf{M}_i \rangle^{(2)}$ and sends them to two cloud servers, respectively, for jointly training word vectors with the co-occurrence matrices of other users.

2) Cloud Server: Since GloVe needs to be trained over a large amount of data for high utility, we employ the lightweight secret sharing technique and a 2-party computation (2PC) model to realize PPGloVe. This 2PC model has been increasingly adopted in recent works on privacy-preserving ML [6], [7], [10], [33], [34]. In such a model, data are encrypted into two shares and sent to two cloud servers in different trust domains for secure training. The two cloud servers collaborate to execute designated secure protocols. Since the two cloud servers are in different trust domains, they are assumed to not collude with each other. Our design also follows this trend and the system has two cloud servers $\mathcal{CS}_{(1)}$ and $\mathcal{CS}_{(2)}$ for collaboration in securely training word vectors. After receiving the encrypted matrices $\{\langle \mathbf{M}_i \rangle^{(b)}\}$ ($b \in \{1, 2\}$) from the users, each server combines these matrices to obtain a share $\langle \mathbf{M} \rangle^{(b)}$ of the big co-occurrence matrix $\mathbf{M}$ that contains the global information of all users' corpora. Then $\mathcal{CS}_{(1)}$ and $\mathcal{CS}_{(2)}$ collaboratively carry out the custom protocols in PPGlove without seeing users' data and each of them produces $\langle \mathbf{V} \rangle^{(b)}$, which is a secret share of the trained word vectors $\mathbf{V}$.

3) Key Server: The key server KS in our system is used to generate and issue the PRF key, which is used by all the users to generate secure tokens for the words appearing in the co-occurrence matrix. The KS only works during the system's initialization phase and it does not participate in secure computation for the actual model training process. Once the key has been generated and issued, the KS can be offline. It is noted that the key server is not a necessary entity in our design, e.g., its functionality could be replaced through group key agreement protocols run among the users.

B. Threat Model and Design Goals

Similar to prior works on privacy-preserving ML [35], [36], [37] under the two-server model, PPGloVe considers a semi-honest and non-colluding adversary model regarding the two cloud servers. Specifically, we consider the scenario where the two cloud servers honestly execute the designated protocols and do not collude with each other, but they are interested in learning the users' private data and trained word vectors from the protocol execution. This assumption is reasonable because cloud service providers typically come from well-established companies like Amazon and Google, and thus, they have little incentive to compromise their reputation [33]. In addition, the key server KS and the users are considered to be trustworthy [17], [38].

Under the above threat model, our PPGloVe aims to provide privacy-preserving and fast collaborative training for GloVe. The specific design goals are as follows.

- **Security**: The security goals include **data security** and **model security**. (a) The data security indicates that the user's word-word co-occurrence matrix $\mathbf{M}_i$ and the words appearing in it should not be leaked to the cloud servers. (b) The model security indicates that the cloud servers can not learn anything about the trained word vectors.
- **Efficiency**: The communication and computation cost during the whole training process should be as small as possible.
- **Utility**: The word vectors trained in the ciphertext domain should have comparable utility to that trained in the plaintext domain.

C. Design Challenges

To achieve the aforementioned design goals, the following questions should be specially considered.

Q1: How to efficiently and securely compute the natural logarithm $\ln(x)$ *during training?* As shown in Eq. (4), the loss function of GloVe contains the natural logarithm $\ln(x)$. To securely compute the nonlinear functions is a time-consuming and difficult problem in privacy-preserving ML. Several secure natural logarithm computation methods have been proposed in prior works [26], [27], [28]. The method in [26] is based on the secret sharing technique and it uses McLaughlin series to approximate the natural logarithm. It is only effective for inputs near zero, because the McLaughlin series converges slowly when the input is far from 0, making its effective input domain very small. However, the input values (elements in co-occurrence matrix $\mathbf{M}$) in our task have a large range (e.g., $(10^{-2}, 10^{6})$) and large errors will occur if directly using this method to our task. Therefore, this method is not suitable for our task. The other methods in [27] and [28] implement secure natural logarithm computation using either garbled circuit or homomorphic encryption. However, both of the used techniques are time-consuming. According to the results in [28], it takes about 6s for the method [28] and about 10s for the method [27] to securely compute the natural logarithm using a workstation (Intel Xeon 2.3 GHz CPU (16 cores)). For a corpus with size 91MB, it is estimated to take more than 3,000 days to compute all the secure natural logarithms in our task when using these methods, which is unacceptable in practice. Thus, it is required to design a more efficient secure natural logarithm computation method. Besides, the training will iterate several epochs, leading to the repeat of the natural logarithm computation $\ln(x)$ and weighting function $f(x)$ in the ciphertext-domain for the same training sample. So the

whole process of GloVe should be re-designed to achieve computation savings in ciphertext-domain training.

Q2: How to reduce the communication overhead of vector-element multiplication in the ciphertext-domain? The secure vector-element multiplication accounts for a large proportion of the communication cost between two cloud servers in the training process. One implementation of the secure vector-element multiplication is to convert it into multiple element-element multiplications in the ciphertext domain. However, due to the high dimension of word vectors and large number of training samples, such way will introduce substantial communication overhead. For example, when the size of the corpus is around 600MB and the dimension of the word vectors is 100, the communication cost of secure vector-element multiplication exceeds 3,000GB in one training epoch. Thus, the secure vector-element multiplication should be optimized to reduce communication overhead.

Q3: How to dynamically adjust the learning rate? During the training process, ML model parameters will gradually converge to the optimal point. Therefore, the learning rate should also be gradually reduced to achieve fine-tuning of the model. The Adagrad optimizer [39] is used to realize the dynamic adjustment of the learning rate in GloVe. This optimizer involves division and square root operations, which can not be performed directly in the secret sharing domain. Previous study [40] uses the Newton-Raphson [41] method for approximation to solve this. However, due to the large number of parameters and training samples, this method will incur large communication overhead in our task and cannot meet the requirements of practicability. Taking a corpus with about 600MB as an example, this method will require more than 10^5 GB communication cost in one training epoch in our task, according to our estimation. Therefore, we should adopt an efficient and secure strategy to dynamically adjust the learning rate during the process of ciphertext-domain training.

IV. Design of PPGloVe

A. Design Overview

Here, we present an overview about how our design can address the challenges described in Section III-C. Firstly, we propose a new mechanism to securely and efficiently compute the nonlinear functions from two aspects. (1) We propose a new secure natural logarithm computation method $SecLn(\cdot)$ using the lightweight secret sharing technique. Following prior work in [28], we also adopt the scaling-first and approximating-second computation order, but propose a novel and more effective realization fully based on the lightweight secret sharing. Concretely, for each input, we use secure comparison to determine a factor $s = 2^n$, where s is the smallest power of 2 that is larger than the input, and use this factor to scale the input into the range of $[-\frac{1}{2}, 0)$. Then we use the Taylor series to approximate the final result in the secret sharing domain. This design has a very high efficiency and supports the input with large value, causing only a minor accuracy loss. (2) Considering that the natural logarithm $\ln(x)$ and weighting function $f(x)$ are computed repeatedly for each training sample, we aim to compute them only once by separating their computations with other operations and reuse the results. To implement this, we divide the training process of GloVe into three stages: **system initialization**, **data processing** and **secure word vectors training**, and move the computations of $f(x)$ and $\ln(x)$ to the stage of data processing. Besides, the $f(x)$ and $\ln(x)$ with different inputs can be computed in advance with parallel processing, which can further accelerate the calculation process.

Secondly, we optimize the secure vector-element multiplication to reduce the communication overhead between the two cloud servers. The secure vector-element multiplication contains numerous independent element-element multiplications. Since these element-element multiplications have a common multiplier, we follow the very recent work in [42] and use the correlated Beaver triples to replace the multiple independent Beaver triples to assist the secure multiplication operation. By this design, the common multiplier of these element-element multiplications needs to be masked and reconstructed only once, resulting in half communication overhead reduction.

Thirdly, we adopt an efficient strategy from [43] to dynamically adjust the learning rate, ensuring that the learning rate adjustment can meet the security and performance requirements of practical training. This strategy calculates the learning rate using the numbers of all the training samples and currently used samples, which guarantees that the learning rate gradually decreases as the training progresses. Since this method does not involve any private information of the training data, it can be computed in the plaintext-domain without introducing any communication overhead. Fig. 2 illustrates a high-level overview of the system initialization, data processing, and secure word vectors training in PPGloVe.

B. System Initialization

All the entities in PPGloVe participate in the system initialization. The key server KS generates a secret key $key \xleftarrow{\$} \{0,1\}^\lambda$ for a PRF function $F_{key}(\cdot)$, and then sends it to all the users through a secure channel. After this operation is performed, we emphasize that the key server can go offline and does not participate in the subsequent secure computation process. Only the two cloud servers in our PPGloVe participate in the subsequent secure computation for the actual model training process.

Each user $\mathcal{U}_i$ generates two shares of co-occurrence matrix and sends them to the two cloud servers as follows.

- Construct a word-to-word co-occurrence matrix $\mathbf{M}_i$ using his/her own private corpus locally. The construction process is the same with that in GloVe [1], which is described in Section II-A.
- Generate a token $\pi_j = F_{key}(j)$ for each word j appearing in $\mathbf{M}_i$ and replace the word using the token. During the training process, these tokens will be used as secure indices to index the elements in $\langle \mathbf{M}_i \rangle$ by cloud servers without exposing the original words.
- Convert all the non-zero elements in $\mathbf{M}_i$ to fixed-point representation.
- Share $\mathbf{M}_i$ to generate $\langle \mathbf{M}_i \rangle^{(1)}$ and $\langle \mathbf{M}_i \rangle^{(2)}$, where $\langle \mathbf{M}_i \rangle^{(1)} + \langle \mathbf{M}_i \rangle^{(2)} \equiv \mathbf{M}_i \pmod{2^\ell}$.
- Send $\langle \mathbf{M}_i \rangle^{(1)}$ and $\langle \mathbf{M}_i \rangle^{(2)}$ with the secure tokens to $CS_{(1)}$ and $CS_{(2)}$, respectively. Only non-zero items are set to the cloud servers, since the zero values are not used for training.

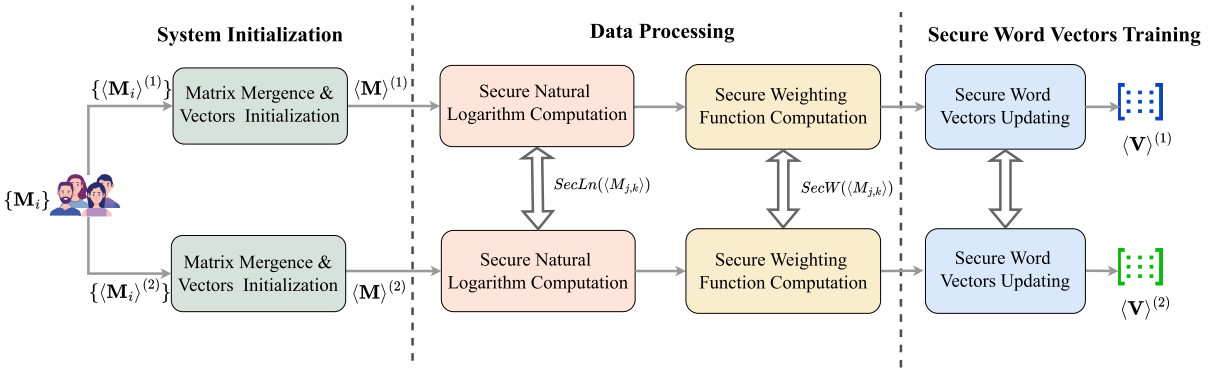


Fig. 2. Workflow of our PPGloVe model.

