An efficient secure k nearest neighbor classification protocol with high-dimensional features

Maohua Sun | Ruidi Yang

School of Management and Engineering, Capital University of Economics and Business, Beijing, China

Correspondence
Maohua Sun, School of Management and Engineering, Capital University of Economics and Business, Beijing 100070, China.
Email: sunmaohua@cueb.edu.cn

Abstract
k Nearest neighbor (kNN) classification algorithm is a prediction model which is widely used for real-life applications, such as healthcare, finance, computer vision, personalization recommendation and precision marketing. The arrival of data explosion era results in the significant increase of feature dimension, which also makes for the increase of privacy concern over the available samples and unlabeled data in the applications of machine learning. In this paper, we present a secure low communication overhead kNN classification protocol that is able to deal with high-dimensional features given in real numbers. First, to deal with feature values given in real numbers, we develop a specific data conversion algorithm, which is used in the chosen fully homomorphic scheme. This conversion algorithm is generic and applicable to other algorithms that need to handle real numbers using the fully homomorphic scheme. Second, we present a privacy-preserving euclidean distance protocol (PPEDP), which works with the Euclidean distance computation between two points given in real numbers in a high-dimensional space. Then, based on the novelty PPEDP and oblivious transfer, we...
propose a new classification approach, efficient secure kNN classification protocol, (ESkNN) with low communication overhead, which is appropriate for a sample set with high-dimensional features and real number feature values. Moreover, we implement ESkNN in C++. Experimental results show that ESkNN is several orders of magnitude faster in performance than existing works, and scales up to 18 000 feature dimension in a memory limited environment.

KEYWORDS
fully homomorphic scheme, honest-but-curious model, machine learning, oblivious transfer, privacy-preserving Euclidean distance, secure kNN, secure multi-party computation

1 | INTRODUCTION

The usage and applications of machine learning in different fields, such as finance, commerce, genetics, geography, information science, and so forth, have grown rapidly these days. This popularity brings about privacy and security concerns in real-life setting applications. For example, using machine learning to predict disease susceptibility in the field of genetics results in individuals’ concern of privacy breach, thus, people are unwilling to expose their genetic data. Whereas, gene banks and hospitals are required to not expose their samples because of privacy regulations. An intuitive approach to solve this problem is by using a trusted third party. Regrettably, there exists no such fully reliable third party.

In the past few years, secure computation methods have had significant performance improvement. There have been more works focusing on the design of privacy-preserving machine learning protocols using the state-of-the-art secure computation methods. Existing privacy-preserving support vector machine (SVM) protocols use differential privacy techniques,1–3 secure multiparty computation (SMC)3–7 and somewhat homomorphic encryption.8 Furthermore, using SMC or homomorphic encryption, there exists protocols that realize secure logistic regression (LR)9–14 or secure convolutional neural networks (CNNs) prediction.10,15–20 Despite the fact that SVM, LR, and CNNs are the most popular machine learning algorithms, k nearest neighbor (kNN) classification remains to be used in many fields such as healthcare, finance, computer vision, personalization recommendation, and precision marketing. In this paper, we mainly focus on the secure kNN classification protocol, which allows two parties to predict an unlabeled item based on a sample set while revealing nothing more than the corresponding label.

1.1 | Related works

Earlier works on secure machine learning with high-dimensional features given in real numbers mainly focus on LR,9–11 linear regression,10 and neural network.10 SecureML10 supports
the privacy-preserving machine learning for linear regression, LR, and neural network training using the stochastic gradient descent method. Using a novel strategy, SecureML is able to work with real numbers and had been tested on a data set with up to 5000 features. More recently, Cock et al.\(^9\) proposes a gradient descent-based algorithm that trains LR model in a secure way. Their algorithm’s novelty point is a protocol that computes the activation function, which does not require secure comparison protocols or Yao’s garbled circuits. The said algorithm was tested using data sets which include up to 17 814 features. Hong\(^11\) presents a privacy-preserving collaborative machine learning using TensorFlow, which performs well on data sets with up to 16 384 features.

Most of the earlier works on secure kNN classification adopt the linear scan strategy.\(^21–31\) Chen et al.\(^32\) present a secure kNN classification protocol using clustering strategy. To achieve privacy-preserving protocol, their solution uses additively homomorphic encryption, distributed oblivious RAM, and garbled circuits. They tested their protocol using a database consisting of 10 million entries, but only with 128 features. Whereas, Xu et al.\(^33\) introduces a privacy-preserving protocol that is based on kd-tree data structure. This protocol employs the technique of combining oblivious RAM and SMC, which does not work well with high-dimensional features due to the fact that kd-tree structure only works well for low-dimensional data.

Many work use Paillier scheme,\(^21,24–26\) ElGamal scheme,\(^22\) BFV homomorphic scheme,\(^32\) or oblivious transfer (OT)-based multiplication\(^27\) to compute the distance between the query and the samples in a secure manner. As for the top-k selection, either the garbled circuit\(^28–30,32\) or BGV homomorphic scheme\(^31,34\) is used. To be specific, Samanthula et al.\(^21\) propose a secure kNN classifier over encrypted data in the cloud using Paillier scheme. Whereas, Rong et al.\(^22\) present a secure outsourced kNN classifier in multiple cloud environments based on the proxy re-encryption (PRE) technique, where different data owners outsource their private data to different cloud server providers. Here, they use a two-servers model with the assumption that the two servers in every cloud environment do not collude with each other. Shaul et al.\(^34\) propose a secure k-ish nearest neighbor classifier, where the output label is computed by the nearest neighbors for \(\kappa \approx k\) with some probability. They implement their classifier using the BGV homomorphic scheme. Experiments show that the accuracy of their classifier is 98%.

While most of the previous works consider the security under the honest-but-curious model, Schoppmann et al.\(^28\) propose a privacy-preserving kNN (PPkNN) classifier based on the cosine similarity between documents represented by their TF-IDF features under the malicious model.

These varieties of existing works were made in effort to make kNN classification more secure. However, they mostly focus on the upscale of the number of samples and do not really consider supporting data using real numbers.

1.2 Motivations and contributions

In the age of big data, secure kNN classification needs to be applied to the large data set. In fact, with the development of artificial intelligence (AI), internet of things (IoT), genomics and genetics, the dimension of features is increasing dramatically. Existing secure kNN classification protocols mainly focus on the scale up in the horizontal direction, that is, the number of samples. Thus, many applications that deal with high-dimensional features have to adopt insecure solution instead. Another common issue for the existing protocols is the inability to deal with real numbers computation, which are often the form of the data given in real-life applications. As this computation is prohibitively expensive to be done on homomorphic
encryption or SMC, such as garbled circuits, most of the existing privacy-preserving machine learning protocols deal only with integers. This results in the inapplicability of these protocols to real-life applications.

Recall that most of the existing secure kNN protocol adopt the linear scan strategy, where the distance between the query and every sample in the training set needs to be computed in a privacy-preserving manner. Meanwhile, large amount of information about the relationship between the query and every sample needs to be transferred from one party to another. With the development of cloud computing, the computing power can easily be improved. In contrast, the communication infrastructure is expensive. Thus, another unsolved common issue of the existing secure kNN protocols is the reduction of communication overhead.

In this paper, we present a secure low communication overhead kNN classification protocol that is able to deal with high-dimensional features given in real numbers. The main novelty contributions of our work are as follows:

(1) We present a new data conversion algorithm that works with real numbers that can be used with the BGV homomorphic scheme. This conversion algorithm is generic and applicable to other algorithms with BGV homomorphic scheme that also need to handle real numbers.

(2) We propose a privacy-preserving euclidean distance protocol (PPEDP) which is based on the BGV homomorphic scheme. By using the novel data conversion algorithm, PPEDP computes the Euclidean distance between two points given in real numbers in a high-dimensional space.

(3) We propose a new classification approach, efficient secure kNN (ESkNN) classification protocol, which is built based on PPEDP and OT. Likewise, our ESkNN can deal with real numbers. We build our protocols using HElib and EMP toolkit, which enables our building block PPEDP to be used on/for other applications. Our experimental results show that our ESkNN performs efficiently, when it is applied to a sample set with high-dimensional features. The feature dimension is scaled up to 18 000 in a memory limited environment. Moreover, the communication overhead is low with carefully selected parameters. The performance of ESkNN is up to 402 times and 17 times speedup, when compared to the performances of existing protocols PPkNN and OCKNN, respectively. Our ESkNN, along with PPEDP, is secure in the honest-but-curious model. We provide the detailed security definitions and proofs for our protocols in the appendix.

