Privacy-Preserving Approximate Nearest Neighbor Search: A Construction and Experimental Results

Huang Ke[1,a)] Satsuya Ohata[2] Kanta Matsuura[1]

Abstract: Secure multi-party computation (MPC) allows a set of parties to jointly compute a function, while keeping their inputs private. MPC has many applications, and we focus on privacy-preserving nearest neighbor search (NNS) in this paper. The purpose of the NNS is to find the closest vector to a query from a given database, and NNS arises in many fields of applications such as computer vision. Recently, some approximation methods of NNS have been proposed for speeding up the search. In this paper, we consider the combination between approximate NNS based on "short code" (searching with quantization) and MPC. We implement a short code-based privacy-preserving approximate NNS on secret sharing-based secure two-party computation and report some experimental results. These results help us to explore more efficient privacy-preserving approximate NNS in the future.

1. Introduction

1.1 Introduction

The nearest neighbor search (NNS) is a kind of algorithm that finds a vector in a given dataset which is closest to a given vector. We can apply this algorithm in many different fields, such as machine learning, computer vision, data mining, etc. With the continuous expansion of data scale, the demand for data scale on NNS has reached millions or even higher. To make search faster, we sometimes use some approximation methods such as short-code. In this strategy, we compress the vector into lower dimensions to reduce computation costs. In such approximation algorithms, the results may not always be correct. However, but we can expect that we can obtain the results that close to the correct answer in many cases. The NNS with approximation is called approximate nearest neighbor search(ANN).

In this paper, we consider the privacy of the query and database in this setting. We can easily find the cases that we query the privacy-sensitive data (e.g., picture of our face, genome strings) to the server and search for similar ones. To the best of our knowledge, there is no previous result of combining ANN with secure multi-party computation (secure computation, or MPC) [1], [2]. A privacy-preserving NNS may dramatically be faster by allowing small errors.

1.2 Related Work

There are some ways (= building blocks) to realize MPC such as (fully) homomorphic encryption [3], [4], garbled circuits [1], etc. In this paper, we focus on the MPC based on secret sharing (SS) [2] since we can obtain large throughput in this scheme. There are many research results on SS-based MPC. For examples, we have results on highly-efficient MPC (e.g., [5], [6]), concrete tools or the toolkit (e.g., [7], [8], [9], [10], [11]), mixed-protocol framework [12], [13], [14], application to privacy-preserving machine learning or data analysis (e.g., [13], [14], [15], [16]), proposal of another model for speeding up the pre-computation [15], [17], etc.

There are some research results on privacy-preserving (k-)NNS [12], [18], [19], [20], [21], [22], [23]. The results other than [12] consider the protocols based on homomorphic encryption or garbled circuit, and other than [22], [23] consider not approximate but exact (k-)NNS.

1.3 Our Contribution

In this paper, we proposed the construction of privacy-preserving approximate nearest neighbor search via secret sharing-based secure two-party computation. We find that the "short code" scheme in ANN algorithm has good compatibility with secure computation when computing the distance between vectors. In the original NNS, the computation of the distance between two vectors requires heavy operations such as rooting and squaring in secure computation. After using short code, distance computation can be transformed into XOR operation. Since we can compute XOR operation without interactions between parties, we can greatly reduce the costs for computation and communication of the algorithm. We consider the client-aided two-party computation in this paper. In this construction, two parties act servers, and they hold a share of database vectors. In this model, the client who wants to search the

[1] Institute of Industrial Science, The University of Tokyo
[2] National Institute of Advanced Industrial Science and Technology (AIST)
[a)] hk94@iis.u-tokyo.ac.jp

nearest vector with queries bears the pre-computation.

We make our experiments in local and LAN environments. We measure the running time of the algorithm. We set the data scale as 1000 to 10000 in our experiments. The results show that the time of distance calculation is linear with the number of data in the database when using short code approximation. The bottleneck of total execution time at present is the top-1 selection part (the operation for finding the shortest distance in a set of distances). The experimental results can help us to construct more efficient privacy-preserving ANN.

1.4 Paper Organization

In Section 2, we will introduce some basic knowledge needed for this paper. In Section 3, we will explain our construction of privacy-preserving ANN. In Section 4, we will explain the contents of the experiment and show their results. Section 5 is a conclusion and future work.

2. Preliminary

2.1 Approximate Nearest Neighbor Search

The algorithm of NNS is summarized as follow:

- Given a vector V and a vector dataset $S \in \{V_1, V_2, \ldots, V_n\}$
- Return a vector in S that is the closest to V.