Each cloud server $CS_{(b)}$ ($b \in \{1, 2\}$) merges the received $\{\langle \mathbf{M}_i \rangle^{(b)}\}$ of all the users into $\langle \mathbf{M} \rangle^{(b)}$ by summing the values of the same token-pairs in $\{\langle \mathbf{M}_i \rangle^{(b)}\}$. Then, $CS_{(b)}$ randomly initializes $\langle \mathbf{V} \rangle^{(b)}$ as the share of the word vectors and $\langle \mathbf{D} \rangle^{(b)}$ as the share of the bias values. Both of them are indexed by the secure tokens, rather than the original words.

C. Data Processing

Since the natural logarithm $\ln(x)$ and weighting function $f(x)$ in Eq. (4) are computed repeatedly with the same inputs $M_{j,k}$ in each training epoch of GloVe, we calculate them in the ciphertext-domain in advance and reuse the results later. Besides, as the computations for each training sample are independent, all the natural logarithm and weighting function computations can be executed with parallel processing, which can greatly reduce the communication rounds and accelerate the computation process.

1) Secure Natural Logarithm: The natural logarithm is a kind of complex nonlinear operations and is difficult to be directly computed in the ciphertext-domain. A commonly used method is to approximate the natural logarithm using some basic arithmetic operations such as Taylor series. However, the Taylor series of the natural logarithm will cause a large error if its input is a large value. To overcome this, prior works in [27] and [28] first convert the input range of the natural logarithm by scaling the input, and then compute the Taylor series to approximate the natural logarithm in the ciphertext-domain. However, these prior works use either heavy homomorphic encryption or garbled circuit to implement secure computation. Both the technologies are time-consuming and far from satisfying the efficiency requirements of our task, as discussed in Section III-C-Q1.

We also follow this computation flow but design a more efficient secure computation method $SecLn(\cdot)$. Firstly, we can represent any $x \in \mathbb{R}^+$ as

$$x = 2^n(1 + \varepsilon), \tag{5}$$

where $n = \lfloor \log_2(x) + 1 \rfloor$ and $\varepsilon = -1 + x/2^n$. Then the natural logarithm $\ln(x)$ can be converted as

$$\ln(x) = \ln(2^n(1 + \varepsilon)) = n \ln(2) + \ln(1 + \varepsilon), \tag{6}$$

where n satisfies that $2^{n-1} \leq x < 2^n$ and $\varepsilon \in [-\frac{1}{2}, 0)$. Then the computation of $\ln(x)$ can be decomposed into the computations of $n \ln(2)$ and $\ln(1 + \varepsilon)$. Since $n \ln(2)$ can be directly computed in the secret sharing domain, we only need to compute $\ln(1 + \varepsilon)$. Because $\varepsilon \in [-\frac{1}{2}, 0)$, $\ln(1 + \varepsilon)$ can be directly approximated using Taylor series with a minor error, which is represented as

$$\ln(1 + \varepsilon) = \sum_{i=1}^{\infty} \frac{(-1)^{i-1}\varepsilon^i}{i} = \varepsilon - \frac{\varepsilon^2}{2} + \frac{\varepsilon^3}{3} - \frac{\varepsilon^4}{4} + \cdots \tag{7}$$

As a result, our $SecLn(\cdot)$ can be divided into two stages. (1) Compute sharings $\langle n \rangle$ and $\langle \varepsilon \rangle$ for input sharing $\langle x \rangle$; (2) Compute $\langle n \rangle \cdot \ln(2)$ and the Taylor series of $\ln(1 + \varepsilon)$ in the secret sharing domain, and add the two results to obtain the natural logarithm result.

a) The first stage of $SecLn(\cdot)$: This stage computes sharings $\langle n \rangle$ and $\langle \varepsilon \rangle$ for input sharing $\langle x \rangle$. For clear description, we suppose that the input x (i.e., $M_{j,k}$) of the natural logarithm is within the range $x \in [2^{\tau-1}, 2^{\omega})$, where ω is a positive integer and τ is a negative one. Because a word pair in users' corpus has countable co-occurrences, the input x has a upper bound and thus ω can be empirically determined. We can also evaluate the lower bound of x empirically to determine τ. Note that our method still works when $x < 2^{\tau-1}$, which will be discussed in the following **Remark** part.

We set a control bit c_i ($i \in [\tau, \omega]$) for each possible value of n, where $c_n = 1$ and $c_i = 0$ ($i \neq n$). Then n and $\frac{1}{2^n}$ can be computed with these control bits as

$$n = \sum_{i=\tau}^{\omega} c_i \cdot i \quad \text{and} \quad \frac{1}{2^n} = \sum_{i=\tau}^{\omega} c_i \cdot \frac{1}{2^i}. \tag{8}$$

According to Eq. (5), ε can be computed as

$$\varepsilon = -1 + x \cdot \left(\sum_{i=\tau}^{\omega} c_i \cdot \frac{1}{2^i} \right) \tag{9}$$

The computations of n and ε are thereby converted to determine the c_i. We use ρ_i to denote the comparison result of x and 2^i. Specifically, $\rho_i = 1$ if $x \geq 2^i$; otherwise $\rho_i = 0$. Recall that n satisfies that $2^{n-1} \leq x < 2^n$. Because i and n are integers, $n > i$ is equal to $n - 1 \geq i$. For all $i < n$, we can get that $x \geq 2^{n-1} \geq 2^i$, which means that $\rho_i = 1$. For all $i \geq n$, we can get that $x < 2^n \leq 2^i$, which means that $\rho_i = 0$. It is obvious that for pairs (ρ_{i-1}, ρ_i) ($i \in [\tau, \omega]$), only $\rho_{n-1} \oplus \rho_n = 1$, and $\rho_{i-1} \oplus \rho_i = 0$ ($i \neq n$). Thus we can assign c_i as $c_i = \rho_{i-1} \oplus \rho_i$.

Algorithm 1 Secure Natural Logarithm $SecLn(\cdot)$

Input: $\langle x \rangle \in \mathbb{Z}_{2^\ell}$.
Output: $\langle \ln(x) \rangle \in \mathbb{Z}_{2^\ell}$.
The first stage of computing $SecLn(\cdot)$:
for $i = \tau$ to $\omega - 1$ **do**
  $CS_{(1)}$ and $CS_{(2)}$ compute $[\![\rho_i]\!] = SecComp(\langle x \rangle, 2^i)$.
$CS_{(1)}$ and $CS_{(2)}$ set $[\![\rho_{\tau-1}]\!] = [\![1]\!]$, $[\![\rho_\omega]\!] = [\![0]\!]$.
for $i = \tau$ to ω **do**
  Each $CS_{(b)}$ ($b \in \{1, 2\}$) computes $[\![c_i]\!]^{(b)} = [\![\rho_{i-1}]\!]^{(b)} \oplus [\![\rho_i]\!]^{(b)}$.
for $i = \tau$ to ω **do**
  $CS_{(1)}$ and $CS_{(2)}$ compute $\langle c_i \rangle = B2A([\![c_i]\!])$.
Each $CS_{(b)}$ computes $\langle n \rangle^{(b)} = \sum_{i=\tau}^{\omega} \langle c_i \rangle^{(b)} \cdot i$.
$CS_{(1)}$ and $CS_{(2)}$ compute $\langle \varepsilon \rangle = -1 + \langle x \rangle \cdot \sum_{i=\tau}^{\omega} \langle c_i \rangle \cdot \frac{1}{2^i}$.
The second stage of computing $SecLn(\cdot)$:
$CS_{(1)}$ and $CS_{(2)}$ compute $\langle T_\theta(\varepsilon) \rangle$.
Each $CS_{(b)}$ computes $\langle \ln(x) \rangle^{(b)} = \langle n \rangle^{(b)} \cdot \ln(2) + \langle T_\theta(\varepsilon) \rangle^{(b)}$.

In our design, the cloud servers $CS_{(1)}$ and $CS_{(2)}$ first compute the Boolean sharing $[\![c_i]\!]$ and then convert $[\![c_i]\!]$ into arithmetic sharing $\langle c_i \rangle$. Let $SecComp(\langle x \rangle, y)$ denote secure comparison function that takes secret sharing $\langle x \rangle$ and public value y as input, and outputs Boolean sharing $[\![1]\!]$ if $x \geq y$, and $[\![0]\!]$ otherwise. $CS_{(1)}$ and $CS_{(2)}$ collaboratively calculate $[\![c_i]\!]$ in two steps. Firstly, they use function $SecComp$ to calculate $[\![\rho_i]\!] = SecComp(\langle x \rangle, 2^i)$ ($i \in [\tau - 1, \omega]$). Secondly, each $CS_{(b)}$ ($b \in \{1, 2\}$) calculates $[\![c_i]\!]^{(b)} = [\![\rho_{i-1}]\!]^{(b)} \oplus [\![\rho_i]\!]^{(b)}$ ($i \in [\tau, \omega]$) locally. The implementation of $SecComp$ is based on the idea of secure bit decomposition. Concretely, the comparison of two fixed-point integers $X, Y \in \mathbb{Z}_{2^\ell}$ under binary complement representation can be converted as the comparison of $X - Y$ and 0. Then the most significant bit (MSB) can be used to represent the relationship between $X - Y$ and 0. The $msb(X - Y)$ is 1 if $X - Y < 0$; otherwise it is 0. We use the secure parallel prefix adder (PPA) introduced in [7] to extract the MSB of the given arithmetic sharing $\langle X - Y \rangle$ and outputs the Boolean sharing $[\![msb(X - Y)]\!]$ in $\mathbb{Z}_2$. Fig. 3 shows an example of extracting the MSB using an 8-bit PPA. The secure PPA only relies on XOR and AND operations in the secret sharing domain and can overcome the serial execution restriction of the full adder, resulting in fewer communication rounds. By this design, $CS_{(1)}$ and $CS_{(2)}$ can obtain $[\![msb(x - 2^i)]\!]$. Specifically, they obtain $[\![0]\!]$ if $x \geq 2^i$ and obtain $[\![1]\!]$ if $x < 2^i$. However, we need $[\![1]\!]$ for $x \geq 2^i$ and $[\![0]\!]$ for otherwise. To achieve this, we let $CS_{(1)}$ negate its result locally. We describe the secure comparison $[\![\rho_i]\!] = SecComp(\langle x \rangle, 2^i)$ as follows:

- $CS_{(1)}$ and $CS_{(2)}$ collaboratively extract the Boolean sharing $[\![\rho_i]\!] = [\![msb(x - 2^i)]\!]$ using secure PPA.
- $CS_{(1)}$ reverses its result $[\![\rho_i]\!]^{(1)}$ to $1 \oplus [\![\rho_i]\!]^{(1)}$.