Our protocols fall into the category of SMC, where we use the traditional mode as shown and explained in Figure 1. In our protocols, there are two involving parties, Alice and Bob. Assuming that Alice has a private sample data set, ESkNN allows the two parties to classify a

![Figure 1](https://wileyonlinelibrary.com)
private query input by Bob using Alice’s data set in a way that Bob gets the resulting label without him learning anything about Alice’s data set. Furthermore, Alice will not learn anything about Bob’s input and the query result.

1.3 Outline

The remainder of this paper is set out as follows. We introduce the preliminaries in Section 2. Section 3 briefly introduces the data conversion algorithm. Next, it presents the build block, PPDEP. Section 4 propose the ESkNN classification protocol with the performance analysis. After that, we describe the implementation and experiments in Section 5. Finally, we present our concluding remarks in Section 6.

2 PRELIMINARY

2.1 Notation

The notations we use in this paper are as follows. σ represents the digits of the fraction part of the real number used in our protocols. Within the kNN algorithm, m is the size of samples, while n represents the feature dimension. In the BGV homomorphic scheme, τ is the slot length, λ is the security level, and p is the plaintext prime modulus. ρ is the bucket length which will be introduced and used in our ESkNN. For a real number r, ⌈r⌉ represents the smallest integer bigger than r. δnτ represents the number of the ciphertext corresponding to every sample. ε = δ/r represents the number of buckets corresponding to every sample.

2.2 kNN classification algorithm

kNN classification algorithm is one of the best algorithms for multivariate classification in the field of machine learning. Euclidean distance is a commonly used metric in kNN. More specifically, for two points \(x = \langle x_1, x_2, ..., x_n \rangle\) and \(y = \langle y_1, y_2, ..., y_n \rangle\) in a \(n\)-dimensional space, we define the Euclidean distance as \(e = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + \cdots + (x_n - y_n)^2}\).

In the kNN classification algorithm, the training data set \([S, Y]_{m \times (n+1)}\) consists of \(m\) samples \([S_i, y_i]\) in which \(S_i = \langle s_{i,1}, s_{i,2}, ..., s_{i,n} \rangle\) is a \(n\)-dimensional feature vector, and \(y_i \in \{1, 2, ..., t\}\) is the label of \(S_i\). In practice, \(s_{i,j}\) is always a real number. The kNN classification algorithm classifies a query \(X = \langle x_1, x_2, ..., x_n \rangle\) to a label \(y\) based on the labels of the \(k\) nearest neighbors. We describe the kNN classification algorithm in Algorithm 1.

Algorithm 1. kNN classification algorithm

1. for \(i \leftarrow 0\) to \(m\) do \(e_i \leftarrow \sqrt{(s_{i,1} - x_1)^2 + (s_{i,2} - x_2)^2 + \cdots + (s_{i,n} - x_n)^2}\) \(E = [e_1, e_2, \cdots, e_m]\)
2. \([p_1, p_2, ..., p_k] = \text{Min}(E, k)\)
3. Compute \(y\) by \([y_{p_1}, y_{p_2}, ..., y_{p_k}]\) using based on the chosen decision rule.

\((\text{Min}(E, k)\) is a function which returns the positions of the \(k\) minimum values in \(E\).)
2.3 | Fully homomorphic encryption

In this paper, we use the BGV fully homomorphic encryption (FHE) scheme to computes the Euclidean distance between two points in a privacy-preserving manner. Assuming that $\lambda$ is the security level, $\tau$ is the slot length and $p$ is the plaintext prime modulus, the BGV homomorphic scheme consists the following algorithms:

FHE.KeyGen($\lambda$) $\rightarrow$ $(pk, sk)$. Given security parameter $\lambda$, the function generates the key pairs $pk$ and $sk$ used for encryption and decryption.

FHE.Enc($sk, M$) $\rightarrow$ $E(M)$. The function encrypts the plaintext vector $M = \{m_1, m_2, ..., m_{\tau}\}$ to a ciphertext $E(M)$.

FHE.CAdd($E(M), E(N)$) $\rightarrow$ $V$. Given encryptions of the plaintext vectors $M = \{m_1, m_2, ..., m_{\tau}\}$ and $N = \{n_1, n_2, ..., n_{\tau}\}$, the function outputs an encryption of the plaintext vector $V = \{m_1 + n_1, m_2 + n_2, ..., m_{\tau} + n_{\tau}\}$.

FHE.CMult($E(M), E(N)$) $\rightarrow$ $V$. Given encryptions of the plaintext vectors $M = \{m_1, m_2, ..., m_{\tau}\}$ and $N = \{n_1, n_2, ..., n_{\tau}\}$, the function outputs an encryption of the plaintext vector $V = \{m_1 \times n_1, m_2 \times n_2, ..., m_{\tau} \times n_{\tau}\}$.

FHE.Sub($E(M), E(N)$) $\rightarrow$ $V$. Given encryptions of the plaintext vectors $M = \{m_1, m_2, ..., m_{\tau}\}$ and $N = \{n_1, n_2, ..., n_{\tau}\}$, the function outputs an encryption of the plaintext vector $V = \{m_1 - n_1, m_2 - n_2, ..., m_{\tau} - n_{\tau}\}$.

FHE.Dec($sk, E(M)$) $\rightarrow$ $M$. Using the private key $sk$, the function decrypts the ciphertext $E(M)$ to get plaintext vector $M = \{m_1 \mod p, m_2 \mod p, ..., m_{\tau} \mod p\}$.

2.4 | Oblivious transfer

OT is a fundamental cryptographic primitive that is widely used in fields such as SMC. In $OT^4_2$, the sender $S$ enters a character pair $(x_0, x_1)$, where $x_0$ is a random number and $x_1$ is a meaningful value. While the receiver $R$ has the selection bit $b \in \{0, 1\}$, through $OT^4_2$, $R$ can obtain $x_b \in \{x_0, x_1\}$ without revealing any information about $b$ to the sender $S$, and receiver $R$ cannot obtain any information other than $x_b$.

In this paper, we use a more commonly used OT which is called $m \times OT^4_2$. Let’s consider the scenario that $S$ inputs $m$ tuples $(x^j_0, x^j_1)$ with $1 \leq j \leq m$, and $R$ inputs $m$ selection bits $r = (r_1, ..., r_m)$. Through $m \times OT^4_2$, $R$ obtains $(x^1_{r_1}, ..., x^m_{r_m})$ in a secure manner as $OT^4_2$. For simplicity, we denote $m \times OT^4_2$ by OT from here on.

2.5 | Security definition

In this paper, we use the honest-but-curious model which is widely used in SMC. Additionally, we follow the simulation-based security definition in the SMC literature.$^{37}$

**Definition 1.** Private computation under Honest-but-Curious model.

In the process of protocol $\pi$, the information that participants $P_1$ and $P_2$ obtain are recorded as:

$$VIEW^\pi_1(x, y) = (x, r^1, m^1_1, m^1_2, ..., m^1_{\tau}),$$

(1)
\[ \text{VIEW}_2^\pi(x, y) = (x, r^2, m_1^2, m_2^2, \ldots, m_i^2). \quad (2) \]

In the formulas above, \( r^i \) represents the random number generated by \( P_i \), and \( m_j^i \) represents the \( j \)th message received by \( P_i \). After the protocol, the output of \( P_i \) is recorded as \( \text{OUTPUT}_i^\pi(x, y) \). We can see that, in fact, \( \text{OUTPUT}_i^\pi(x, y) \) is part of \( \text{VIEW}_i^\pi(x, y) \).

Protocol \( \pi \) computes a deterministic function \( f \) under the honest-but-curious model privately if and only if there are two probability polynomial time algorithms \( S_1 \) and \( S_2 \) who conforms the following formulas:

\[
\begin{align*}
\{S_1(x, f_1(x, y))\}_{x, y \in \{0, 1\}} & \equiv \left\{ \text{VIEW}_i^\pi(x, y) \right\}_{x, y \in \{0, 1\}} \quad (3) \\
\{S_2(y, f_2(x, y))\}_{x, y \in \{0, 1\}} & \equiv \left\{ \text{VIEW}_2^\pi(x, y) \right\}_{x, y \in \{0, 1\}} \quad (4)
\end{align*}
\]

3 | PRIVACY-PRESERVING EUCLIDEAN DISTANCE PROTOCOL

In this section, we present our PPEDP. First, we propose a data conversion algorithm which is used in the BGV homomorphic scheme. Then, we present our PPEDP and analyze the security and efficiency.