The NNS arises in many fields of applications such as computer vision. To find a similar image, we convert the image data to the RGB vector. Then we use the NNS to find the nearest vector, which is represented to the similar image. In the current research, the scale of the dataset we use has expanded to millions or even billions. In these scales, the general NNS has been unable to maintain high efficiency. In order to achieve fast NNS with a large amount of data, some approximation algorithms are usually used. The locality sensitive hashing is a kind of solution for ANN search, and there are also many studies on LSH, but when combined with the secure computation in this paper, LSH needs a large amount of computation. In order to reduce the amount of computation, the approximate scheme used in this paper is the short code method.

2.2 Short Code

The current research on short code methods can be divided into the Hamming type and lookup type, and we use the Hamming type in our research. The short code methods based on Hamming type reduces the dimension of the original vector by using multiple hash functions and compresses the vector into a Boolean vector in which all the elements are Boolean values. In further explanation, we first need to define a parameter k, which represents the dimension of the compressed Boolean vector. In related research of short code-based ANN, k is set to 128. After defining the value of k, we need to create k hash functions. Each hash function takes the whole original vector as input and outputs a Boolean value of 0 or 1. Fig.1 shows the compression process of the short code method. Compressed vectors reduce

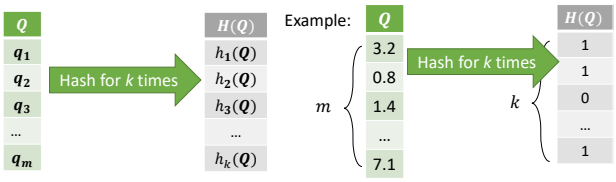

Fig. 1 Compression process of the short code method with parameter k. It hashes the original m-dimension vector $Q = \{q_1, q_2, ..., q_m\}$ into a k-dimension Boolean vector $H(Q) = \{H_1(Q), H_2(Q), ..., H_k(Q)\}$.For $i \in \{1, 2, ..., k\}$, $H_i(Q) \in \{1, 2, ..., k\}$

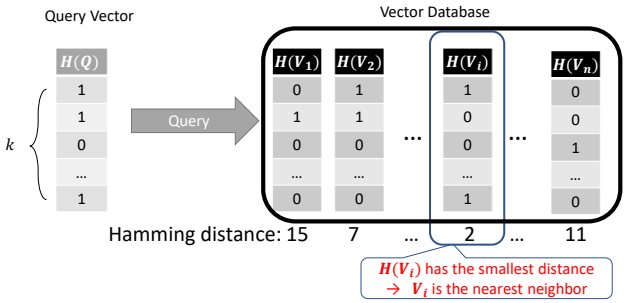

Fig. 2 Short code-based approximate nearest neighbor search, the distance calculation uses XOR-operations with linear cost.

the required memory space compared with the original vectors. When calculating the distance between two vectors, we only need to calculate the Hamming distance between two vectors. The XOR operation is much simpler than the original operation. In the application of the nearest neighbor algorithm, the distance between each database vector and the request vector can be quickly calculated and compared. Fig.2 illustrates the computation of the nearest neighbor search based on short code.

In order to systematically represent the compression process of short code and apply it to secure computation, we transform the k hash functions for compression into a matrix of $k * m$. The whole compression process can be seen as a matrix multiplication operation between this $k \times m$ matrix and the m-dimensional vector. Since the elements in the compressed vector are Boolean values, we also need to transform the output value of matrix multiplication. Here we choose to use step function ε to process the output values. The definition of a step function is as follows:

$$\varepsilon(x) = \begin{cases} 1 & x \geq 0 \\ 0 & x < 0 \end{cases}$$

Using the step functions mentioned above, we can convert the results of matrix multiplication into Boolean values. In secure computation, such step functions can be implemented by comparison protocols, which can compare the secret value with zero and output a Boolean value.

The method of matrix generation determines the accuracy of the searching result. How to generate the parameters of the matrix is an important research subject. One method is to generate parameters in a random way. This method has high efficiency in generating parameters, but random methods cannot guarantee the accuracy of the final search

results. Another approach to improve the accuracy is to train the original data set and determine the parameters of the matrix by the results of training. This approach is similar to pattern recognition in machine learning. The accuracy of the matrix generated by training in this approach depends on the quality of the training algorithm. If there is no training data, some special transformation algorithms such as spectral hashing [24] can be used to create matrix parameters.