Since $2^{\tau-1} \leq x < 2^\omega$, $\rho_{\tau-1} = 1$ and $\rho_\omega = 0$ are always true. According to this, we can let $CS_{(1)}$ and $CS_{(2)}$ collaboratively set $[\![\rho_{\tau-1}]\!] = [\![1]\!]$ and $[\![\rho_\omega]\!] = [\![0]\!]$. In this way, we only need to preform the secure comparison for $i \in [\tau, \omega - 1]$. After the secure comparison, each $CS_{(b)}$ computes $[\![c_i]\!]^{(b)} = [\![\rho_{i-1}]\!]^{(b)} \oplus [\![\rho_i]\!]^{(b)}$ ($i \in [\tau, \omega]$) locally to finish the computation of $[\![c_i]\!]$.

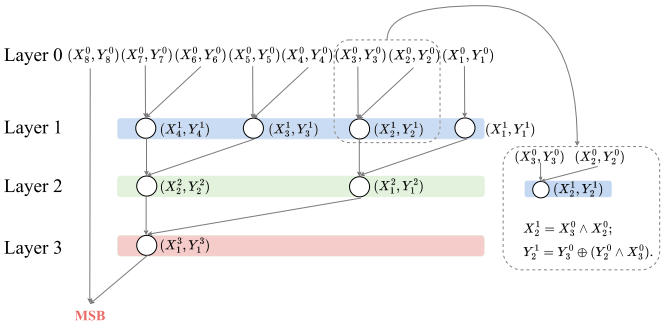


Fig. 3. An example of extracting the MSB using an 8-bit PPA.

After obtaining the Boolean shares $[\![c_i]\!]^{(1)}$ and $[\![c_i]\!]^{(2)}$, $CS_{(1)}$ and $CS_{(2)}$ collaboratively convert them into arithmetic shares $\langle c_i \rangle^{(1)}$ and $\langle c_i \rangle^{(2)}$ and use these values to compute $\langle n \rangle$ and $\langle \varepsilon \rangle$. We adopt the secure $B2A$ function from [33] to achieve this conversion. Specifically, given a secret value x, $B2A$ function can convert its Boolean sharing $[\![x]\!]$ in $\mathbb{Z}_2$ to the arithmetic sharing $\langle x \rangle$ in $\mathbb{Z}_{2^\ell}$. Given two cloud servers $CS_{(1)}$ and $CS_{(2)}$, the secure $B2A([\![x]\!])$ is performed as follows:

- $CS_{(1)}$ sets $\langle u \rangle^{(1)} = [\![x]\!]^{(1)}$, $\langle h \rangle^{(1)} = 0$.
- $CS_{(2)}$ sets $\langle u \rangle^{(2)} = 0$, $\langle h \rangle^{(2)} = [\![x]\!]^{(2)}$.
- $CS_{(1)}$ and $CS_{(2)}$ collaboratively compute $\langle x \rangle = \langle u \rangle + \langle h \rangle - 2 \cdot \langle u \rangle \cdot \langle h \rangle$.

After obtaining $\langle c_i \rangle$, $CS_{(1)}$ and $CS_{(2)}$ calculate $\langle n \rangle$ and $\langle \varepsilon \rangle$ in the secret sharing domain following Eq. (8) and Eq. (9). Note that $CS_{(1)}$ and $CS_{(2)}$ compute $\langle n \rangle$ locally, but compute $\langle \varepsilon \rangle$ collaboratively because multiplication over arithmetic sharing is involved.

b) The second stage of $SecLn(\cdot)$: In the second stage, $CS_{(1)}$ and $CS_{(2)}$ compute $\langle n \rangle \cdot \ln(2)$ and use Taylor series to approximate $\ln(1 + \varepsilon)$ in the secret sharing domain. The computation of $\langle n \rangle \cdot \ln(2)$ can be performed by each cloud server locally. We use $T_\theta(\cdot)$ to denote the computation of the Taylor series defined in Eq. (7), where θ is the number of used terms. To compute $T_\theta(\cdot)$ in the secret sharing domain, $CS_{(1)}$ and $CS_{(2)}$ first agree upon the parameter θ and then collaboratively compute $T_\theta(\cdot)$ with the input $\langle \varepsilon \rangle$ by executing the addition and multiplication operations in the secret sharing domain. After the computations, each $CS_{(b)}$ ($b \in \{1, 2\}$) gets the share $\langle T_\theta(\varepsilon) \rangle^{(b)}$. Finally, each $CS_{(b)}$ outputs $\langle \ln(x) \rangle^{(b)} = \langle n \rangle^{(b)} \cdot \ln(2) + \langle T_\theta(\varepsilon) \rangle^{(b)}$ to finish the second phase of $SecLn(\cdot)$. Algorithm 1 shows the whole steps of our secure natural logarithm $SecLn(\cdot)$.

Remark: To better describe our design, we suppose that the input x is larger than a fixed small value, namely $x \geq 2^{\tau-1}$. However, our $SecLn(\cdot)$ can still work when the input $x < 2^{\tau-1}$. Following the aforementioned computation steps, we can obtain that $n = \tau$ when $x < 2^{\tau-1}$. Since $\varepsilon = -1 + x/2^n$, we then get that $\varepsilon \in (-1, -\frac{1}{2})$, namely $(1 + \varepsilon) \in (0, \frac{1}{2})$. As a result, when x is a little smaller than $2^{\tau-1}$, we can obtain that ε is a little smaller than $-\frac{1}{2}$ and the Taylor series of $\ln(1 + \varepsilon)$ can still achieve a close approximation with small error. However, the error will increase with the decrease of x.

2) Secure Weighting Function: In PPGloVe, we should securely compute the weighting function $f(x)$ defined in

Algorithm 2 Secure Weighting Function $SecW(\cdot)$

Input: $\langle x\rangle \in \mathbb{Z}_{2^\ell}$.
Output: $\langle f(x)\rangle \in \mathbb{Z}_{2^\ell}$.
Calculate the control bit:
1: $CS_{(1)}$ and $CS_{(2)}$ compute the Boolean sharing $[\![t_1]\!] = SecComp(\langle x\rangle, x_{max})$.
2: $CS_{(1)}$ computes $[\![t_2]\!]^{(1)} = 1 \oplus [\![t_1]\!]^{(1)}$.
3: $CS_{(2)}$ computes $[\![t_2]\!]^{(2)} = [\![t_1]\!]^{(2)}$.
Convert the Boolean sharing to arithmetic sharing:
4: $CS_{(1)}$ and $CS_{(2)}$ compute $\langle t_1\rangle = B2A([\![t_1]\!])$ and $\langle t_2\rangle = B2A([\![t_2]\!])$.
Calculate Eq. (10) in the secret sharing domain:
5: Each $CS_{(b)}$ $(b \in \{1,2\})$ computes $\langle x\rangle^{(b)}/x_{max}$.
6: $CS_{(1)}$ and $CS_{(2)}$ compute $\langle f(x)\rangle = 1\cdot\langle t_1\rangle + \langle x\rangle/x_{max}\cdot\langle t_2\rangle$.

Eq. (3) in the secret sharing domain. The power exponent α in $f(x)$ is set to 1 in our design since it has few impact to the training result. Then the function $f(x)$ can be converted as two polynomials that are triggered by two opposite control bits t_1 and t_2. Suppose that $t_1 = 0$ and $t_2 = 1$ if $x < x_{max}$; otherwise, $t_1 = 1$ and $t_2 = 0$. Then $f(x)$ can be rewritten as

$$f(x) = 1 \cdot t_1 + (x/x_{max}) \cdot t_2. \tag{10}$$

Since $t_2 = \neg t_1$, the key point to implement the secure weighting function $SecW(\cdot)$ is to compute $\langle t_1\rangle$ and $\langle x\rangle/x_{max}$. The two cloud servers $CS_{(1)}$ and $CS_{(2)}$ perform the following steps to obtain the secret sharing of $f(x)$.

- $CS_{(1)}$ and $CS_{(2)}$ use the function $SecComp$ introduced in Section IV-C.1 to compute the Boolean sharing $[\![t_1]\!] = SecComp(\langle x\rangle, x_{max})$.
- $CS_{(1)}$ negates $[\![t_1]\!]^{(1)}$ to obtain $[\![t_2]\!]^{(1)} = 1 \oplus [\![t_1]\!]^{(1)}$ and $CS_{(2)}$ sets $[\![t_2]\!]^{(2)} = [\![t_1]\!]^{(2)}$, implementing $[\![t_2]\!] = [\![\neg t_1]\!]$.
- $CS_{(1)}$ and $CS_{(2)}$ convert the Boolean sharing $[\![t_1]\!]$, $[\![t_2]\!]$ into the arithmetic sharing $\langle t_1\rangle$ and $\langle t_2\rangle$ using the secure $B2A$ function introduced in Section IV-C.1.
- Each $CS_{(b)}$ $(b \in \{1,2\})$ computes $\langle x\rangle^{(b)}/x_{max}$ locally.
- $CS_{(1)}$ and $CS_{(2)}$ obtain the sharing of $f(x)$ by computing $\langle f(x)\rangle = 1 \cdot \langle t_1\rangle + \langle x\rangle/x_{max} \cdot \langle t_2\rangle$.

Algorithm 2 presents the construction for secure weighting function $SecW(\cdot)$.

D. Secure Word Vectors Training

During the training process, the cloud servers $CS_{(1)}$ and $CS_{(2)}$ use the tokens of words to index the word vectors and biases without knowing the original words. They compute the gradient values and update the word vectors v_j, $\tilde{v}_k$ and their biases d_j, $\tilde{d}_k$ in the secret sharing domain. Then, each $CS_{(b)}$ $(b \in \{1,2\})$ returns a share $\langle \mathbf{V}\rangle^{(b)}$ of the trained word vectors $\mathbf{V}$ to the user $\mathcal{U}_i$, where $\mathbf{V}$ is the set of all the v_j and $\tilde{v}_k$. Finally, $\mathcal{U}_i$ restores $\mathbf{V}$ using $\mathbf{V} = \langle \mathbf{V}\rangle^{(1)} + \langle \mathbf{V}\rangle^{(2)}$ and recovers the related words using PRF and his/her kept key locally.