3.1 | Data conversion algorithm

To use the BGV homomorphic scheme on a real number, we need to convert it to an integer in advance. Assuming a real number \( x \) with a decimal representation has \( \sigma \) digits in the fraction part, \( x \) is converted into an integer \( y \) by \( y = (x \times 10^\sigma) \mod p \) where \( p \) is the plaintext prime modulus in the BGV homomorphic scheme. When we decrypt a ciphertext \( y \), the plaintext \( D(y) \) need to be converted into its true value \( x \) according to the operations performed on the ciphertext.

If no operations were performed on the ciphertext \( y \), we can compute the true plaintext by \( x = (10^{-\sigma} \times D(y)) \mod p \). If \( y \) was obtained by \( m \) multiplications on some other ciphertext, we have \( x = (10^{-(m+1)\sigma} \times D(y)) \mod p \). If \( y \) was obtained by \( m \) additions or subtractions on some other ciphertext, we have \( x = (10^{-\sigma} \times D(y)) \mod p \).

3.2 | PPEDP

Alice has a point \( x = \langle x_1, x_2, \ldots, x_n \rangle \) in a \( n \)-dimensional space, meanwhile Bob has a point \( y = \langle y_1, y_2, \ldots, y_n \rangle \). Alice and Bob want to calculate the Euclidean distance between their points without revealing their point information. After the calculation, only Bob gets the resulting Euclidean distance.

In practice, the values in a \( n \)-dimensional space are always real numbers. Before the protocol, Alice and Bob initially needs to convert their input to integers using our Data Conversion Algorithm as described above, so that the feature values can be performed using the BGV homomorphic scheme. For simplicity, we assume that every value in \( x \) and \( y \) has been multiplied by \( 10^{-\sigma} \) before the protocol. Our PPEDP is described in Protocol 1.
Protocol 1: Privacy-Preserving Euclidean Distance Protocol

**Input**: Alice inputs $x = <x_1, x_2, ..., x_n>$, Bob inputs $y = <y_1, y_2, ..., y_n>$.

**Output**: Alice gets nothing, Bob gets the Euclidean distance $e = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + \cdots + (x_n - y_n)^2}$.

**System Parameters**: $\tau, \rho, \lambda, \delta, \varepsilon$

**Preprocessing stage**:
1. Alice calls the key generation algorithm of the BGV cryptosystem to generate the system key $(pk, sk)$.
   \[
   (pk, sk) \leftarrow \text{FHE.KeyGen}(1^\lambda)
   \]
2. Alice encrypts $x$ using the public key $pk$ and gets the ciphertext $E(x)$.
   for $i \leftarrow 1$ to $\delta$
   \[
   A_i \leftarrow <x_{i+\tau+1}, x_{i+\tau+2}, \ldots, x_{i+\tau}> \\
   E(A_i) \leftarrow \text{FHE.Enc}(pk, A_i)
   \]
   $E(x) \leftarrow <E(A_1), E(A_2), \ldots, E(A_\delta)>$
3. Alice discloses $pk, E(x)$ on network.
4. Bob gets $pk, E(x)$ published by Alice.

**Processing stage**:
1. Bob encrypts its input data $y$ using the public key $pk$ and gets the ciphertext $E(y)$.
   for $i \leftarrow 1$ to $\delta$
   \[
   B_i \leftarrow <y_{i+\tau+1}, y_{i+\tau+2}, \ldots, y_{i+\tau}> \\
   E(B_i) \leftarrow \text{FHE.Enc}(pk, B_i)
   \]
   $E(y) \leftarrow <E(B_1), E(B_2), \ldots, E(B_\delta)>$
2. Bob performs multiplication and addition on ciphertext to obtain a ciphertext set $C$.
   for $i \leftarrow 1$ to $\varepsilon$
   for $j \leftarrow 1$ to $\rho$
   \[
   \Delta C \leftarrow \text{FHE.CMult}(E(A_i), E(B_j))
   \]
   $C_i \leftarrow \text{FHE.CAdd}(C_i, \Delta C)$
   $r_i \leftarrow \text{random}$
   $E(r_i) \leftarrow \text{FHE.Enc}(pk, r_i)$
   $C_i \leftarrow \text{FHE.CAdd}(C_i, E(r_i))$
   $C = <C_1, C_2, ..., C_\varepsilon>$
3. Bob sends $C$ to Alice.
4. Alice receives the ciphertext set $C$.
5. Alice decrypts $C$ using the private key $sk$ to obtain $v$.
   \[
   \alpha = [\alpha_1, \alpha_2, ..., \alpha_\varepsilon]
   \]
   $v = 10^{-\sigma} \times \sum_{i=1}^{\varepsilon} x_i^2$
   for $i \leftarrow 1$ to $\varepsilon$
   \[
   \alpha \leftarrow \text{FHE.Dec}(sk, C_i)
   \]
   for $j \leftarrow 1$ to $\tau$
   \[
   v \leftarrow v - (2 * \alpha_j) \times 10^{-\sigma}
   \]
6. Alice sends $v$ to Bob.
7. Bob receives $v$ and calculates the Euclidean distance $e = \sqrt{10^{-\sigma} \times \sum_{i=1}^{\varepsilon} y_i^2 + v - 10^{-\sigma} \times \sum_{i=1}^{\varepsilon} r_i}$. 
3.3 Correctness and efficiency

**Theorem 1.** Assuming every element of the input can be represented in \( \mu \) digits after data conversion, our PPEDP computes the Euclidean distance between two points correctly when \( \mu < \log_2 p/(\rho + 1) \).

**Proof.** To compute the Euclidean distance correctly, the logical plaintext related to every ciphertext need to be less than \( p \), where \( p \) is the plaintext prime modulus in the BGV homomorphic scheme. We observe that the largest plaintext is decrypted form \( \text{FHE.Dec}(sk, C_i) \) in step 5. Assuming \( \alpha' \) is one of the logical plaintext related to \( \text{FHE.Dec}(sk, C_i) \). To make it easier to understand, we consider the logical operations in clear. \( \alpha' \) is obtained by \( \delta \) additions on some product of two elements. So, we have \( (\rho + 1) \times (10^\mu - 1)^2 < p \). That is, \( \mu < \log_2 p/(\rho + 1) \).

As many previous works, we mainly focus on the efficiency in the processing stage. In our PPEDP, Alice and Bob transmit in total \( \varepsilon = \bar{n}/(\tau \times \rho) \) ciphertext and \( m \) real numbers. The computation overhead is \( \delta = \bar{n}/\tau \) encryptions, \( \delta \) homomorphic additions and \( \varepsilon \) decryptions.

4 | ESkNN CLASSIFICATION PROTOCOL

In this section, we propose our ESkNN using PPEDP and OT. We also analyze the performance of our new protocol theoretically.

Alice has a private sample data set \([S \ Y]\). Bob has a private feature vector \( X \). Alice and Bob wish to predict the label of \( X \) based on the kNN algorithm with the training samples \([S \ Y]\). After the computation, Bob gets the label of \( X \), which is denoted by \( y \), while Alice outputs nothing. For the privacy concern, Bob never knows anything useful about \([S \ Y]\) while Alice cannot get \( X \) and \( y \).

For the sake of readability, we use the following notations. \( Y = [y_1 \ y_2 \ ... \ y_m]^T \) is the label set. \( X \) is the query of Bob with \( n \) real numbers denoted by \( x_{is} \). \( i \in \{1, 2, ..., n\} \). \( S \) consists of \( m \) sample feature vectors \( A_i \) while each feature vector is \( n \)-dimensional, that is, \( S = [A_1 \ A_2 \ ... \ A_m]^T \). Every element in \( S \) is a real number denoted by \( s_{ij}s \). \( i \in \{1, 2, ..., m\}, j \in \{1, 2, ..., n\} \).

For the convenience of FHE operations, we divided \( A_i \) and \( X \) into \( \delta \) parts separately. Each part consists of \( \tau \) feature values.

\[
A_i = [A_{i,1} \ A_{i,2} \ ... \ A_{i,\delta}], \ i \in \{1, 2, ..., m\}
\]

\[
A_{i,j} = [s_{i,j,\tau+1}s_{i,j,\tau+2} \ ... \ s_{i,j,\tau+\tau}], \ i \in \{1, 2, ..., m\}, \ j \in \{1, 2, ..., \delta\}
\]

\[
X = [X_1 X_2 \ ... X_\delta]
\]

\[
X_i = [x_{i,\tau+1} x_{i,\tau+2} \ ... x_{i,\tau+\tau}], \ i \in \{1, 2, ..., \delta\}
\]

4.1 ESkNN

Our ESkNN consists of three phases, that is, the preprocessing phase, the Euclidean distance computation phase and the label computation phase.
4.1.1 | Preprocessing phase

In the preprocessing phase, Alice and Bob compute as follows separately.