2.3 Secret Sharing

Secret sharing refers to methods for distributing a secret amongst a group of parties, each of whom holds a part of shared values. In this paper, we use secret sharing under the concept of 2PC, so here we consider two parties P_0 and P_1. For an arbitrary value a, when it is considered in secret sharing protocol, we express it as $[a]$. When $[a]$ is shared, we divide it into two part expressed as $[a]_0$ and $[a]_1$, usually P_0 will hold $[a]_0$ and P_1 will hold $[a]_1$.The shared protocols used in this paper are mainly divided into Arithmetic sharing and Boolean sharing, in which the sharing values are integers and Boolean values. For any sharing value $[a]$, if it is a Boolean value that has only one bit, we express it as $[a]^B$. If it is an Arithmetic value with l-bits, we express it as $[a]^A$, and all the operations are under the ring $\mathbb{Z}_{2^l}$.

In Arithmetic sharing, the parties share the value in the form of Arithmetic numbers, the length of the Arithmetic numbers is defined by the parameter l, and all the operations occurred in Arithmetic sharing is executed in the ring $\mathbb{Z}_{2^l}$. As a simple case, we suppose that P_0 and P_1 share a l-bits secret value $[x]^A$. P_0 and P_1 both have a part of this value. We set $[x]_0^A + [x]_1^A = [x]^A$ with $[x]_0^A, [x]_1^A \in \mathbb{Z}_{2^l}$,for $i \in \{0,1\}$, P_i holds the value $[x]_i^A$. P_0 and P_1 are unable to obtain the shared value of each other. In this step, if we want to guarantee the privacy of $[x]^A$, this sharing process cannot be executed by either P_0 or P_1, because the party who execute the process can know the values of both parts. In this paper, we adopt the client-aided model, in which the sharing step will be performed by a client as the third party, and the value will be distributed to P_0 and P_1. The detail of client-aided model will be informed in Section 3.4, and the detail steps of Arithmetic sharing are as follows.

- `Shared Values`: For an Arithmetic sharing $[x]^A$, set $[x]_0^A + [x]_1^A = [x]^A$, for $i \in 0,1$, P_i holds the value $[x]_i^A$.
- `Sharing`: $[Shr]_i^A(x), i \in 0,1$: This operation is used for the case that P_i shares a part of the value $[x]_{1-i}^A \in \mathbb{Z}_{2^l}$ to P_{1-i}. P_i choose a value $r \in \mathbb{Z}_{2^l}$, set $[x]_i^A = x - r$, $[x]_{1-i}^A = r$,and send $[x]_{1-i}^A$ to P_{1-i}.
- `Reconstruction`: $[Rec]_i^A(x), i \in 0,1$: This operation is used for the case that P_0 and P_1 reconstruct the shared value $[x]_0^A$ and $[x]_1^A$ to original value $[x]^A$. For $i \in 0,1$, P_{1-i} sends $[x]_{1-i}^A$ to P_i and P_i computes $[x]^A = [x]_0^A + [x]_1^A$.

The basic operations for Arithmetic sharing are addition and multiplication, evaluated as follow.

- `Addition`: Suppose that P_0 and P_1 have already shared two Arithmetic value $[x]^A$ and $[y]^A$. For $i \in 0,1$, P_i holds the values $[x]_i^A$ and $[y]_i^A$. When computing $[z]^A = [x]^A + [y]^A$, for $i \in 0,1$, P_i locally compute $[z]_i^A = [x]_i^A + [y]_i^A$.
- `Multiplication`: Suppose that P_0 and P_1 have already shared two Arithmetic value $[x]^A$ and $[y]^A$. For $i \in 0,1$, P_i holds the values $[x]_i^A$ and $[y]_i^A$. When computing $[z]^A = [x]^A * [y]^A$, an extra Beaver multiplication triple is necessary. The multiplication triple is constructed as $[c]^A = [a]^A * [b]^A$ and is shared to P_0 and P_1. For $i \in 0,1$, P_i holds the values $[a]_i^A$,$[b]_i^A$ and $[c]_i^A$. P_i computes $[e]_i^A = [x]_i^A - [a]_i^A$, $[f]_i^A = [y]_i^A - [b]_i^A$, and use $[Rec]_i^A(e)$ and $[Rec]_i^A(f)$ to get the value of $[e]^A$ and $[f]^A$. After doing all the steps above, P_i can compute $[z]_i^A = i * [e]^A * [f]^A + [f]^A * [a]_i^A + [e]^A * [b]_i^A + [c]_i^A$.