The parameters $\langle v_j\rangle$, $\langle \tilde{v}_k\rangle$, $\langle d_j\rangle$, and $\langle \tilde{d}_k\rangle$ are learned upon each training sample $(\pi_j, \pi_k, \langle M_{j,k}\rangle)$, where π_j and π_k are the tokens of the words j and k, respectively. We use the stochastic gradient descent algorithm to learn the parameters. To obtain the word vectors updating equations, we first calculate the derivatives of the loss function $\boldsymbol{J}$ (shown in Eq. (4)) regarding to v_j and $\tilde{v}_k$ in the plaintext-domain as follows

$$g_{v_j} = \frac{\partial \boldsymbol{J}}{\partial v_j} = \tilde{v}_k f\left(M_{j,k}\right)\left(v_j^T \tilde{v}_k + d_j + \tilde{d}_k - \ln(M_{j,k})\right), \tag{11}$$

$$g_{\tilde{v}_k} = \frac{\partial \boldsymbol{J}}{\partial \tilde{v}_k} = v_j f\left(M_{j,k}\right)\left(v_j^T \tilde{v}_k + d_j + \tilde{d}_k - \ln(M_{j,k})\right). \tag{12}$$

Then the updating equations for word vectors v_j and $\tilde{v}_k$ can be obtained as

$$v_j = v_j - \eta g_{v_j} \quad \text{and} \quad \tilde{v}_k = \tilde{v}_k - \eta g_{\tilde{v}_k}, \tag{13}$$

where η denotes a positive learning rate. Similarly, we can get the gradients and updating equations for biases d_j and $\tilde{d}_k$ as

$$g_{d_j} = g_{\tilde{d}_k} = f\left(M_{j,k}\right)\left(v_j^T \tilde{v}_k + d_j + \tilde{d}_k - \ln(M_{j,k})\right), \tag{14}$$

$$d_j = d_j - \eta g_{d_j} \quad \text{and} \quad \tilde{d}_k = \tilde{d}_k - \eta g_{\tilde{d}_k}. \tag{15}$$

After obtaining the updating equations, $CS_{(1)}$ and $CS_{(2)}$ can train the word vectors in the secret sharing domain as follows.

- $CS_{(1)}$ and $CS_{(2)}$ collaboratively calculate the gradients for word vectors v_j and $\tilde{v}_k$ in the secret sharing domain.
- Each $CS_{(b)}$ $(b \in \{1,2\})$ uses the intermediate result obtained in the previous step to calculate the gradients for biases b_j and $\tilde{b}_k$ in the secret sharing domain locally.
- Each $CS_{(b)}$ updates the four parameters in the secret sharing domain locally according to their updating equations to complete secure word vectors updating.

Algorithm 3 presents the whole operations of secure word vectors updating for one training sample in secret sharing domain and we use symbol $\odot$ to denote the inner product of two secret-shared vectors. All the training tasks can be performed with parallel processing by dividing the training set into multiple sub-sets and processing simultaneously.

1) Dynamic Adjustment of the Learning Rate: The learning rate η is a critical hyperparameter in the training process and it should be dynamically adjusted to implement self-adaptation. In GloVe, the Adagrad optimizer [39] is used to dynamically adjust the learning rate. However, this optimizer involves division and square root operations, which can not be directly performed in the secret sharing domain.

To adapt to our task, we use an efficient strategy [43] to update the learning rate η as follows

$$\eta = \eta_0\left(1 - \frac{used\_samples}{total\_samples}\right), \tag{16}$$

where η_0 is the initial value of the learning rate, $used\_samples$ is the number of the currently used training samples, and $total\_samples$ represents the total number of the training samples. At the first beginning of training, $\eta = \eta_0$ and the model has a faster convergence rate. As the training progresses, the word vector parameters are getting closer to the optimal values and η becomes small. So the fine-tuning of the parameters can be achieved. Note that this learning rate adjustment strategy utilizes only the number of the training samples, which is no need to be protected. Thus, the learning rate updating is performed in the plaintext-domain and does

Algorithm 3 Secure Word Vectors Updating

Input: A training sample $(\pi_j, \pi_k, \langle M_{j,k}\rangle)$.
Output: Updated secret-shared word vectors $\langle v_j\rangle$, $\langle \tilde{v}_k\rangle$ and updated secret-shared biases $\langle d_j\rangle$, $\langle \tilde{d}_k\rangle$.

Calculate the sharing of gradients for word vectors:

1: $CS_{(1)}$ and $CS_{(2)}$ compute $\langle G\rangle = \langle f(M_{j,k})\rangle \cdot (\langle v_j^T\rangle \odot \langle \tilde{v}_k\rangle + \langle d_j\rangle + \langle \tilde{d}_k\rangle - \langle \ln(M_{j,k})\rangle)$.
2: $CS_{(1)}$ and $CS_{(2)}$ compute $\langle g_{v_j}\rangle = \langle \tilde{v}_k\rangle \cdot \langle G\rangle$.
3: $CS_{(1)}$ and $CS_{(2)}$ compute $\langle g_{\tilde{v}_k}\rangle = \langle v_j\rangle \cdot \langle G\rangle$.

Calculate the shares of gradients for biases:

4: Each $CS_{(b)}$ $(b \in \{1,2\})$ sets local shares $\langle g_{d_j}\rangle^{(b)} = \langle G\rangle^{(b)}$ and $\langle g_{\tilde{d}_k}\rangle^{(b)} = \langle G\rangle^{(b)}$.

Update the shares of word vectors:

5: Each $CS_{(b)}$ updates $\langle v_j\rangle^{(b)} = \langle v_j\rangle^{(b)} - \eta \cdot \langle g_{v_j}\rangle^{(b)}$.
6: Each $CS_{(b)}$ updates $\langle \tilde{v}_k\rangle^{(b)} = \langle \tilde{v}_k\rangle^{(b)} - \eta \cdot \langle g_{\tilde{v}_k}\rangle^{(b)}$.

Update the shares of biases:

7: Each $CS_{(b)}$ updates $\langle d_j\rangle^{(b)} = \langle d_j\rangle^{(b)} - \eta \cdot \langle g_{d_j}\rangle^{(b)}$.
8: Each $CS_{(b)}$ updates $\langle \tilde{d}_k\rangle^{(b)} = \langle \tilde{d}_k\rangle^{(b)} - \eta \cdot \langle g_{\tilde{d}_k}\rangle^{(b)}$.

not introduce additional communication overhead. Note that our implementation of GloVe has replaced the learning rate dynamic adjustment mechanism and modified the value of α in the weighting function $f(x)$. Thus, we use the adapted-GloVe to represent the modified model in this paper.

2) Optimization: When computing the gradients $\langle g_{v_j}\rangle$ and $\langle g_{\tilde{v}_k}\rangle$ (lines 2-3 of Algorithm 3), we should compute the multiplications of two m-dimensional secret-shared vectors $\langle \tilde{v}_k\rangle$ and $\langle v_j\rangle$ with a value $\langle G\rangle$ in the secret sharing domain, namely $\langle \tilde{v}_k\rangle \cdot \langle G\rangle$ and $\langle v_j\rangle \cdot \langle G\rangle$. This indicates that each element in the two vectors needs to be multiplied by $\langle G\rangle$. A straightforward approach is to convert these vector-element multiplications into $2m$ independent element-element multiplications, using $2m$ independent Beaver triples. Then the communication cost is $8m\ell$ bits, according to the discussion in Section II-B.1.

Since the above $2m$ multiplications have the same multiplier $\langle G\rangle$ that is masked multiple times when using independent Beaver triples, we can integrate the computations of $\langle g_{v_j}\rangle$ and $\langle g_{\tilde{v}_k}\rangle$ into one protocol and optimize the computations following the prior work [42]. Specifically, we replace the $2m$ independent Beaver triples with $2m$ correlated Beaver triples $(\langle l\rangle, \langle o_1\rangle, \cdots, \langle o_{2m}\rangle, \langle p_1\rangle, \cdots, \langle p_{2m}\rangle)$, where $p_i = l \cdot o_i$ $(i \in [1, 2m])$. The sharing $\langle l\rangle$ is used to mask $\langle G\rangle$, and $\langle o_i\rangle$ is used to mask the elements in the secret-shared vectors. In this way, the constant multiplier $\langle G\rangle$ needs to be masked only once. Note that these correlated Beaver triples can be pre-generated using the same way as the standard triples [42].

Algorithm 4 shows the secure vector-element multiplication protocol using correlated Beaver triples. We use the symbol $v(i)$ to denote the i-th element in vector v. Since β, $\{\delta_i\}$, and $\{\delta_{m+i}\}$ $(i \in [1, m])$ can be reconstructed in one communication round, $CS_{(1)}$ and $CS_{(2)}$ can execute this protocol by communicating once. The communication overhead for reconstructing β is 2ℓ bits, and that for reconstructing $\{\delta_i\}$ and $\{\delta_{m+i}\}$ $(i \in [1, m])$ is $4m\ell$ bits. Thus, the total communication overhead is $2\ell + 4m\ell$ bits, which is nearly cut in half compared to $8m\ell$ bits in the original computations.

Algorithm 4 Secure Vector-Element Multiplication

Input: Two m-dimensional secret-shared vectors $\langle \tilde{v}_k\rangle$ and $\langle v_j\rangle$, a secret-shared element $\langle G\rangle$, and correlated Beaver triples $(\langle l\rangle, \langle o_1\rangle, \cdots, \langle o_{2m}\rangle, \langle p_1\rangle, \cdots, \langle p_{2m}\rangle)$.
Output: Secret-shared vectors $\langle \tilde{v}_k \cdot G\rangle$ and $\langle v_j \cdot G\rangle$.

1: Each $CS_{(b)}$ $(b \in \{1,2\})$ computes $\langle \beta\rangle^{(b)} = \langle G\rangle^{(b)} - \langle l\rangle^{(b)}$.
2: $CS_{(1)}$ and $CS_{(2)}$ reconstruct β.
3: **for** $i \in [1, m]$ **do**
4: Each $CS_{(b)}$ computes $\langle \delta_i\rangle^{(b)} = \langle \tilde{v}_k(i)\rangle^{(b)} - \langle o_i\rangle^{(b)}$.
5: Each $CS_{(b)}$ computes $\langle \delta_{m+i}\rangle^{(b)} = \langle v_j(i)\rangle^{(b)} - \langle o_{m+i}\rangle^{(b)}$.
6: $CS_{(1)}$ and $CS_{(2)}$ reconstruct δ_i, δ_{m+i}.
7: Each $CS_{(b)}$ computes $\langle \tilde{v}_k(i) \cdot G\rangle^{(b)} = (b-1) \cdot \beta \cdot \delta_i + \beta \cdot \langle o_i\rangle^{(b)} + \langle l\rangle^{(b)} \cdot \delta_i + \langle p_i\rangle^{(b)}$.
8: Each $CS_{(b)}$ computes $\langle v_j(i) \cdot G\rangle^{(b)} = (b-1) \cdot \beta \cdot \delta_{m+i} + \beta \cdot \langle o_{m+i}\rangle^{(b)} + \langle l\rangle^{(b)} \cdot \delta_{m+i} + \langle p_{m+i}\rangle^{(b)}$.

V. Complexity Analysis and Security Analysis

A. Complexity Analysis

Since our PPGloVe involves only a limited number of additions and multiplications that have low computation cost, we focus on analyzing the communication complexity of our design. Our PPGloVe contains three fundamental building blocks: secure natural logarithm $SecLn(\cdot)$, secure weighting function $SecW(\cdot)$, and secure word vectors updating.