1) First, Alice generates the key pair \((pk, sk)\) of the BGV homomorphic scheme.

\[(pk, sk) \leftarrow \text{FHE. KeyGen}(1^3)\]

2) Second, Alice encrypts \(S\) using the public key \(pk\) as follows.

\[
\text{for } i \leftarrow 1 \text{ to } m \text{ do } \\
\text{for } j \leftarrow 1 \text{ to } \delta \text{ do } \\
E(A_{i,j}) \leftarrow \text{FHE. Enc}(pk, A_{i,j}) \\
E(A_i) \leftarrow [E(A_{i,1})E(A_{i,2})...E(A_{i,\delta})] \text{ } \text{for } i \text{ from } 1 \text{ to } m \text{.} \\
E(S) = [E(A_1)E(A_2)...E(A_m)]^T
\]

3) Third, Alice computes the sum of squared feature values in every row of \(S\).

\[V = [v_1v_2...v_m] s. t. \text{ } v_i = \sum_{j=1}^{n} s_{i,j}^2, i \in \{1, 2, ..., m\}\]

4) Then, Alice discloses \(pk\) and \(E(S)\). Once Alice disclosed \(pk\) and \(E(S)\), everyone can get them without any authentication.

5) At last, Bob gets \(pk\) and \(E(S)\).

4.1.2 | Euclidean distance computation phase

In the linear scan kNN algorithm, the \(k\) nearest neighbors are chosen by the top \(k\) minimum Euclidean distances between every samples and the query. In ESkNN, we do not compute the true Euclidean distances \(e_i = \sqrt{(s_{i,1}-x_1)^2+(s_{i,2}-x_2)^2+\cdots+(s_{i,n}-x_n)^2}, i \in \{1, 2, ..., m\}\). Instead, we only compute \(e_i' = \sum_{j=1}^{n} s_{i,j}^2 - \sum_{j=1}^{n} 2s_{i,j}x_j\).

1) Bob generates a matrix \(R\) with \(m \times \varepsilon \times \tau\) random numbers \(r_{i,j}, i \in \{1, 2, ..., m\}, j \in \{1, 2, ..., \varepsilon \times \tau\}\).

\[R = [R_1 R_2 ... R_m]^T\]

\[R_i = [R_{i,1} R_{i,2} ... R_{i,\varepsilon}] s. t. i \in \{1, 2, ..., m\}\]

\[R_{i,j} = [r_{i,j+\tau+1} r_{i,j+\tau+2} ... r_{i,j+\tau}] s. t. i \in \{1, 2, ..., m\}, j \in \{1, 2, ..., \varepsilon\}\]
2) Bob encrypts $R$ using the public key $pk$ as follows.

$$
\text{for } i \leftarrow 1 \text{ to } m \text{ do} \\
\quad \text{for } j \leftarrow 1 \text{ to } \varepsilon \text{ do} \\
\quad \quad E(R_{ij}) \leftarrow \text{FHE}. \text{Enc}(pk, R_{ij}) \\
\quad E(R_i) \leftarrow \left[ E(R_{i,1}) E(R_{i,2}) \ldots E(R_{i,\varepsilon}) \right] \\
E(R) = \left[ E(R_1) E(R_2) \ldots E(R_m) \right]^T
$$

3) Bob encrypts $X$ using the public key $pk$.

$$
E(X) = \left[ E(X_1) E(X_2) \ldots E(X_\delta) \right] \\
E(X_i) \leftarrow \text{FHE}. \text{Enc}(pk, X_i) \text{ s. t. } i \in \{1, 2, \ldots, \delta\}.
$$

4) Bob multiplies every ciphertext in $E(A_i)$ by every ciphertext in $E(X)$ as follows.

$$
\text{for } i \leftarrow 1 \text{ to } m \text{ do} \\
\quad \text{for } j \leftarrow 1 \text{ to } \delta \text{ do} \\
\quad \quad C_{ij} \leftarrow \text{FHE}. \text{CMult}(E(A_{ij}), E(X_j)) \\
\quad C_i \leftarrow \left[ C_{i,1} C_{i,2} \ldots C_{i,\delta} \right] \\
C = \left[ C_1 C_2 \ldots C_m \right]^T
$$

5) Bob does the homomorphic additive operation on every $\rho$ ciphertext in $C_i$ as follows.

$$
\text{for } i \leftarrow 1 \text{ to } m \text{ do} \\
\quad \text{for } j \leftarrow 1 \text{ to } \varepsilon \text{ do} \\
\quad \quad B_{ij} \leftarrow C_{ij,\rho} \\
\quad \quad \text{for } \kappa \leftarrow 1 \text{ to } \rho \text{ do} \\
\quad \quad \quad B_{ij} \leftarrow \text{FHE}. \text{CAdd}(B_{ij}, C_{ij,\rho+\kappa}) \\
\quad B_i \leftarrow \left[ B_{i,1} B_{i,2} \ldots B_{i,\varepsilon} \right] \\
B = \left[ B_1 B_2 \ldots B_m \right]^T
$$

6) Bob confuses $B$ using $E(R)$.

$$
\text{for } i \leftarrow 1 \text{ to } m \text{ do} \\
\quad \text{for } j \leftarrow 1 \text{ to } \varepsilon \text{ do} \\
\quad \quad D_{ij} \leftarrow \text{FHE}. \text{CAdd}(B_{ij}, E(R_{ij})) \\
\quad D_i \leftarrow \left[ D_{i,1} D_{i,2} \ldots D_{i,\varepsilon} \right] \\
D = \left[ D_1 D_2 \ldots D_m \right]^T
$$
7) Bob sends $D$ to Alice.
8) Alice receives $D$.
9) Alice decrypts $D$ using the secret key $sk$.
   for $i \leftarrow 1$ to $m$
   do:
   $\alpha = [\alpha_1,\alpha_2, \ldots, \alpha_\tau]$
   for $j \leftarrow 1$ to $\varepsilon$
   do:
   $\alpha \leftarrow \text{FHE} \cdot \text{Dec}(sk, E(D_{i,j}))$
   for $\kappa \leftarrow 1$ to $\tau$
   do:
   $W_i \leftarrow W_i + (2 \times \alpha_\kappa) \times 10^{-\sigma}$
   $W = [W_1, W_2, \ldots, W_m]^T$
10) Alice adds a random number to $V+W$
   $r \leftarrow \text{random}$
   for $i \leftarrow 1$ to $m$
   do:
   $Q_i \leftarrow V_i + W_i + r$
   $Q = [Q_1, Q_2, \ldots, Q_m]^T$
11) Alice sends $Q$ to Bob.
12) Bob receives $Q$.
13) Bob subtracts the random numbers generated in step1 form $Q$.
   for $i \leftarrow 1$ to $m \times \tau$
   do:
   for $j \leftarrow 1$ to $\varepsilon \times \tau$
   do:
   $r'_i \leftarrow r'_i + n_{i,j} \times 10^{-\sigma}$
   $T_i = Q_i - r'_i$
   $T = [T_1, T_2, \ldots, T_m]^T$

4.1.3 | Label computation phase

1) Bob finds the top $k$ minimum numbers in $T$. We use $P = [p_1, p_2, \ldots, p_k]$ to indicate the position of the $k$ minimum numbers in $T$.
2) Bob constructs a string $L$ with $m$ bits, where $L_i$ represents the $i$th bit in $L$.
   
   $$L_i = \begin{cases} 0 & \text{if } i \in P \\ 1 & \text{if } i \notin P \text{ s. t. } i \in \{1, 2, \ldots, m\} \end{cases}$$
3) Alice generates a tuple set $(Y, D)$, where $D = [d_1, d_2, \ldots, d_m]^T$ and $d_i$ is random numbers.
4) Alice and Bob invoke the OT protocol where Alice is the sender with the input $(Y, D)$ and Bob is the receiver with the input $L$. After the OT protocol, Bob outputs a vector $Y = \langle Y_1, Y_2, \ldots, Y_m \rangle$.
5) Bob chooses the labels of the $k$ nearest neighbors, that is, $\{Y_{p_1}, Y_{p_2}, \ldots, Y_{p_k}\}$.
6) Bob computes the output label $y$ based on the chosen decision rule using $\{Y_{p_1}, Y_{p_2}, \ldots, Y_{p_k}\}$.
4.2 Performance analysis

Similar to PPEDP, our ESkNN performs the kNN prediction correctly when \( \mu < \log_2(\sqrt{p/(\rho + 1)}) \).