In Boolean sharing, the value is share in the form of single bit $[x]^B \in 0,1$. The GMW protocol[2] is widely used for Boolean sharing. Boolean sharing with GMW protocol has the following operations.

- `Shared Values`: For an Boolean sharing $[x]^B$, set $[x]_0^B \bigoplus [x]_1^B = [x]^B$, for $i \in 0,1$, P_i holds the value $[x]_i^B$.
- `Sharing`: $[Shr]_i^B(x), i \in 0,1$: This operation is used for the case that P_i shares a part of the value $[x]_{1-i}^B \in 0,1$ to P_{1-i}. P_i choose a value $r \in 0,1$, set $[x]_i^B = x \bigoplus r$, $[x]_{1-i}^B = r$,and send $[x]_{1-i}^B$ to P_{1-i}.
- `Reconstruction`: $[Rec]_i^B(x), i \in 0,1$: This operation is used for the case that P_0 and P_1 reconstruct the shared value $[x]_0^B$ and $[x]_1^B$ to original value $[x]^B$. For $i \in 0,1$, P_{1-i} sends $[x]_{1-i}^B$ to P_i and P_i computes $[x]^B = [x]_0^B \bigoplus [x]_1^B$.

The basic operations in Boolean sharing are XOR-gate and AND-gate, these two kinds of gate can be extended to any other gates in bit operation. XOR-gate and AND-gate are evaluated as follow:

- `XOR`: Suppose that P_0 and P_1 have already shared two Boolean value $[x]^B$ and $[y]^B$. For $i \in 0,1$, P_i holds the values $[x]_i^B$ and $[y]_i^B$. When computing $[z]^B = [x]^B \bigoplus [y]^B$, for $i \in 0,1$, P_i locally compute $[z]_i^B = [x]_i^B \bigoplus [y]_i^B$.
- `AND`: Suppose that P_0 and P_1 have already shared two Boolean value $[x]^B$ and $[y]^B$. For $i \in 0,1$, P_i holds the values $[x]_i^B$ and $[y]_i^B$. When computing $[z]^B = [x]^B \wedge [y]^B$, similar to the multiplication triple required in the multiplication operation of Arithmetic sharing, here we need to generate a Boolean precomputing triple, which can be used to calculate AND-gate while maintaining input privacy. As another choice, we can use oblivious transfer protocol to achieve the operation which is introduced in [25].

In addition to the above basic operations used in Arithmetic sharing and Boolean sharing, here we introduce two complex protocols: comparison protocol and B2A protocol. These two protocols are also constructed by basic protocols and operations, but they are important protocols needed for privacy preserving ANN in this paper. They are constructed

as follow :

- `Comparison protocol`: This protocol is used to compared two values in Arithmetic sharing and return a flag to indicate result. It takes as input 2 shared l-bits Arithmetic value $[x]^A$ and $[y]^A$, return a Boolean value $[cp]^B$. The value satisfies that:

$$[cp]^B = \begin{cases} true & [x]^A \leq [y]^A \\ false & [x]^A > [y]^A \end{cases}$$

  The detail algorithm and an implement of this protocol is proposed in [11]. When the comparison protocol is executed, two users start related operations at the same time. For $i \in 0, 1$, P_i takes $[x]_i^A$ and $[y]_i^A$ as input. After the computation in this protocol, P_0 and P_1 obtain $[cp]_0^B$ and $[cp]_1^B$ respectively. These to values satisfies that:

$$[cp]_0^B \bigoplus [cp]_1^B = [cp]^B$$

  Due to the conclusion in [11] and our research, P_0 and P_1 needs 2 rounds of communications when the parameter l is equal to 8(since the vector compressed by short code used in this paper has 128 elements, the Hamming distance is not more than 2^8). In addition, P_0 and P_1 require 1012 pairs of multiplication triples and 3 pairs of Boolean triples. These pre-computing data can be generated by a third party in advance and transferred together in one round.