1) Complexity of $SecLn(\cdot)$: The $SecLn(\cdot)$ contains $\omega - \tau$ $SecComp$ operations, $\omega - \tau + 1$ $B2A$ operations, one computation for $\langle \epsilon\rangle$, one computation for $\langle n\rangle$ and one computation for the sharing $\langle T_\theta(\epsilon)\rangle$ of Taylor series. The communication overhead and communication round for performing one secure comparison function $SecComp$ on ℓ-bit secret sharing are $12\ell - 16$ bits and $\log(\ell) + 1$, respectively [35]. Thus, the communication overhead for $SecComp$ in $SecLn(\cdot)$ is $(\omega - \tau)(12\ell - 16)$ bits. Because the $\omega - \tau$ $SecComp$ operations are independent and can be performed with parallel processing, the communication round for these $\omega - \tau$ $SecComp$ operations is still $\log(\ell) + 1$. Each $B2A$ operation has only one secure multiplication operation that requires interaction. So the communication cost is 4ℓ bits and the communication round is one. Since these $\omega - \tau + 1$ $B2A$ operations can also be computed with parallel processing, the communication cost caused by $B2A$ operations is $4(\omega - \tau + 1)\ell$ bits, and the communication round is one. The computation for $\langle \varepsilon\rangle$ requires one secure multiplication with 4ℓ bits of communication overhead and one communication round. The computation for $\langle n\rangle$ can be performed locally with no communication overhead. The computation for $\langle T_\theta(\epsilon)\rangle$ requires $\lceil \log(\theta)\rceil$ communication rounds. For example, the communication round is 3 when $\theta = 5$. Specifically, $\langle \epsilon^2\rangle$ can be calculated in the first communication round; $\langle \epsilon^3\rangle$ and $\langle \epsilon^4\rangle$ can be calculated in the second communication round; and $\langle \epsilon^5\rangle$ can be calculated in the third communication round. Since the calculation for $\langle T_\theta(\epsilon)\rangle$ requires $\theta - 1$ multiplications in the secret sharing domain, its communication overhead is $4(\theta - 1)\ell$ bits. As a result, the communication overhead of the $SecLn(\cdot)$ is $(\omega - \tau)(12\ell - 16) + 4(\omega - \tau + 1)\ell + 4\theta\ell$ bits, and its communication round is $3 + \log(\ell) + \lceil \log(\theta)\rceil$.

2) Complexity of $SecW(\cdot)$: The $SecW(\cdot)$ includes one $SecComp$ operation, two $B2A$ operations, and one multiplication operation in the secret sharing domain. Thus its communication overhead is $24\ell - 16$ bits and its communication round is $\log(\ell) + 3$.

3) Complexity of Secure Word Vectors Updating: The secure word vectors updating contains one secure multiplication, one secure m-dimensional vector inner product and computations of $\langle g_{v_j} \rangle$ and $\langle g_{\tilde{v}_k} \rangle$. The communication overhead and communication round for one secure multiplication is 4ℓ bits and one. The secure m-dimensional vector inner product can be divided as m independent secure multiplications and $m - 1$ additions. Then its communication overhead is $4m\ell$ bits and communication round is one. The computations of $\langle g_{v_j} \rangle = \langle \tilde{v}_k \rangle \cdot \langle G \rangle$ and $\langle g_{\tilde{v}_k} \rangle = \langle v_j \rangle \cdot \langle G \rangle$ are implemented by the secure vector-element multiplication protocol shown in Algorithm 4. Then the communication overhead is $2\ell + 4m\ell$ bits and the communication round is one, according to the discussion in Section IV-D.2. As a result, the communication overhead of the secure word vectors updating is $(8m + 6)\ell$ bits and its communication round is 3.

B. Security Analysis

We now analyze the security of PPGloVe. As per the specification of our proposed PPGloVe model, we replace the words in the users' co-occurrence matrices $\{\mathbf{M}_i\}$ with secure tokens generated by PRF and encrypt $\{\mathbf{M}_i\}$ and word vectors $\mathbf{V}$ as secret shares based on the secret sharing technique. The interactions between the two non-colluding cloud servers are supported by the Beaver triples that can provide provable security guarantees. During the execution of PPGloVe, the two non-colluding cloud servers receive only their corresponding secret shares that are uniformly distributed and reveal nothing about the private data. Thus the privacy of the user data and word vectors is preserved.

We follow the standard simulation-based paradigm to prove the security, as in recent works on privacy-preserving machine learning [6], [13], [35], [40], [44]. Formally, we first define an ideal functionality $\mathcal{F}$ to capture the security requirements PPGloVe aims to achieve.

- **Input.** The users submit the matrices $\{\mathbf{M}_i\}$ to $\mathcal{F}$.
- **Computation.** After receiving $\{\mathbf{M}_i\}$, $\mathcal{F}$ merges these matrices and adopts the adapted-GloVe model to train word vectors.
- **Output.** $\mathcal{F}$ returns the trained word vectors $\mathbf{V}$ to users.

Let Π denote the protocol that realizes the ideal functionality $\mathcal{F}$. We formally define the security of Π as in Definition 1 and take the following Lemma 1 [44] and Lemma 2 [6] to simplify the proofs.

Definition 1: Let $view_{(b)}^{\Pi}$ denote each $CS_{(b)}$'s ($b \in \{1, 2\}$) view during the execution of Π. Π is secure in the semi-honest and non-colluding setting, if for each $CS_{(b)}$ there exists a probabilistic polynomial-time simulator such that $S_{(b)}^{\Pi} \overset{c}{\equiv} view_{(b)}^{\Pi}$, where $S_{(b)}^{\Pi}$ is a simulated view of $CS_{(b)}$ generated by the simulator, and $\overset{c}{\equiv}$ means indistinguishable.

Lemma 1: The whole system is simulatable if all of its building blocks are simulatable.

Lemma 2: The operations SecMul and SecAdd are simulatable in the semi-honest and non-colluding setting.

Theorem 1: PPGloVe can securely realize the ideal functionality $\mathcal{F}$ in the semi-honest and non-colluding setting according to Definition 1.

Proof: Note that PPGloVe consists of several subroutines: system initialization (SI), secure comparison ($SecComp$), secure natural logarithm ($SecLn(\cdot)$), secure weighting function ($SecW(\cdot)$), and secure word vectors updating (SWV). According to Lemma 1, we can conclude that PPGloVe is secure if the simulator for each subroutine exits. We use $Sim_{\Delta}^{(b)}$ to represent the simulator which generates $CS_{(b)}$'s view in the execution of subroutine Δ. As the roles of $CS_{(1)}$ and $CS_{(2)}$ are symmetric in these protocols, it suffices to demonstrate the existence of simulators for $CS_{(1)}$ in these subroutines.

1) Existence of Simulator $Sim_{SI}^{(1)}$: During system initialization, the words in $\langle \mathbf{M}_i \rangle^{(1)}$ are replaced with secure tokens generated by PRF and $CS_{(1)}$ performs only $SecAdd$ operation to sum up the values of the same token-pairs in $\{\langle \mathbf{M}_1 \rangle^{(1)}, \langle \mathbf{M}_2 \rangle^{(1)}, \cdots, \langle \mathbf{M}_i \rangle^{(1)}, \cdots\}$. According to Lemma 2, the simulator $Sim_{SI}^{(1)}$ exists.

2) Existence of Simulator $Sim_{SecComp}^{(1)}$: Since the $SecComp$ function is implemented using a PPA in the secret sharing domain, which involves only secret sharing XOR (identical to $SecAdd$) and AND (identical to $SecMul$). According to Lemmas 1 and 2, the view of $CS_{(1)}$ in $SecComp$ is simulatable and the simulator $Sim_{SecComp}^{(1)}$ exits.

3) Existence of Simulator $Sim_{SecLn}^{(1)}$: The $SecLn(\cdot)$ contains $SecComp$, $B2A$, $SecMul$, and $SecAdd$ operations. $CS_{(1)}$ first performs $SecComp$ function for input $\langle x \rangle^{(1)}$. Since $SecComp$ has been proven to be simulatable, the output $\{[\![\rho_\tau]\!]^{(1)}, [\![\rho_{\tau+1}]\!]^{(1)}, \cdots, [\![\rho_{\omega-1}]\!]^{(1)}\}$ can be simulated. Then $CS_{(1)}$ performs $B2A$ function. As introduced in Section IV-C.1, the $B2A$ function contains only the $SecMul$ and $SecAdd$ operations. Thus, the simulator for $CS_{(1)}$ in $B2A$ also exists. So the output of $B2A$ $\{\langle c_\tau \rangle^{(1)}, \langle c_{\tau+1} \rangle^{(1)}, \cdots, \langle c_\omega \rangle^{(1)}\}$ can be simulated. Since other intermediate results in $SecLn(\cdot)$ can be computed using $SecMul$ and $SecAdd$ that have been proven to be simulatable, these intermediate results can also be simulated. As a result, the simulator $Sim_{SecLn}^{(1)}$ exists.

4) Existence of Simulator $Sim_{SecW}^{(1)}$: The $SecW(\cdot)$ also contains the $SecComp$, $B2A$, $SecMul$ and $SecAdd$ operations. Both the $SecComp$ and $B2A$ operations have been proven to be simulatable. So the output of $SecComp$, $[\![t_1]\!]^{(1)}$ and the output of $B2A$, $\{\langle t_1 \rangle^{(1)}, \langle t_2 \rangle^{(1)}\}$ can be simulated. Other intermediate results in $SecW(\cdot)$ can be computed using $SecMul$ and $SecAdd$, so these results can also be simulated. As a result, the simulator $Sim_{SecW}^{(1)}$ exists.

5) Existence of Simulator $Sim_{SWV}^{(1)}$: The SWV involves the optimized secure vector-element multiplication (Algorithm 4). As introduced in Section IV-D.2, we optimize the vector-element multiplication by replacing the $2m$ independent Beaver triples with $2m$ correlated Beaver triples ($\langle l \rangle$, $\langle o_1 \rangle$, $\cdots$, $\langle o_{2m} \rangle$, $\langle p_1 \rangle$, $\cdots$, $\langle p_{2m} \rangle$), where $p_i = l \cdot o_i$ ($i \in [1, 2m]$). The sharing $\langle l \rangle$ is used to mask the constant multiplier $\langle G \rangle$ and $\langle o_i \rangle$ is used to mask the element in the secret-shared vectors. Since the correlated Beaver triples can be generated based on the standard secret sharing, their shares received by $CS_{(1)}$ are randomly-distributed values. Thus the view of $CS_{(1)}$ on the correlated Beaver triples can be simulated. Then $CS_{(1)}$ receives the shares $\langle G - l \rangle^{(2)}$, $\{\langle \tilde{v}_k(i) - o_i \rangle^{(2)}\}$,

TABLE II
DATASET STATISTICS

Corpus Subset	File Size (MB)	Total Words	Dictionary Size
Dataset1	24	3,979,859	26,112
Dataset2	91	15,389,721	62,889
Dataset3	621	105,462,199	253,689

and $\{\langle v_j(i) - o_{m+i}\rangle^{(2)}\}$ from $CS_{(2)}$, and outputs the results $\{\langle \tilde{v}_k(i) \cdot G\rangle^{(1)}\}$, $\{\langle v_j(i) \cdot G\rangle^{(1)}\}$ $(i \in [1, m])$. Note that $\langle \tilde{v}_k(i) - o_i\rangle^{(2)}/\langle v_j(i) - o_{m+i}\rangle^{(2)}$ is a function of the correlated Beaver triples shares, $G - l$, and $\langle \tilde{v}_k(i) \cdot G\rangle^{(1)}/\langle v_j(i) \cdot G\rangle^{(1)}$. Therefore, these values are uniformly random and independent and their joint distributions in the real view and simulated view are identical [42]. So the simulator for $CS_{(1)}$ in the optimized secure vector-element multiplication exists. Since other intermediate results in SWV can be computed using $SecMul$ and $SecAdd$ that have been proven to be simulatable, the simulator $Sim^{(1)}_{SWV}$ exists, according to Lemmas 1 and 2.