In the processing phase, besides the data transmitted in the OT protocol, Alice and Bob transmit \( m \times \varepsilon = m \times n/(\tau \times \rho) \) ciphertext and \( m \) real numbers. The computation overhead is \( m \times \delta = m \times n/\tau \) encryptions, \( m \times \delta \) homomorphic additions, \( m \times \varepsilon \) decryptions and the computation overhead in the OT protocol.

5 IMPLEMENTATION AND PERFORMANCE RESULTS

We implement a PPkNN prediction system based on our protocol and show the experimental results in this section. We only evaluate the performance of our ESkNN because (1) a lot of previous works have been done for the performance evaluation of PPkNN prediction, and (2) PPEDP is achieved as part of ESkNN system.

The system is implemented in C++. We use the fully homomorphic cryptosystem HElib\textsuperscript{35} to handle the encryption, decryption, and homomorphic operations. OT is implemented using the EMP toolkit.\textsuperscript{36} The experiments are performed on a JD cloud c.n2.xlarge compute-optimized standard instance with an Intel\textregistered Xeon\textregistered Four-CoreTM CPU of a 2.4 GHz processor and 8 GB RAM. Each of the parties ran on a CPU core. We collected 10 runs for each test and report the average. We report experimental results only for the processing phase of our ESkNN because the preprocessing phase can be executed only one time for different queries. In our experiments, we use synthetic data sets to test our implementation. In the synthetic data set, the samples and the queries are generated randomly. Each feature value is two decimals between 0.00 and 0.99 after the normalization processing.

5.1 Results analysis

We start with the experimental results for our ESkNN in different settings. We initialized the BGV homomorphic scheme with the following parameters: (1) the cyclotomic polynomial \( phi(\omega) \) is set with \( \omega = 32767 \), (2) the plaintext prime modulus \( p \) is set to be 999007, (3) the number of bits of the modulus chain is set to be 590. With the parameters above, the security level \( \lambda \) yields to be 99, while the length of slots \( \tau \) is 180. According to Theorem 1, the maximum bucket length \( \rho \) is 100. We choose the parameters for three reasons. (1) The security level is similar to the comparing works. (2) To the extent allowed by hardware conditions, we want the plaintext space to be as large as possible. (3) To reduce the bandwidth, the expected length of slots \( \tau \) is 180 which is an empirical value.

First, to examine how the processing phase scales, we run experiments on data set with the dimension \( n \) of the feature vector increasing from 3600 to 18 000 and the bucket length \( \rho \) increasing from 20 to 100. As shown in Figure 2, when fixing \( m = 5, k = 3 \), the running time scales almost linearly with the dimension of the feature vector while the bucket length has less effect. Meanwhile, when fixing the dimension of the feature vector, the communication size mainly depends on the bucket length. When fixing the bucket length, the relationship between the communication size and the dimension of the feature vector can be expressed in a step function. In particular, by fixing \( k = 3, \rho = 100 \), it takes 646.2 s with the communication size to be 408 MB when classification with kNN model in a privacy-preserving manner on five samples with 18 000 features each. It is obviously that, in this scenario, to make full use of resources, the
bucket length should set to be 100. If fact, the bucket length is set to be a const when using our implementation in practice. Usually, to make full use of the communication resources, the bucket length should be set to the maximum effective value according to Theorem 1.

Next, we run experiments on data set with the sample size \( m \) increasing from 20 to 60 and the number \( k \) of the nearest neighbors increasing from 3 to 11. Currently we could not implement our protocol for larger sample size due to large memory requirements. But if one can manage large memory, it is expected that our protocol would run on large parameters. As shown in Figure 3, the running time and the communication size are independent of the number \( k \) of the nearest neighbors. When fixing the dimension of the feature vector and bucket length, the running time and the communication size grow linearly with the sample size. In particular, by fixing \( m = 60, n = 360, \) and \( \rho = 100 \), our protocol takes 324.1 s with the communication size to be 981 MB.

We also observe that with the optimized bucket length and a fixed data set size, that is, \( m \times n \), the overhead of our protocol is much lower when the sample data set has many more features than samples, that is, \( n \gg m \). In particular, we analyze the situation by fixing the data set size to be 18 000 and the bucket length to be 100. When \( m = 50, n = 360 \), our protocol takes 270.8 s with the communication size to be 817 MB. Meanwhile, it takes 127.3 s with the communication size to be 81 MB by fixing \( m = 5, n = 3600 \).

In conclusion, with fixed \( k, \rho, \tau \), we get that for our ESkNN, (1) the computation overhead which is evaluated by the running time scales almost linearly with the dimension of the feature vector and the sample size; (2) the communication size go in a step function with the dimension of the feature vector, while it grows linearly with the sample size; and (3) our protocol is more suitable for the scenario where the sample data set has many more features than samples. These observations agree with the asymptotic complexity derived above.

5.2 | Comparison with prior works

As surveyed in the related works, PPkNN classification protocols are also considered by PPkNN\(^{21}\) and outsourced collaborative kNN (OckNN)\(^{22}\) based on the same distance metric.
For comparison, we use the results reported in the work directly as their security levels are nearly equal to ours and their running environment is more powerful than ours. PPkNN and OCkNN are tested on 4898 samples with 12 features each. Our protocol is tested on 33 samples with 1800 features each. It is obviously that the data set sizes are nearly equal. In fact, our data set size is a little bit more than theirs.

The running time and communication size of both PPkNN and OCkNN increases with the growth of $k$, while our overhead is independent of $k$. As shown in Figure 4, compared with PPkNN and OCkNN, the computation overhead which is evaluated by the running time is improved significantly in ESkNN. For example, compared with PPkNN, when $k = 25$, our protocol has a $402 \times$ speedup. Compared with OCkNN, when $k = 25$, our protocol has a $17 \times$ speedup.

Our communication size is significantly smaller than PPkNN’s. Compared with OCkNN, our communication overhead is lower than OCkNN’s when $k > 15$.

In addition, our ESkNN can work with high dimensional features given in real numbers while PPkNN and OCkNN mainly focus on low dimensional features represented in integers.

6 CONCLUSION

In this paper, we presented a PPEDP, which enabled us to compute the Euclidean distance between two points given in real numbers in a high dimensional space without worrying about privacy leakage. Based on PPEDP and OT, we designed an ESkNN classification protocol, ESkNN. Experiment results showed that our ESkNN dealt with real numbers, performed well in the scenario where the sample data set had many more features than samples and run faster than existing works.

We note that our protocol has the following limitations. First, our data conversion algorithm is highly dependent on the operations performed on the ciphertext. Our future work will look to design a data conversion algorithm working in a black box manner with the fully homomorphic
scheme. Second, with the fixed BGV homomorphic parameters, the overhead of our protocol depends on the dimension of the feature vector, the sample size, the slot length and the bucket length. The overhead of our protocol increases sharply when the sample data set had many more samples than features. At last, our protocol can only deal with plaintext which has two digits in our hardware limited experiment environment. We would like to design a protocol practical for larger digits of the plaintext in the future.

ACKNOWLEDGMENTS
This study was partially supported by Project of High-level Teachers in Beijing Municipal Universities in the Period of 13th Five-year Plan (CIT&TCD201904097) and CUEB Supporting Plan for the Youth Backbone Teachers.

CONFLICT OF INTERESTS
The authors declare that there are no conflict of interests.