- `Boolean-to-Arithmetic(B2A) protocol`: This protocol is used to convert a Boolean sharing into Arithmetic sharing. It takes as input a shared Boolean value $[x]^B$, Return an l-bits Arithmetic value $[x]^A$. The value satisfies that:

$$[x]^A = \begin{cases} 1 & [x]^B = true \\ 0 & [x]^B = false \end{cases}$$

  This protocol also requires P_0 and P_1 to execute simultaneously, and we can also find the implement and detail algorithm in [11]. The algorithm has 4 steps that is listed as follow:

  (1) For $i \in 0, 1$, P_i locally converts his values $[x]^B$ to $[\dot{x}]^B$ by the following rule:

$$[\dot{x}]_i^A = \begin{cases} 1 & [x]_i^B = true \\ 0 & [x]_i^B = false \end{cases}$$

  (2) For $i \in 0, 1$, P_i runs $Shr_i^A([\dot{x}]_i^A)$ and share the value as $[[\dot{x}]_i^A]_i^A$ and $[[\dot{x}]_i^A]_{1-i}^A$.

  (3) P_0 and P_1 compute $[[\dot{x}]_0^A]^A * [[\dot{x}]_1^A]^A$ using a multiplication protocol. We set $[m]^A = [[\dot{x}]_0^A]^A * [[\dot{x}]_1^A]^A$, for $i \in 0, 1$, P_i holds the value $[m]_i^A$.

  (4) For $i \in 0, 1$, P_i locally computes $[x]_i^A = [\dot{x}]_i^A - [m]_i^A$.

  In this protocol, P_0 and P_1 require one round of communication to exchange intermediate value, and a multiplication tripe is required.

2.4 Secure Computation

The secret computation tool used in this paper is the 2PC based on secret sharing. As we denote in Section 1, this kind of protocol requires a lot of pre-computing data. Here we use client-aided model[15]. Specifically, we consider three parties in the whole protocol. Two of them act as servers that compute the protocols, and the other one acts as a client to provide part of secret input. In addition, this client provides pre-computed data for the two parties.

2.5 Semi-honest Model

If a protocol is secure in a semi-honest model, it means that each participant in the protocol will follow the rules of the protocol. They will not try to modify their inputs and outputs, attempt to break the protocol or cooperate with other participants to obtain more information. They only record the data and intermediate information they can get and try to infer more useful information.

Semi-honest model is the general and basic security model. The construction proposed in this paper is secure in the semi-honest model.

3. Privacy Preserving ANN

3.1 General Introduction

The secret sharing protocol has good compatibility with the short code. The short code method uses XOR operation to compute the Hamming distance and compare the distance. In secret sharing protocol, the XOR operation is computed locality. This greatly reduces the time required for privacy-preserving calculation for Hamming distances. In addition, the secure computation protocol based on secret sharing has more advantages than other secure computing protocols when using short code method to compress the query vector. Since the compressing process is a kind of matrix multiplication, in which each element is multiplied separately and then added together, all multiplications can be calculated in parallel. Therefore, P_0 and P_1 can transfer intermediate values altogether through one communication in multiplication protocol. Similarly, the multiplication triangle required for multiplication can also be transmitted in one communication. Adopting the above method can reduce the round of communication in secret sharing-based matrix multiplication to one round. This greatly increases the transmission efficiency.

We divide the process in privacy preserving ANN search into preparation part, distance computing part and comparison part. In the protocol mentioned below, the secret owned by parties will be taken as input, but it doesn't mean that parties are required to transmit their own secret. In the process of protocol, two parties calculate separately and exchange some intermediate data. The data transmitted will not reveal their secrets.

We consider the following situation. A server has a vector database, and a user also holds a vector. User wants to search for the vector in the database that is closest to his vector. During searching, user wants to ensure that his

own data is private, and server does not want to reveal the database either. The secret sharing protocol and short code method introduced in Chapter 2 can be used in the privacy preserving ANN to protect privacy of parties.

We use the client-aided model in our construction. Suppose that there are three parties P_0, P_1 and P_2. We take P_2 as client, P_0 and P_1 as distribute servers. P_0 and P_1 can communicate and transfer data to each other, but they should ensure that after the client starts querying, the two servers will not reveal the private data from client.

3.2 Preparation Part

As preparation, we suppose that the database $[vdb]^B = \{[V^1]^B, [V^2]^B, ..., [V^n]^B\}$ already compressed by the short code that each vector has 128 binary elements. P_0 and P_1 have shared the vector database as follows:

$$\begin{cases} [vdb]_0^B = \{[V^1]_0^B, [V^2]_0^B, ..., [V^n]_0^B\} \\ [vdb]_1^B = \{[V^1]_1^B, [V^2]_1^B, ..., [V^n]_1^B\} \end{cases}$$

$[vdb]_0^B$ and $[vdb]_1^B$ can be reconstructed as $[vdb]^B$. For $i \in 0, 1$, P_i holds $[vdb]_i^B$. At the same time, P_2 compresses his query vectors into a 128-dimensions vector called $[qv]^B$ that has the same length as the vector in database. This process requires matrix multiplication, which is temporarily omitted in our program and replaced by randomly generated database and vectors. After the above preparation, P_2 divides his query vector into two parts $[qv]_0^B$ and $[qv]_1^B$ by sharing the protocol and sends them to P_0 and P_1 respectively. In addition, P_2 also send some pre-computing data to P_0 and P_1, which we will mention in Section 3.5.