The above concludes the proof of Theorem 1. ■

VI. Simulation Evaluation

A. Experimental Settings

We implement a prototype of our PPGloVe using C programming language and all experiments are run on a machine with 16 Intel(R) Xeon(R) Gold 6130 CPU @2.10GHz cores, 256GB RAM, and Ubuntu 16.04 OS.[1] In line with existing literatures [6], [7], [45] using similar privacy-preserving techniques, we also consider the local area network (LAN) scenarios for the communication between the two cloud servers in our PPGloVe. The network bandwidth and delay are set to 1 GB/s and 0.17 ms, respectively, following the settings in [6]. We use the file system to simulate the communication for LAN network scenarios. In our PPGloVe, the communication through the LAN occurs only between the two cloud servers during the model training.

PPGloVe works in secret sharing domain and we set the ring size as $\mathbb{Z}_{2^{128}}$ in our experiments. We set the scaling factor q as 22 to present a real value to 128-bit fixed-point integer. To implement $SecLn(\cdot)$, we set the term number θ in Taylor series as 16, and the parameters τ and ω as -4 and 22, respectively. We assume that the participation user number is 10, and all the users have almost the same size of data. The dimension of a word vector is set to $m = 100$ by default, and the window size in the model is set to 15. Besides, the number of training epochs is set as 10 and the number of threads during training is 50.

The public Wikimedia dump is widely used in word vectors training task [1], [46] and our experiment also uses it as our training corpus. More specifically, we generate three sub-datasets with different scales, which are listed in Table II.

B. Miscrobenchmarks

We firstly evaluate $SecLn(\cdot)$ and $SecW(\cdot)$, which are the two most important building blocks in PPGloVe. Since these two building blocks can be executed with parallel processing in data processing stage, we evaluate their performance with serial processing and parallel processing, respectively. Fig. 4(a) shows the running time of $SecLn(\cdot)$ and $SecW(\cdot)$ with different numbers of executions in our simulation. As can be seen, $SecLn(\cdot)$ and $SecW(\cdot)$ separately take about 2.4 ms and 1.5 ms to complete one execution with serial processing, and separately take about 0.015 ms and 0.001 ms with parallel processing. This indicates that $SecLn(\cdot)$ and $SecW(\cdot)$ have very fast execution speed, and their efficiency is much higher with parallel processing.

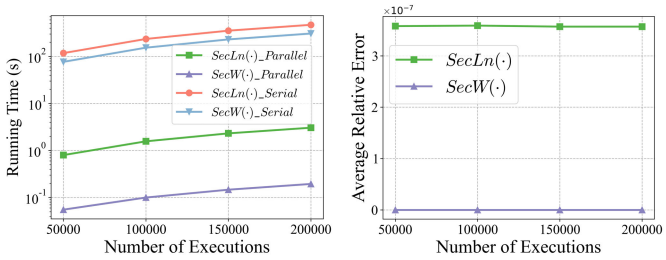


Fig. 4. Performance of the $SecLn(\cdot)$ and $SecW(\cdot)$. (a) The running time with serial processing and parallel processing; (b) the average relative errors with the plaintext-domain computations on randomly distributed data.

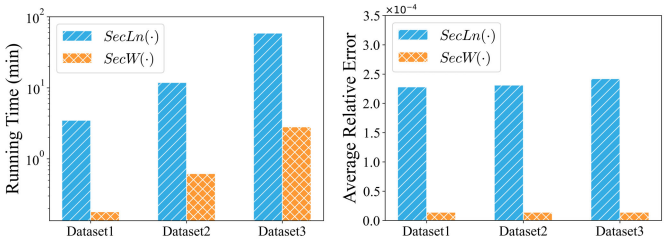
We also evaluate the accuracy of $SecLn(\cdot)$ and $SecW(\cdot)$ on randomly distributed data and calculate their average relative errors with the plaintext-domain computations. Fig. 4(b) demonstrates the average relative errors with different execution numbers. The relative error of $SecLn(\cdot)$ is 3.6×10^{-7} on average, and that of $SecW(\cdot)$ is 0. This indicates that $SecLn(\cdot)$ causes only an extremely small error and $SecW(\cdot)$ does not cause any error.

C. Performance of PPGloVe

1) Performance on Data Processing: We first evaluate the running time and error of the system in data processing stage, and show the results in Fig. 5. As can be observed in Fig. 5(a), the computation of $SecLn(\cdot)$ accounts for the most time at data processing stage. For example, it takes 58.7 minutes to compute $SecLn(\cdot)$ and 2.9 minutes to compute $SecW(\cdot)$ on Dataset3. These time cost is practically acceptable, indicating the efficiency of our design. Fig. 5(b) shows the average relative errors of the two secure computations compared to their plaintext-domain operations. The average relative errors of $SecLn(\cdot)$ and $SecW(\cdot)$ on the three datasets are about 2.3×10^{-4} and 1.4×10^{-5}, respectively. The relative errors are a little larger than that in miscrobenchmarks shown in Fig. 4(b) and the difference is caused by the different distributions of the test data. In real datasets, many elements $M_{j,k}$ in the matrix $\mathbf{M}$ are close to 1, causing that $\ln(M_{j,k})$ is close to 0.

We also compare the efficiency of our secure natural logarithm $SecLn(\cdot)$ with previous secure natural logarithm computation methods in [27] and [28]. We select these two works as our comparison baselines because they are the only secure natural logarithm computation methods that can support a wide range of inputs. Although comparing these secure natural logarithm computation methods utilizing different privacy-preserving techniques may not be entirely equitable, the results can demonstrate that previous works

[1]Our implementation is available at https://github.com/TTigerTT/PPGloVe



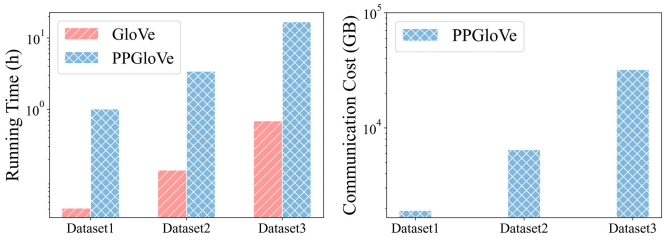
(a) (b)

Fig. 5. Performance of data processing stage. (a) The running time; (b) the average relative errors with the plaintext-domain computations.

TABLE III

TIME OF SECURELY COMPUTING ALL THE NATURAL LOGARITHMS IN OUR TASK USING DIFFERENT METHODS

Corpus Subset	Number of Executions	Total Time Cost		
		Lindell *et al.*'s Method [27]	Teo *et al.*'s Method [28]	Our $SecLn(\cdot)$
		Parallel Processing (20 threads)		
Dataset1	15,913,267	$\approx$ 1,800 days	$\approx$ 1,100 days	$\approx$ 4.1 mins
Dataset2	53,862,467	$\approx$ 6,200 days	$\approx$ 3,700 days	$\approx$ 13.8 mins
Dataset3	266,221,881	$\approx$ 30,800 days	$\approx$ 18,400 days	$\approx$ 69.5 mins
		Serial Processing		
Dataset1	15,913,267	-	$\approx$ 7,300 days	$\approx$ 0.5 days
Dataset2	53,862,467	-	$\approx$ 24,900 days	$\approx$ 1.5 days
Dataset3	266,221,881	-	$\approx$ 123,200 days	$\approx$ 7.4 days



(a) (b)

Fig. 6. Performance of word vectors training. (a) Running time; (b) communication overhead.

cannot meet our computational efficiency goal. This highlights the necessity of designing a new secure natural logarithm computation method. We separately apply each computation method to our task and Table III lists the estimated time cost. The work [28] has reported the time cost of one execution of the Lindell et al.'s [27] and Teo et al.'s [28] methods on a workstation with Intel Xeon 2.3 GHz CPU (16 cores) and 20 threads. Specifically, one execution of Lindell et al.'s [27] method takes about 10s with parallel processing, while that of Teo et al.'s [28] method takes about 6s and 40s with parallel processing and serial processing, respectively. We directly refer to these results. The used thread number in our $SecLn(\cdot)$ is also set as 20 in this comparison, to provide a relatively fair comparison and one execution of our $SecLn(\cdot)$ takes about 0.016ms and 2.4ms with parallel processing and serial processing respectively in our simulation. As can be seen, it takes around 7,300 days for Teo et al.'s method [28] to securely compute all the natural logarithms with serial processing when training the smallest Dataset1, which is completely unacceptable in practice. However, our $SecLn(\cdot)$ only takes about 0.5 days for Dataset1, which is more than $10^4\times$ faster than the previous methods. Besides, the results also show that our $SecLn(\cdot)$ is up to $10^5\times$ faster in parallel processing than the previous methods.

TABLE IV

TRAINING OVERHEAD OF SEVERAL SECURE TRAINING SCHEMES UNDER THE 2-PARTY COMPUTATION (2PC) SETTING

Protocol	Model	Dataset	Time (hour)	Communication Cost (GB)
SecureML [6]	2 layer network	MNIST (52 MB)	2.9	-
Piranha [11]	VGG16	CIFAR10 (177 MB)	63.7	35,455
Quotient [10]	LeNet	MNIST (52 MB)	87.2	-
PPGloVe	GloVe	Wikimedia dump (621MB)	16.8	31,974

TABLE V

EXAMPLE QUESTIONS IN THE TEST DATASET

	Word Pair 1		Word Pair 2	
Semantic	abuja	nigeria	accra	ghana
	boy	girl	brothers	sisters
Syntactic	amazing	amazingly	apparent	apparently
	dancing	danced	increasing	increased

2) Performance on Secure Word Vectors Training: We evaluate the efficiency of traning process in terms of training time and communication cost. Fig. 6(a) shows the training time of GloVe and PPGloVe. It can be seen that our PPGloVe requires 16.80 hours to train on the biggest Dataset3, and only 1.0 hour on the smallest Dataset1. Fig. 6(b) shows the communication overhead of PPGloVe. GloVe is trained in plaintext-domain with no communication cost. For the smallest Dataset1, PPGloVe requires about 1,911.2GB to transfer data, and the communication cost increases with the increase of dataset. This training overhead is acceptable, aligning with the levels observed in prior secure 2PC training models [6], [10], [11], as demonstrated in Table IV. Note that the reported results of prior works are from their respective papers.