REFERENCES
1. Lin KP, Chen MS. Privacy-preserving outsourcing support vector machines with random transformation. In Proceedings of the 16th International Conference on Knowledge Discovery and Data Mining; 2010:63-372.
2. Lin KP, Chen MS. On the design and analysis of the privacy-preserving SVM classifier. IEEE T Knowl Data Eng. 2011;23(11):1704-1717.
3. Teo SG, Han S, Lee VC. Privacy preserving support vector machine using non-linear kernels on hadoop mahout. In Proceedings of IEEE 16th International Conference; 2013:941-948.
4. Makri E, Rotaru D, Smart NP, Vercauteren F. PICS: Private Image Classification with SVM. 2017. https://eprint.iacr.org/2017/1190
5. Rahulamathavan Y, Phan RCW, Veluru S, Cumanan K, Rajarajan M. Privacy-preserving multi-class support vector machine for outsourcing the data classification in cloud. IEEE T Depend Secure. 2014;11(5):467-479.
6. Laur S, Lipmaa H, Mielič-ainen T. Cryptographically private support vector machines. In Proceedings of the 12th International Conference on Knowledge Discovery and Data Mining; 2006:618-624.
7. Vaidya J, Yu H, Jiang X. Privacy-preserving SVM classification. Knowl Inf Syst. 2008;14(2):161-178.
8. Barnett A, Santokhi J, Simpson M, et al. Image classification using non-linear support vector machines on encrypted data; 2017. https://eprint.iacr.org/2017/857
9. Cock MD, Dowsley R, Nascimento ACA, Railsback D, Shen JW, Todoki A. High performance logistic regression for privacy-preserving genome analysis; 2020. https://eprint.iacr.org/2020/171
10. Mohassel P, Zhang YP. SecureML: a system for scalable privacy-preserving machine learning. In Proceedings of IEEE Symposium on Security and Privacy. 2017;19-38.
11. Hong C, Huang ZC, Lu WJ, Qu H. Privacy-preserving collaborative machine learning on genomic data using TensorFlow; 2020. https://eprint.iacr.org/2020/159
12. El-Emam K, Samet S, Arbuckle L, Tamblyn R, Earle C, Kantarcioglu M. A secure distributed logistic regression protocol for the detection of rare adverse drug events. J Am Med Inform Asm. 2012;20(3):453-461.
13. Nardi Y, Fienberg SE, Hall RJ. Achieving both valid and secure logistic regression analysis on aggregated data from different private sources. J Privacy Confidentiality. 2012;4(1):189-220.
14. Xie W, Wang Y, Boker SM, Brown DE. Privlogit: Efficient privacy-preserving logistic regression by tailoring numerical optimizers; 2016. https://arxiv.org/abs/1611.01170
15. Rouhani BD, Riazi MS, Koushanfar F. Deepsecure: scalable provably-secure deep learning. In Proceedings of Design Automation Conference; 2018.
16. Riazi MS, Weinert C, Tkachenko O, Songhori EM, Schneider T, Koushanfar F. Chameleon: a hybrid secure computation framework for machine learning applications. In Proceedings of the 18th ACM ASIACCS; 2018.
17. Juvekar C, Vaikuntanathan V, Chandrakasan A. GAZELLE: a low latency framework for secure neural network inference; 2018. https://eprint.iacr.org/2018/073
18. Liu J, Juuti M, Lu Y, Asokan N. Oblivious neural network predictions via minionn transformations. In Proceedings of 2017 ACM SIGSAC Conference on Computer and Communications Security; 2017;619-631.
19. Gilad-Bachrach R, Dowlin N, Laine K, Lauter K, Naehrig M, Wernsing J. CryptoNets: applying neural networks to encrypted data with high throughput and accuracy. In Proceedings of International Conference on Machine Learning; 2016;201-210.
20. Bourse F, Minelli M, Minihold M, Paillier P. Fast homomorphic evaluation of deep discretized neural network; 2017. https://eprint.iacr.org/2017/1114
21. Samanthula BK, Elmehdwi Y, Jiang W. K-nearest neighbor classification over semantically secure encrypted relational data. IEEE T Knowl Data Eng. 2015;27(5):1-14.
22. Rong H, Wang HM, Liu J, Xian M. Privacy-preserving k-nearest neighbor computation in multiple cloud environments. IEEE Access. 2016;4:9589-9603.
23. Brakerski Z, Gentry C, Vaikuntanathan V. Fully homomorphic encryption without boot-strapping. Theor Comput Sci. 2012;309-325.
24. Elmehdwi Y, Samanthula BK, Jiang W. Secure k-nearest neighbor query over encrypted data in outsourced environments. In Proceedings of IEEE 30th International Conference on Data Engineering; 2014;664-675.
25. Erkin Z, Franz M, Guajardo S, Lagendijk I, Toft T. Privacy-preserving face recognition. In Proceedings of Privacy Enhancing Technologies Symposium; 2009;235-253.
26. Evans D, Huang Y, Katz J, Malka L. Efficient privacy-preserving biometric identification. In Proceedings of Network and Distributed System Security Symposium (NDSS); 2011.
27. Demmler D, Schneider T, Zohner M. Aby-a framework for efficient mixed-protocol secure two-party computation. In Proceedings Network and Distributed System Security Symposium (NDSS); 2015.
28. Schoppmann P, Gascón A, Balle B. Private nearest neighbors classification in federated databases; 2018. https://eprint.iacr.org/2018/289
29. Songhori EM, Hussain SU, Sadeghi AR, Koushanfar F. Compacting privacy-preserving k-nearest neighbor search using logic synthesis. In Proceedings of Design Automation Conference; 2015;1-6.
30. Asharov G, Halevi S, Lindell Y, Rabin T. Privacy-preserving search of similar patients in genomic data. Privacy Enhancing Technol. 2018;4:104-124.
31. Shaull H, Feldman D, Rus D. Scalable secure computation of statistical functions with applications to k-nearest neighbors; 2018. https://arxiv.org/abs/1801.07301
32. Chen H, Chilotti I, Dong YH, Poburinnaya O, Razenshteyn I, Riazi MS. SANNS: Scaling Up Secure Approximate k-Nearest Neighbors Search; 2019. https://eprint.iacr.org/2019/359
APPENDIX

**Theorem 2.** In the honest-but-curious model, assuming that the underlying fully homomorphic encryption (FHE) scheme is secure, our privacy-preserving euclidean distance protocol (PPEDP) calculates the Euclidean distance securely.

**Proof.** First, we prove the security of PPEDP in case Alice is corrupted. In this section, we use protocol $\pi$ to represent PPEDP. During the execution of $\pi$, the view of Alice is 

$$\text{view}^{\pi}_{\text{Alice}}(x, y) = \{x, f_1(x, y), (pk, sk), E(X), C, \alpha, w, vl\},$$

where $f_1(x, y) = \bot$.

Now, we construct a simulator $S_1$ so that $S_1$ receives Alice’s input and output information, that is, $(x, f_1(x, y)) = (x, \bot)$. $S_1$ simulates the execution of protocol $\pi$ as follows.

1) $S_1$ constructs a set $x' = x$.
2) $S_1$ constructs a set $y' = \{y'_1, y'_2, \ldots, y'_m\}$, where $y'_i$ s. t. $i \in \{1, 2, \ldots, m\}$ is a random number.
3) $S_1$ calls the key generation algorithm of the BGV homomorphic scheme to generate the key pairs $(pk', sk')$.
4) $S_1$ encrypts $x'$ using the public key $pk'$ and gets the ciphertext $E(X')$.
5) $S_1$ encrypts $y'$ by using the public key $pk'$ and get $E(y')$.
6) $S_1$ performs multiplication and addition on ciphertext to obtain a ciphertext set $C'$.

for $i \leftarrow 1$ to $\varepsilon$ do

for $j \leftarrow 1$ to $\rho$ do

$$\Delta C' \leftarrow \text{FHE} \cdot \text{CMult}(E(A'_i), E(B'_j))$$

$$C'_i \leftarrow \text{FHE} \cdot \text{CAdd}(C'_i, \Delta C')$$

$$n' \leftarrow \text{random}$$

$$E(n') \leftarrow \text{FHE} \cdot \text{Enc}(pk', n')$$

$$C'_i \leftarrow \text{FHE} \cdot \text{CAdd}(C'_i, E(n'))$$

$$C' = \{C'_1, C'_2, \ldots, C'_\varepsilon\}$$

---

**How to cite this article:** Sun M, Yang R. An efficient secure k nearest neighbor classification protocol with high-dimensional features. *Int J Intell Syst*. 2020;35:1791-1813. https://doi.org/10.1002/int.22272
7) \( S_1 \) decrypts \( C' \) using the private key \( sk' \) to obtain \( v' \).

\[
\alpha' = [\alpha'_1 \alpha'_2 \ldots \alpha'_r]
\]

\[
v' = 10^{-\sigma} \times \sum_{i=1}^{n} x'_i^2
\]

for \( i \leftarrow 1 \) to \( \varepsilon \) do

\[
\alpha' \leftarrow \text{FHE}. \text{Dec}(sk', C'_i)
\]

for \( j \leftarrow 1 \) to \( \rho \) do

\[
v' \leftarrow v' - (2 \times \alpha'_j) \times 10^{-\sigma}
\]

Because \( S_1 \) simulates \( \pi \) in case Alice is corrupted, the view of \( S_1 \) is \( \text{view}_{\text{sim}}(x, y) = \{x', f_{\text{sim}}(x, y), (pk', sk'), E(X'), C', \alpha', w', v'\} \). Assuming the underlying homomorphic encryption scheme is secure in the honest-but-curious model, we have \( (pk', sk') \equiv (pk, sk), E(X') \equiv E(X), C' \equiv C \). As \( x' = x \), we have \( w' = w \). As \( \eta' \) and \( \eta \) chosen randomly, we have \( \eta' \equiv \eta \) and \( \alpha' \equiv \alpha \). As \( w' = w \) and \( \alpha' \equiv \alpha \), we have \( v' \equiv v \).