3.3 Distance Computing Part

This part of the protocol calculates the Hamming distance between the query vector and each vector in the database and return a set of 8-bit arithmetic sharing with n elements(n is the vector amount of database). As an example, to calculate the distance between $[qv]^B$ and $[V^a]^B$(parameter a satisfies that $1 \leq a \leq n$), P_0 and P_1 need to do the following operations.

(1) For $i \in 0, 1$, P_i locally computes $[hv^a]_i^B = [qv]_i^B \bigoplus [V^a]_i^B$.

(2) P_0 and P_1 use a B2A function to convert the Boolean values $[hv^a]_0^B$ and $[hv^a]_1^B$ into Arithmetic sharing value $[hv^a]_0^A$ and $[hv^a]_1^A$.

(3) For $i \in 0, 1$, P_i locally adds up the values in each dimension of $[hv^a]_i^A$ and store the new value as $[hd^a]_i^A$. Specifically, suppose $[hv^a]_i^A = \{hv^1, hv^2, ..., hv^{128}\}$, then P_i calculates $[hd^a]_i^A = hv^1 + hv^2 + ... + hv^{128}$ in the ring of $\mathbb{Z}_{2^l}$.

In the whole process, we need to calculate every element in the database and get Hamming distance of all database vectors. We call the Hamming distance data set as $[HD]^A$. For $i \in 0, 1$, P_i holds a sharing of the dataset as follows:

$$[HD]_i^A = \{[hd^1]_i^A, [hd^2]_i^A, ..., [hd^n]_i^A\}$$

The time complexity of the program is linear with the scale of the database. For each database vector, we need to calculate the distance with the query vector. In distance calculation, P_0 and P_1 need one round of communication to complete the B2A protocol. Therefore, in the database with n vectors, we usually need n rounds of communication to complete the distance calculation.

Here we can use an optimization scheme since the calculation of each vector in the database can be carried out parallelly, we can also merge the communication in the B2A function together. Through this operation, the data exchanges that originally need n rounds can be reduced to one round.

3.4 Comparison Part

In this part, we group the elements of $[HD]^A$ in pairs, compare each pair and leave the smaller elements in each pair. Then we group the remaining elements in pairs again and leave the smaller elements. For a dataset $[HD]^A$ which is generated from a database with n vector, after $\lceil log_2 n \rceil$ rounds of operations, there is only one element left, and this element is the smallest element in the original dataset. The comparison part we constructed takes as input two Hamming distances in Arithmetic sharing and return the sharing smaller distance. We use the comparison protocol, B2A protocol and multiplication protocol in the algorithm. Suppose we need to compare the two parameters $[a]^A$ and $[b]^A$ and leave the smaller value. For $i \in 0, 1$, P_i holds $[a]_i^A$ and $[b]_i^A$, and they will do the following operations:

(1) P_0 and P_1 run comparison protocol that take as input $([a]_0^A, [a]_1^A, [b]_0^A, [b]_1^A)$. P_0 gets $[cp]_0^B$ and P_1 gets $[cp]_1^B$ as the comparison result.

(2) P_0 and P_1 run B2A protocol that take as input $[cp]_0^B$ and $[cp]_1^B$, P_0 gets $[e]_0^A$ and P_1 gets $[e]_1^A$.

(3) P_0 calculates $[\bar{cp}]_0^B = ![cp]_0^B$ and P_0 calculate $[\bar{cp}]_1^B = [cp]_1^B$.

(4) P_0 and P_1 run B2A protocol that take as input $[\bar{cp}]_0^B$ and $[\bar{cp}]_1^B$, P_0 gets $[\bar{e}]_0^A$ and P_1 gets $[\bar{e}]_1^A$.

(5) P_0 and P_1 run multiplication protocol that take as input $([a]_0^A, [a]_1^A, [e]_0^A, [e]_1^A)$, compute $[c_1]^A = [a]^A * [e]^A$. P_0 gets $[c_1]_0^A$ and P_1 gets $[c_1]_1^A$.