3) Effectiveness of PPGloVe: We also evaluate the effectiveness of PPGloVe on practical task. Note that our PPGloVe is the first privacy-preserving GloVe model, so there are no previous works for direct performance comparison. Since we have modified the original GloVe model, we also use the adapted-GloVe to train word vectors in the plaintext-domain to study the impact of these modifications. We use the parameter setting in Section VI-A to test the quality of the word vectors trained by GloVe, adapted-GloVe, and PPGloVe. Specifically, we measure the quality of these word vectors by using them on word analogies tasks [43]. The word analogy task consists of questions such as "a is to b as c is to __ ?". The test set has 19,544 questions consisting of 8,869 semantic questions and 10,675 syntactic questions. Table V shows some example questions. We answer the question "a is to b as c is to __ ?" by searching the vector space for the word d closest to $v_b - v_a + v_c$

TABLE VI
WORD ANALOGIES ACCURACY OF USING WORD VECTORS GENERATED BY DIFFERENT MODELS

Model	Semantic Accuracy (%)	Syntactic Accuracy (%)	Total Accuracy (%)
GloVe	61.54	36.17	47.67
Adapted-GloVe	61.09	34.10	46.34
PPGloVe	60.54	34.27	46.18

by cosine distance. The question is answered correctly only when the word d is the same as the correct answer.

We first generate word vectors by training the three models on Dataset3. We set the training epoch and initialized learning rate of GloVe as 50 and 0.05, respectively, according to the setting in [1]. The initialized learning rate of PPGloVe and adapted-GloVe is set to 0.13, ensuring that they can achieve a good performance. We train each model five times to obtain the average value. Table VI shows the accuracy of using different word vectors. The total accuracy of PPGloVe has fallen only 0.16% when compared to the adapted-GloVe, indicating that the errors caused by our $SecLn(\cdot)$, $SecW(\cdot)$, and other secure operations have very little impact on the system's accuracy. Meanwhile, the total accuracy of PPGloVe falls only 1.49% compared to GloVe. Such accuracy drop is common in secure training protocols. This decline is mainly caused by the insufficient training epochs (only 10 epochs for efficiency consideration). The quality of the obtained word vectors can be further improved by increasing the training epochs. Note that it typically occurs accuracy drop exceeding 2% in various secure protocols like [47], [48]. Thus, our design losses only a little accuracy and is suitable for practical tasks.

VII. RELATED WORKS

A. Privacy-Preserving Machine Learning

Existing privacy-preserving ML models can be divided into two categories. The first category focuses on secure ML inference tasks [7], [12], [13], [14], [49], which indicates that users can get correct inference results without exposing their data and model parameters to the adversaries. For example, Niu et al. [14] used the homomorphic encryption to build a secure SVM model to provide online diagnosis. The users can be offline after they have uploaded their data. Liu et al. [7] used secret sharing technique to build a secure neural network inference framework which can provide medical diagnostic service. The second category focuses on ML training tasks [6], [10], [11], which enables a ML model to be trained in the dark such that user data and trained model can be protected. Specifically, Mohassel et al. [6] combine secret sharing, garbled circuit, and oblivious transfer to realize the privacy-preserving protocol to train neural networks. Agrawal et al. [10] ternarize the network weights and design a new fixed-point optimization algorithm to make the training faster. Watson et al. [11] adopt the GPU to accelerate the secure training of neural networks. However, these representative works are different from ours. They focus on the secure training of neural networks and deal with different types of nonlinear functions compared to our research. Thus, their developed techniques cannot be applied in our specific task.

TABLE VII
RELATED WORKS IN THE FIELD OF PRIVACY-PRESERVING MACHINE LEARNING

Work	Application Scenario	Type	Model	Crypto. Technique
Liu *et al.* [7]	Medical Diagnosis	Inference	NN	SS
Xie *et al.* [12]	Medical Diagnosis	Inference	SVM	HE
Wang *et al.* [13]	Classification	Inference	NN	SS
Niu *et al.* [14]	Spam Detection	Inference	SVM	HE
Liu *et al.* [49]	Classification	Inference	NN	HE,SS
Mohassel *et al.* [6]	Classification	Training	NN	SS
Agrawal *et al.* [10]	Classification	Training	NN	SS
Watson *et al.* [11]	Classification	Training	NN	SS
Feng *et al.* [16]	Translation	Inference	NN	SS
Wang *et al.* [17]	Word Vectors	Training	Word2Vec	HE
Reich *et al.* [19]	Hate Detection	Inference	AdaBoost	SS
Resende *et al.* [18]	Spam Detection	Inference	Naive Bayes	SS
PPGloVe	Word Vectors	Training	GloVe	SS

HE: Homomorphic Encryption. SS: Secret Sharing. NN: Neural Network.

B. Privacy-Preserving NLP Models

Recently, some researches are proposed to provide security guarantee to the ML models in NLP tasks [16], [17], [18], [19], [20]. Among them, Feng et al. [16] proposed a SecureNLP scheme, focusing on the privacy preservation in a sequence-to-sequence model for neural machine translation. They combine multiplicative sharing and additive sharing to compute the nonlinear functions $sigmoid(\cdot)$ and $Tanh(\cdot)$, achieving a quicker speed than the garbled circuit-based computation. Reich et al. [19] used secure multi-party computation to implement a secure text classification system. They proposed a privacy-preserving text feature extraction protocol and adopted logistic regression and AdaBoost model as the classifier. Resende et al. [18] optimized the work in [19] by using a faster secure comparison to implement the privacy-preserving extraction of text feature and replacing the classifier with Naive Bayes. However, both works in [19] and [18] require that the users should participate in inference process and cannot be offline.

There exist several research studies that focus on designing secure protocols for word vector models [17], [20]. Specifically, Wang et al. [17] present a secure training protocol for the Word2Vec model [23]. They use Paillier encryption [21] as the cryptographic primitive and the quality of word vectors obtained in the ciphertext domain is comparable with that in the plaintext domain. The work in [20] adopts CKKS encryption scheme [22] to realize a secure protocol for the fastText [24] model and uses GPU to accelerate the computation process. However, both of the works use homomorphic encryption algorithms, causing significant computation burden on the cloud side. As reported in [17], the largest size of the training dataset being evaluated is limited to only 80 MB, which is clearly insufficient for many practical applications. Differently, PPGloVe utilizes lightweight secret sharing technique, leading to low computational overhead. Besides, PPGloVe necessitates the secure computation of the natural logarithm, which is

not involved in these previous works. Table VII provides a summary of comparison with the related works we have mentioned.

VIII. Conclusion

In this paper, we present PPGloVe, the first system framework that enables many collaborating users to securely train word vectors using GloVe on the cloud. We use the lightweight additive secret sharing to ensure the security of system and devise some secure and lightweight arithmetic primitives as components of our design. We propose a fast secure natural logarithm computation method that is $10^4\times$ faster than previous methods. By design, the corpus information from participants and the trained word vectors are all kept private along the whole training flow. We theoretically analyze the complexity of PPGloVe and justify its security in the semi-honest and non-colluding adversary model. The extensive experiments demonstrate that PPGloVe can achieve word vectors with comparable quality to the results obtained in the plaintext-domain, with practically affordable overhead.

References

[1] J. Pennington, R. Socher, and C. Manning, "GloVe: Global vectors for word representation," in *Proc. Conf. Empirical Methods Natural Lang. Process. (EMNLP)*, 2014, pp. 1532–1543.

[2] L. Burgueño, R. Clarisó, S. Gérard, S. Li, and J. Cabot, "An NLP-based architecture for the autocompletion of partial domain models," in *Proc. Conf. Adv. Inf. Syst. Eng.*, Jun. 2021, pp. 91–106.

[3] Y. Sharma, G. Agrawal, P. Jain, and T. Kumar, "Vector representation of words for sentiment analysis using GloVe," in *Proc. Int. Conf. Intell. Commun. Comput. Techn. (ICCT)*, Dec. 2017, pp. 279–284.

[4] X. Liu, R. H. Deng, K. R. Choo, and Y. Yang, "Privacy-preserving outsourced support vector machine design for secure drug discovery," *IEEE Trans. Cloud Comput.*, vol. 8, no. 2, pp. 610–622, Apr. 2020.

[5] H. Chen, H. Li, Y. Wang, M. Hao, G. Xu, and T. Zhang, "PriVDT: An efficient two-party cryptographic framework for vertical decision trees," *IEEE Trans. Inf. Forensics Security*, vol. 18, pp. 1006–1021, 2023.

[6] P. Mohassel and Y. Zhang, "SecureML: A system for scalable privacy-preserving machine learning," in *Proc. IEEE Symp. Security Privacy (SP)*, 2017, pp. 19–38.

[7] X. Liu, Y. Zheng, X. Yuan, and X. Yi, "Towards secure and lightweight deep learning as a medical diagnostic service," in *Proc. Eur. Symp. Res. Comput. Secur.*, Oct. 2021, pp. 519–541.

[8] N. Kumar, M. Rathee, N. Chandran, D. Gupta, A. Rastogi, and R. Sharma, "CrypTFlow: Secure TensorFlow inference," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2020, pp. 336–353.

[9] P. Mohassel and P. Rindal, "ABY3: A mixed protocol framework for machine learning," in *Proc. ACM SIGSAC Conf. Comput. Commun. Security*, 2018, pp. 35–52.

[10] N. Agrawal, A. Shahin, M. J. Kusner, and A. Gascón, "QUOTIENT: Two-party secure neural network training and prediction," in *Proc. ACM SIGSAC Conf. Comput. Commun. Security*, London, U.K., 2019, pp. 1231–1247.

[11] J.-L. Watson, S. Wagh, and R. A. Popa, "Piranha: A GPU platform for secure computation," in *Proc. USENIX Secur. Symp.*, Aug. 2022, pp. 827–844.

[12] B. Xie, T. Xiang, X. Liao, and J. Wu, "Achieving privacy-preserving online diagnosis with outsourced SVM in Internet of Medical Things environment," *IEEE Trans. Dependable Secure Comput.*, vol. 19, no. 6, pp. 4113–4126, Nov. 2022.

[13] J. Wang, D. He, A. Castiglione, B. B. Gupta, M. Karuppiah, and L. Wu, "PCNNCEC: Efficient and privacy-preserving convolutional neural network inference based on cloud-edge-client collaboration," *IEEE Trans. Netw. Sci. Eng.*, early access, May 26, 2022, doi: 10.1109/TNSE.2022.3177755.

[14] C. Niu, F. Wu, S. Tang, S. Ma, and G. Chen, "Toward verifiable and privacy preserving machine learning prediction," *IEEE Trans. Dependable Secure Comput.*, vol. 19, no. 3, pp. 1703–1721, May 2022.