Now, we can get the conclusion that

\[
\{x', f_{\text{sim}}(x, y), (pk', sk'), E(X'), C', \alpha', w', v'\} \equiv \{x, f_1(x, y), (pk, sk), E(X), C, \alpha, w, v\},
\]

i.e., \( \text{view}_{\text{sim}}^\pi(x, y) \equiv \text{view}_{\text{Alice}}^\pi(x, y) \).

Next, we prove the security of PPEDP in case Bob is corrupted. During the execution of protocol \( \pi \), the view of Bob is \( \text{view}_{\text{Bob}}^\pi(x, y) = \{y, f_2(x, y), pk, E(x), E(y), C, v\} \), where \( f_2(x, y) = e \).

Now, we construct a simulator \( S_2 \) so that \( S_2 \) receives the input and output of Bob, that is \( (y, f_2(x, y)) = (y, e) \). Now, \( S_2 \) simulates the execution of protocol \( \pi \) as follows.

1) \( S_2 \) constructs a set \( y' \) such that \( y' = y \).
2) \( S_2 \) constructs a set \( x' = \{x'_1, x'_2, \ldots, x'_m\} \), where \( x'_1, x'_2, \ldots, x'_{m-1} \) are random numbers. \( x'_m \) is calculated by the following formula.

\[
|x'_m - y'_m| = \sqrt[2]{10^{-\sigma} \times e^2 - (x'_1 - y'_1)^2 + (x'_2 - y'_2)^2 + \cdots (x'_{m-1} - y'_{m-1})^2}
\]

3) \( S_2 \) calls the key generation algorithm of the BGV homomorphic scheme to generate the key pairs \( (pk', sk') \).
4) \( S_2 \) encrypts \( x' \) using the public key \( pk' \) and gets the ciphertext \( E(x') \).
5) \( S_2 \) encrypts \( y' \) by using the public key \( pk' \) and get \( E(y') \).
6) \( S_2 \) performs multiplication and addition on ciphertext to obtain \( C' \).

for \( i \leftarrow 1 \) to \( \varepsilon \) do

for \( j \leftarrow 1 \) to \( \rho \) do

\[
\Delta C' \leftarrow \text{FHE}. \text{CMult}(E(A'_i), E(B'_j))
\]

\[
C'_i \leftarrow \text{FHE}. \text{CAdd}(C'_i, \Delta C')
\]

\[
\eta' \leftarrow \text{random}
\]

\[
E(\eta') \leftarrow \text{FHE}. \text{Enc}(pk', \eta')
\]

\[
C'_i \leftarrow \text{FHE}. \text{CAdd}(C'_i, E(\eta'))
\]

\[
C' = <C'_1, C'_2, \ldots, C'_\varepsilon>
\]
7) $S_2$ decrypts $C'$ using the private key $sk'$ to obtain $v'$.

$$\alpha' = [\alpha'_1 \alpha'_2 \ldots \alpha'_r]$$

$$v' = 10^{-\sigma} \times \sum_{i=1}^{n} x'_i^2$$

for $i \leftarrow 1$ to $\varepsilon$ do

$$\alpha' \leftarrow \text{FHE . Dec}(sk', C')$$

for $j \leftarrow 1$ to $\tau$ do

$$v' \leftarrow v' - (2 \times \alpha'_j) \times 10^{-\sigma}$$

8) $S_2$ calculates the Euclidean distance $e' = \sqrt{10^{-\sigma} \times \sum_{i=1}^{n} y'_i^2 + v' - 10^{-\sigma} \times \sum_{i=1}^{\varepsilon} r'_i}$.

Because $S_2$ simulates $\pi$ in case Bob is corrupted, the view of $S_2$ is $\text{view}_{\text{sim}}(x, y) = \{y', f_{\text{sim}}(x, y), pk', E(x'), E(y'), C', v'\}$. Assuming the underlying homomorphic encryption scheme is secure in the honest-but-curious model, we can get $E(x') \equiv E(x)$, $E(r') \equiv E(r')$, $E(y') \equiv E(y)$, $C' \equiv C$. As $r'_i$ and $\eta$ follow the same probability distribution, we can get $r' \equiv r$. As $v'$ and $v$ are confused by $r'$ and $r$, respectively as shown in the following formula, we have $v' \equiv v$.

$$v' = 10^{-\sigma} \times \sum_{i=1}^{n} x'_i^2 - 2 \times 10^{-\sigma} \times \sum_{i=1}^{n} x'_i \times y'_i + 10^{-\sigma} \times \sum_{i=1}^{\varepsilon} r'_i$$

$$v = 10^{-\sigma} \times \sum_{i=1}^{n} x_i^2 - 2 \times 10^{-\sigma} \times \sum_{i=1}^{n} x_i \times y_i + 10^{-\sigma} \times \sum_{i=1}^{\varepsilon} r_i$$

As we assumed $|x'_m - y'_m| = \sqrt{10^{-\sigma} \times e^2 - (x'_1 - y'_1)^2 + (x'_2 - y'_2)^2 + \cdots (x'_{m-1} - y'_{m-1})^2}$, we can get the following formula:

$$e' = \sqrt{10^{-\sigma} \times \sum_{i=1}^{n} y'_i^2 + v' - 10^{-\sigma} \times \sum_{i=1}^{\varepsilon} r'_i}$$

$$= \sqrt{10^{-\sigma} \times \sum_{i=1}^{n} y_i^2 + 10^{-\sigma} \times \sum_{i=1}^{n} x_i^2 - 2 \times 10^{-\sigma} \times \sum_{i=1}^{n} x_i \times y_i} = e.$$

Now, we can get the conclusion that

$$\{y', f_{\text{sim}}(x, y), pk', E(x'), E(y'), C', v'\} \equiv \{y, f_2(x, y), pk, E(x), E(y), C, v\}$$

$$\text{view}_{\text{sim}}(x, y) \equiv \text{view}_{\text{Bob}}(x, y)$$

**Theorem 3.** In the honest-but-curious model, assuming that the underlying FHE scheme and the OT protocol is secure, ESkNN calculates the kNN prediction securely.

**Proof** Now, we prove the security of ESkNN in case Alice is corrupted. In this section, we use protocol $\pi$ to represent ESkNN. During the execution of protocol $\pi$, Alice’s view is $\text{view}_{\text{Alice}}([SY], X) = \{[SY], f_1([SY], X), (pk, sk), E(S), D, \alpha, W, V, r, Q, \text{view}_{\text{Alice}}^\text{OT}((Y, D), L)\}$, where $f_1([SY], X) = \bot$.