(6) P_0 and P_1 run multiplication protocol that take as input $([b]_0^A, [b]_1^A, [\bar{e}]_0^A, [\bar{e}]_1^A)$, compute $[c_2]^A = [b]^A * [\bar{e}]^A$. P_0 gets $[c_2]_0^A$ and P_1 gets $[c_2]_1^A$.

(7) For $i \in 0, 1$, P_i locally compute $[c]_i^A = [c_1]_i^A + [c_2]_i^A$.

After these operations. P_i stores the value $[c]_i^A$ as the sharing of smaller value from $[a]^A$ and $[b]^A$.

One comparison protocol, two B2A protocols, and two multiplication protocols are used in the above operations. According to the previous conclusion, the total number of communication rounds required by these protocols is $2 + 2 \times 1 + 2 \times 1 = 6$ rounds. In fact, this part can also be optimized. The two B2A protocols and the two multiplication protocols can be implemented in parallel. The intermediate parameters of the two protocols can be transmitted by one round of communication. In the optimized scheme, the rounds of communication needed for one comparison has reduced to 3.

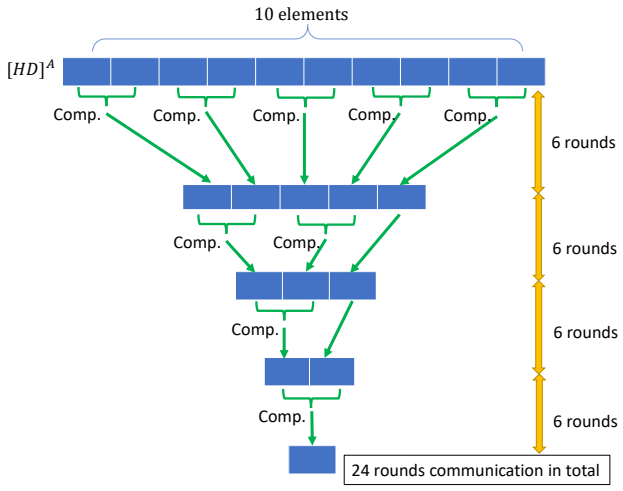

Fig. 3 Process of comparison part with $n = 10$, it requires 24 rounds of communications which is equal to $6 \times \lceil \log_2 n \rceil$.

In the whole comparison part, there are many places that can be optimized by parallel processing. When grouping the elements in pairs, all pairs can be processed in parallel. Therefore, in the whole comparison part, 6 rounds of communication are used for each round of pair. Since we need $\lceil \log_2 n \rceil$ rounds of pairing, the total rounds of communication in this part can be reduced to $6 \times \lceil \log_2 n \rceil$ after optimization.

Fig.3 shows the actual execution of the comparison protocol in the case that $n = 10$. By this way, the number of communication returns in the whole comparison protocol is reduced as much as possible, and the efficiency of the secure ANN based on secret sharing is improved.

3.5 Pre-computing Part

There are a lot of pre-computing data that need to be calculated in advance in the whole protocol. If these data are generated during the calculation process, it will slow down the execution time of the whole protocol. Since if the parameter n is determined in the protocol, the required operation process is also determined, so all the pre-computing data can also be created and sent to users before the implementation of the protocol, and it can be used directly when it is required. The amount of pre-computing data is linear with the number of vectors in the database.

The in multiplication triple B2A protocol, comparison protocol, and multiplication protocol can all be created by P_2 in advance and transmitted to P_0 and P_1 before executing the calculation. These pre-computed data are not affected by the input data, so they can be stored in the memory space of P_0 and P_1 and used sequentially when needed. According to the data size of the database, we can calculate the amount of multiplication triple required in privacy preserving ANN. Previous conclusions show that in a database of n vectors, $3n$ B2A protocols, $2n$ multiplication protocols and n comparison protocols are required.

4. Experimental Results

We implement our construction based on Python with NumPy library. We write three separate programs represented to P_0, P_1 and P_2 respectively. The program contains the *python.socket* library, which enables programs to communicate with each other.

We first make the experiment and run the program on a personal laptop with Intel Core i7-6700HQ 2.60GHz and 16GB RAM. We run programs of P_0, P_1 and P_2 on this laptop at the same time, so that they can calculate and communicate with each other. We set the vector amount in the database to 1000, 2000, 3000 and 10000, and tested the running time of the program. The test results are shown in Table 1.