[15] J.-C. Bajard, P. Martins, L. Sousa, and V. Zucca, "Improving the efficiency of SVM classification with FHE," *IEEE Trans. Inf. Forensics Security*, vol. 15, pp. 1709–1722, 2020.

[16] Q. Feng, D. He, Z. Liu, H. Wang, and K. R. Choo, "SecureNLP: A system for multi-party privacy-preserving natural language processing," *IEEE Trans. Inf. Forensics Security*, vol. 15, pp. 3709–3721, 2020.

[17] Q. Wang et al., "Privacy-preserving collaborative model learning: The case of word vector training," *IEEE Trans. Knowl. Data Eng.*, vol. 30, no. 12, pp. 2381–2393, Dec. 2018.

[18] A. Resende, D. Railsback, R. Dowsley, A. C. A. Nascimento, and D. F. Aranha, "Fast privacy-preserving text classification based on secure multiparty computation," *IEEE Trans. Inf. Forensics Security*, vol. 17, pp. 428–442, 2022.

[19] D. Reich, A. Todoki, R. Dowsley, M. D. Cock, and A. C. A. Nascimento, "Privacy-preserving classification of personal text messages with secure multi-party computation," in *Proc. Adv. Neural Inf. Process. Syst.*, Dec. 2019, pp. 3752–3764.

[20] A. A. Badawi, L. Hoang, C. F. Mun, K. Laine, and K. M. M. Aung, "PrivFT: Private and fast text classification with homomorphic encryption," *IEEE Access*, vol. 8, pp. 226544–226556, 2020.

[21] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *Proc. Int. Conf. Theory Appl. Cryptograph. Techn.*, Jan. 1999, pp. 223–238.

[22] J. H. Cheon, A. Kim, M. Kim, and Y. Song, "Homomorphic encryption for arithmetic of approximate numbers," in *Proc. Int. Conf. Theory Appl. Cryptol. Inf. Secur.*, 2017, pp. 409–437.

[23] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Proc. Adv. Neural Inf. Process. Syst.*, Dec. 2013, pp. 3111–3119.

[24] A. Joulin, E. Grave, P. Bojanowski, and T. Mikolov, "Bag of tricks for efficient text classification," in *Proc. 15th Conf. Eur. Chapter Assoc. Comput. Linguistics*, 2017, pp. 427–431.

[25] S. Wagh, S. Tople, F. Benhamouda, E. Kushilevitz, P. Mittal, and T. Rabin, "FALCON: Honest-majority maliciously secure framework for private deep learning," in *Proc. Priv. Enhancing Technol. Symp.*, Jul. 2021, pp. 188–208.

[26] Z. Ma, Y. Liu, X. Liu, J. Ma, and F. Li, "Privacy-preserving outsourced speech recognition for smart IoT devices," *IEEE Internet Things J.*, vol. 6, no. 5, pp. 8406–8420, Oct. 2019.

[27] Y. Lindell and B. Pinkas, "Privacy preserving data mining," in *Proc. Annu. Int. Cryptol. Conf.*, Aug. 2000, pp. 36–54.

[28] S. G. Teo, J. Cao, and V. C. S. Lee, "DAG: A general model for privacy-preserving data mining," *IEEE Trans. Knowl. Data Eng.*, vol. 32, no. 1, pp. 40–53, Jan. 2020.

[29] D. Demmler, T. Schneider, and M. Zohner, "ABY—A framework for efficient mixed-protocol secure two-party computation," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, Feb. 2015, pp. 1–15.

[30] D. Beaver, "Efficient multiparty protocols using circuit randomization," in *Proc. Annu. Int. Cryptol. Conf.*, 1991, pp. 420–432.

[31] M. S. Riazi, C. Weinert, O. Tkachenko, E. M. Songhori, T. Schneider, and F. Koushanfar, "Chameleon: A hybrid secure computation framework for machine learning applications," in *Proc. Asia Conf. Comput. Commun. Security*, 2018, pp. 707–721.

[32] A. Patra, T. Schneider, A. Suresh, and H. Yalame, "ABY2.0: Improved mixed-protocol secure two-party computation," in *Proc. USENIX Secur. Symp.*, Aug. 2021, pp. 2165–2182.

[33] Y. Zheng, H. Duan, and C. Wang, "Towards secure and efficient outsourcing of machine learning classification," in *Proc. Eur. Symp. Res. Comput. Security*, 2019, pp. 22–40.

[34] D. Rathee et al., "CrypTFlow2: Practical 2-party secure inference," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2020, pp. 325–342.

[35] X. Liu, Y. Zheng, X. Yuan, and X. Yi, "Securely outsourcing neural network inference to the cloud with lightweight techniques," *IEEE Trans. Dependable Secure Comput.*, vol. 20, no. 1, pp. 620–636, Jan. 2023.

[36] M. Li, S. S. M. Chow, S. Hu, Y. Yan, C. Shen, and Q. Wang, "Optimizing privacy-preserving outsourced convolutional neural network predictions," *IEEE Trans. Dependable Secure Comput.*, vol. 19, no. 3, pp. 1592–1604, May 2022.

[37] P. Mishra, R. Lehmkuhl, A. Srinivasan, W. Zheng, and R. A. Popa, "DELPHI: A cryptographic inference service for neural networks," in *Proc. 29th USENIX Secur. Symp.*, 2020, pp. 2505–2522.

[38] N.-W. Lo and J.-L. Tsai, "An efficient conditional privacy-preserving authentication scheme for vehicular sensor networks without pairings," *IEEE Trans. Intell. Transp. Syst.*, vol. 17, no. 5, pp. 1319–1328, May 2016.

[39] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *J. Mach. Learn. Res.*, vol. 12, no. 61, pp. 2121–2159, 2011.

[40] S. Wang, Y. Zheng, X. Jia, and X. Yi, "Privacy-preserving analytics on decentralized social graphs: The case of eigendecomposition," *IEEE Trans. Knowl. Data Eng.*, vol. 35, no. 7, pp. 7341–7356, Jul. 2023.

[41] S. Akram and Q. U. Ann, "Newton Raphson method," *Int. J. Sci. Eng. Res.*, vol. 6, no. 7, pp. 1748–1752, 2015.

[42] M. Kelkar, P. H. Le, M. Raykova, and K. Seth, "Secure Poisson regression," in *Proc. USENIX Secur. Symp.*, Aug. 2022, pp. 791–808.

[43] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," 2013, *arXiv:1301.3781*.

[44] D. Bogdanov, S. Laur, and J. Willemson, "SHAREMIND: A framework for fast privacy-preserving computations," in *Proc. Eur. Symp. Res. Comput. Secur.*, Oct. 2008, pp. 192–206.

[45] M. S. Riazi, M. Samragh, H. Chen, K. Laine, K. Lauter, and F. Koushanfar, "XONN: XNOR-based oblivious deep neural network inference," in *Proc. USENIX Secur. Symp.*, Aug. 2019, pp. 1501–1518.

[46] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov, "Enriching word vectors with subword information," *Trans. Assoc. Comput. Linguistics*, vol. 5, pp. 135–146, Dec. 2017.

[47] M. Abspoel, D. Escudero, and N. Volgushev, "Secure training of decision trees with continuous attributes," in *Proc. Priv. Enhancing Technol. Symp.*, Jan. 2021, pp. 167–187.

[48] Q. Li, Z. Wu, Z. Wen, and B. He, "Privacy-preserving gradient boosting decision trees," in *Proc. AAAI Conf. Artif. Intell.*, 2020, pp. 784–791.

[49] J. Liu, M. Juuti, Y. Lu, and N. Asokan, "Oblivious neural network predictions via MiniONN transformations," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2017, pp. 619–631.

Zhongyun Hua (Senior Member, IEEE) received the B.S. degree in software engineering from Chongqing University, Chongqing, China, in 2011, and the M.S. and Ph.D. degrees in software engineering from the University of Macau, Macao, China, in 2013 and 2016, respectively.

He is currently an Associate Professor with the School of Computer Science and Technology, Harbin Institute of Technology, Shenzhen, China. His works have appeared in prestigious venues, such as IEEE TRANSACTIONS ON DEPENDABLE AND SECURE COMPUTING, IEEE TRANSACTIONS ON IMAGE PROCESSING, IEEE TRANSACTIONS ON SIGNAL PROCESSING, IEEE TRANSACTIONS ON MULTIMEDIA, and *ACM Multimedia*. He has published about 80 papers on the subject, receiving more than 6000 citations. His current research interests include chaotic systems, multimedia security, and secure cloud computing. He has been recognized as a Highly Cited Researcher 2023 and a Highly Cited Researcher 2022.

Yan Tong received the B.E. degree in computer science and technology from the Harbin Institute of Technology, Harbin, China, in 2021. He is currently pursuing the degree with the Harbin Institute of Technology, Shenzhen, China. His current research interests include cloud computing security and secure machine learning.

Yifeng Zheng received the Ph.D. degree in computer science from the City University of Hong Kong, Hong Kong, in 2019. He was a Post-Doctoral Researcher with the Commonwealth Scientific and Industrial Research Organization (CSIRO), Australia, and the City University of Hong Kong. He is currently an Assistant Professor with the School of Computer Science and Technology, Harbin Institute of Technology, Shenzhen, China. His work has appeared in prestigious venues, such as ESORICS, DSN, ACM AsiaCCS, IEEE INFOCOM, IEEE ICDCS, IEEE TRANSACTIONS ON DEPENDABLE AND SECURE COMPUTING, IEEE TRANSACTIONS ON INFORMATION FORENSICS AND SECURITY, and IEEE TRANSACTIONS ON SERVICES COMPUTING. His current research interests include security and privacy related to cloud computing, the IoT, machine learning, and multimedia. He received the Best Paper Award from the European Symposium on Research in Computer Security (ESORICS) in 2021.

Yuhong Li received the bachelor's degree in software engineering from Xidian University, Xi'an, China, in 2010, and the Ph.D. degree from the Department of CIS, University of Macau, Macau, China, in 2016.

He is currently a Senior Staff Engineer with Xiaohongshu Technology Company Ltd., Shanghai, China. He has authored more than 20 papers on top-tier conferences and journals, including *The VLDB Journal*, ICDE, ICASSP, CIKM, IEEE TRANSACTIONS ON VISUALIZATION AND COMPUTER GRAPHICS, and EDBT. His current research interests include AI for social good, multimodal understanding, self-supervised learning on big data, and edge computing. He was a recipient of the Stars of Tomorrow (Award of Excellent Intern), Microsoft Research Asia, Technology Talent of Xidian University, and AliStar.

Yushu Zhang (Senior Member, IEEE) received the B.S. degree from the School of Science, North University of China, Taiyuan, China, in 2010, and the Ph.D. degree from the College of Computer Science and Technology, Chongqing University, Chongqing, China, in 2014. He held various research positions with the City University of Hong Kong, Southwest University, the University of Macau, and Deakin University. He is currently a Professor with the College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, China. His research interests include multimedia security, blockchain, and artificial intelligence security. He is an Associate Editor of *Information Sciences*, *Journal of King Saud University-Computer and Information Sciences*, and *Signal Processing*.