Now, we construct a simulator $S_1$ so that $S_1$ receives Alice’s input and output information, that is, $([SY], f_1([SY], X)) = ([SY], \bot)$. $S_1$ simulates the execution of protocol $\pi$ as follows.
1) $S_1$ constructs matrix $S' = S$ and $Y' = Y$.
2) $S_1$ constructs a matrix $X' = [X_i' X_{i+1}' \cdots X_{i+\tau}']$, where $X_i' = [x_{i+\tau+1}' x_{i+\tau+2}' \cdots x_{i+\tau+\tau}' \ldots \cdots]$. $x_i'$'s. t. $i \in \{1, 2, \ldots, n\}$ is a random number.
3) $S_1$ generates the key pair $(pk', sk')$ of the BGV homomorphic scheme.
   
   $$(pk', sk') \leftarrow \text{FHE. KeyGen}(1^\lambda)$$

4) $S_1$ encrypts $S'$ using the public key $pk'$ and gets $E(S')$.

   for $i \leftarrow 1$ to $m$ do
   for $j \leftarrow 1$ to $\delta$ do
   
   $E(A_{ij}') \leftarrow \text{FHE. Enc}(pk', A_{ij})$

   $E(A_i') \leftarrow [E(A_{1i}') E(A_{2i}') \cdots E(A_{\delta i}')]$

   $E(S') = [E(A_1') E(A_2') \cdots E(A_m')]^T$

5) $S_1$ computes the sum of squared feature values in every row of $S'$ and gets $V' = [v_1' v_2' \cdots v_m']$.

   $$v_i' = \sum_{j=1}^{n} s_{ij}^2 \text{s. t. } i \in \{1, 2, \ldots, m\}$$

6) $S_1$ generates a matrix $R'$ with $m \times \varepsilon \times \tau$ random numbers $r_i' j, i \in \{1, 2, \ldots, m\}, j \in \{1, 2, \ldots, \varepsilon \times \tau\}$.

   $$R' = [R_1' R_2' \cdots R_m']^T$$

   $$R_i' = [R_{i,1}' R_{i,2}' \cdots R_{i,\varepsilon}' \cdots R_{i,\tau}']$$

   $$R_{ij}' = [r_{i,\tau+1}' r_{i,\tau+2}' \cdots r_{i,\tau+\tau}']$$

7) $S_1$ encrypts $R'$ using the public key $pk'$ and gets $E(R')$.

   for $i \leftarrow 1$ to $m$ do
   for $j \leftarrow 1$ to $\varepsilon$ do
   
   $E(R_{ij}') \leftarrow \text{FHE. Enc}(pk', R_{ij})$

   $E(R_i') \leftarrow [E(R_{1i}') E(R_{2i}') \cdots E(R_{\varepsilon i}')]$

   $E(R') = [E(R_1') E(R_2') \cdots E(R_m')]^T$

8) $S_1$ encrypts $X'$ by using the public key $pk'$ and gets $E(X')$.

   $$E(X') = [E(X_1') E(X_2') \cdots E(X_\delta')]$$

   $E(X_i') \leftarrow \text{FHE. Enc}(pk', X_i')$ s. t. $i \in \{1, 2, \ldots, \delta\}$
9) $S_1$ multiplies every ciphertext in $E(A'_i)$ by every ciphertext in $E(X'_i)$ and gets $C'_i$.

$$
\text{for } i \leftarrow 1 \text{ to } m \text{ do}
\text{for } j \leftarrow 1 \text{ to } \delta \text{ do}
C'_{i,j} \leftarrow \text{FHE. CMult}(E(A'_i), E(X'_j))
C'_i \leftarrow [C'_{i,1}C'_{i,2}\cdots C'_{i,\delta}]
C' = [C'_1C'_2\cdots C'_m]^T
$$

10) $S_1$ does the homomorphic additive operation on every $\rho$ ciphertext in $C'_i$ to obtain $B'_i$.

$$
\text{for } i \leftarrow 1 \text{ to } m \text{ do}
\text{for } j \leftarrow 1 \text{ to } \varepsilon \text{ do}
B'_{i,j} \leftarrow C'_{i,j,\rho}
\text{for } \kappa \leftarrow 1 \text{ to } \rho \text{ do}
B'_{i,j} \leftarrow \text{FHE. CAdd}(B'_{i,j}, C'_{i,j,\rho+\kappa})
B'_i \leftarrow [B'_{i,1}B'_{i,2}\cdots B'_{i,\varepsilon}]
B' = [B'_1B'_2\cdots B'_m]^T
$$

11) $S_1$ confuses $B'$ using $E(R')$ and gets $D$.

$$
\text{for } i \leftarrow 1 \text{ to } m \text{ do}
\text{for } j \leftarrow 1 \text{ to } \varepsilon \text{ do}
D'_{i,j} \leftarrow \text{FHE. CAdd}(B'_{i,j}, E(R'_i))
D'_i \leftarrow [D'_{i,1}D'_{i,2}\cdots D'_{i,\varepsilon}]
D' = [D'_1D'_2\cdots D'_m]^T
$$

12) $S_1$ decrypts $D'$ using the secret key $sk'$ and gets $W'$.

$$
\text{for } i \leftarrow 1 \text{ to } m \text{ do}
\alpha' = [\alpha'_1\alpha'_2\cdots\alpha'_\varepsilon]
\text{for } j \leftarrow 1 \text{ to } \varepsilon \text{ do}
\alpha' \leftarrow \text{FHE. Dec}(sk', E(D'_i))
\text{for } \kappa \leftarrow 1 \text{ to } \tau \text{ do}
W'_i \leftarrow W'_i + (2 \ast \alpha'_\kappa) \times 10^{-\tau}
W' = [W'_1W'_2\cdots W'_m]^T
$$

13) $S_1$ adds a random number to $V' + W'$ and gets $Q'$.

$$
r \leftarrow \text{random}
\text{for } i \leftarrow 1 \text{ to } m \text{ do}
Q'_i \leftarrow V'_i + W'_i + r'
Q' = [Q'_1Q'_2\cdots Q'_m]^T$$
14) $S_1$ subtracts the random numbers $t_i$ form $Q'$ and gets $T'$.

$$\text{for } i \leftarrow 1 \text{ to } m \text{ do}$$
$$\text{for } j \leftarrow 1 \text{ to } \varepsilon \times \tau \text{ do}$$
$$t_i \leftarrow t_i + r'_{i,j} \times 10^{-\sigma}$$

$$T'_i = Q'_i - t_i$$

$$T' = [T'_1T'_2 \cdots T'_m]^T$$

15) $S_1$ finds the top $k$ minimum numbers in $T'$. We use $P' = [p'_1 \ p'_2 \cdots p'_k]$ to indicate the position of the $k$ minimum numbers in $T'$.

16) $S_1$ constructs a string $L'$ with $m$ bits. $L'_i$ is the $i$th bit in $L'$.

$$L'_i = \begin{cases} 0 & \text{if } i \in P' \\ 1 & \text{if } i \notin P' \end{cases} \text{ s. t. } i \in \{1, 2, \cdots, m\}$$

17) $S_1$ generates a tuple set $(Y', D')$, where $D' = [d'_1d'_2 \cdots d'_m]^T$ and $d'_i$ is random numbers.

18) $S_1$ invokes the OT protocol including the sender with the input $(Y', D')$ and the receiver with the input $L'$. After the OT protocol, $S_1$ outputs a vector $Y' = \{Y'_1, Y'_2, \cdots, Y'_m\}$.

19) $S_1$ gets the label of the $k$ nearest neighbors $\{Y'_p, Y'_{p_2}, \cdots, Y'_{p_k}\}$.

20) $S_1$ gets the label $y'$ by a decision rule using $\{Y'_p, Y'_{p_2}, \cdots, Y'_{p_k}\}$.

Because $S_1$ simulates $\pi$ in case Alice is corrupted, the view of $S_1$ is $\text{view}_{\text{sim}}^\pi ([SY], X) = \{[S' Y'], f_{\text{sim}}([SY], X), (pk', sk'), E(S'), D', \alpha', W', V', r', Q', \text{view}_{\text{sim}}^{\text{OT}}((Y', D'), L')\}$.

Assuming the underlying homomorphic encryption scheme is secure in the honest-but-curious model, we can get $(pk', sk') \equiv (pk, sk), E(S') \equiv E(S), \text{ and } D' \equiv D$. As $S' = S$, we have $V' = V$. As and $R'$ and $R$ are matrices with random numbers, we have $R' \equiv R, \alpha' \equiv \alpha$ and $W' \equiv W$. As and $r'$ and $r$ chosen randomly, we have $r' \equiv r$ and $Q' \equiv Q$.

Assuming the OT protocol is secure in the honest-but-curious model, we can get $\text{view}_{\text{sim}}^{\text{OT}}((Y', D'), L') \equiv \text{view}_{\text{Alice}}^{\text{OT}}((Y, D), L)$.

Now, we can get the conclusion that

$$\{[S' Y'], f_{\text{sim}}([SY], X), (pk', sk'), E(S'), D', \alpha', W', V', r', Q', \text{view}_{\text{sim}}^{\text{OT}}((Y', D'), L')\}$$

$$\equiv \{[SY], f_{\text{}}([SY], X), (pk, sk), E(S), D, \alpha, W, V, r, Q, \text{view}_{\text{Alice}}^{\text{OT}}((Y, D), L),$$

$$\text{view}_{\text{sim}}^\pi ([SY], X) \equiv \text{view}_{\text{Alice}}^\pi ([SY], X).$$

We can get the same conclusion when Bob is corrupted. For simplicity, we do not show the details.