After the above experiments, we used three computers and built a LAN network, and carried out the second round of experiments under the LAN environment. We let the laptop used in the first experiment as the client, which is also called P_2 in our construction, and two other computers as P_0 and P_1. The experimental results are recorded in Table 2.

In the experimental results, pre-computation time indicates the time that P_2 provides pre-computing data for the whole computation process, including the generation of basic parameters(such as l and n in our construction), query vectors and multiplication triples. The time of this part includes both the time of calculation and the time of data transmission. In the computation time, we show the time of distance computation and the time of comparison respectively. Distance computation is the process that P_0 and P_1 compute Hamming distance from the query vector and database vectors and returns the vector of Hamming distance. Comparison time includes the time of finding the minimum distance from the vector of Hamming distance. Both parts of the time include computing time and communication time.

The experimental results show that the running time of privacy-preserving ANN construction that we proposed in this paper is linear with the amount of database vectors. At the same time, depending on current construction, when the data amount is less than 10000, it is possible to put it into practical use. At the same time, we find that the bottleneck of the current construct is the comparison part that to compare the Hamming distance and return to the minimum one. In the local environment, the running time of the current comparison protocol accounts for 94.4% of the total computing time. In LAN, this time accounts for 94.9%. It can be inferred that in higher latency environment, the time spent by comparing protocols determines the running time of the whole program. Similar conclusions can be drawn from the inference of the algorithm itself because the number of multiplication triples generated in one comparison protocol of an 8-bit Arithmetic sharing is over a thousand, which is tens of times larger than that of other protocols. However, in the current research, there is no better solution to the privacy-preserving comparison protocol. With the further development of comparison protocol, the efficiency of short code-based ANN algorithm can be further

Table 1 Run Time of Privacy Preserving ANN Pre-computation/Computation/Total Process with Different Amount of Database Vector in Local Environment(All Program Run in One Computer)

Data amount	Pre-computation Time (s)	Computation Time			Total Time (s)
		Distance Computation Time (s)	Comparison Time (s)	Total Computation Time (s)	
1000	7.2	0.3	5.3	5.6	12.8
2000	14.3	0.6	10.7	11.3	25.6
3000	23.2	0.9	16.8	17.7	40.9
10000	74.7	2.9	49.3	52.2	126.9

Table 2 Run Time of Privacy Preserving ANN Pre-computation/Computation/Total Process with Different Amount of Database Vector in LAN (3 Computers)

Data amount	Pre-computation Time (s)	Computation Time			Total Time (s)
		Distance Computation Time (s)	Comparison Time (s)	Total Computation Time (s)	
1000	9.2	0.3	6.2	6.5	15.7
2000	16.8	0.7	12.3	13.0	29.8
3000	27.6	1.0	17.7	18.7	46.3
10000	78.2	3.3	61.2	64.5	142.7

improved.

5. Conclusion

We propose a construction of privacy-preserving ANN and test its expression through experiments. However, there are still other parts that need to be added to the current construction. Firstly, the process of compressing the original vector by short code is not included in the program we made, and there is no conclusion about the best method of generating the parameters of the compression matrix. Secondly, current research just constructs the privacy-preserving ANN algorithm that outputs the smallest distance. But in practice, it is not enough for users just to know the minimum distance of compressed vectors, and they also need to know the corresponding original vectors and obtain the original vectors. Here we propose two approaches:

(1) While sharing database vectors, the server binds the original vectors together. In the whole computing process, only the Hamming distance is compared in the comparison protocol, and the original vectors are bind together to calculate in other protocols. In this way, the minimum Hamming distance can be output bind with the corresponding original vector.

(2) Instead of binding the original vectors while sharing, here the data bound to the database vector is the index of the original vector. After the same process in approach 1, the index of the original vector can be output. Then a private information retrieval (PIR) protocol [26] is used between the user and the server, and the original vector is obtained by the user.

The first approach is easy to implement, but it will affect the efficiency of the whole construction because the bound original vector data also takes up a certain amount of memory space and participates in the calculation. The second method has little impact on efficiency but requires additional PIR schemes. Choosing which approach is one of the future topics.

The current experimental results show that privacy-preserving ANN has acceptable running time when the vector amount is less than 10000 in the database. For the current popular ANN search of million-scale or even billion-scale, it is necessary to classify the whole database and reduce the number of vectors for searching to less than 10000, so that they can be used in secure computation. The current experiment results also provide the basis and objectives for future research.

Acknowledgments This work was partly supported by JSPS KAKENHI Grant Number 17KT0081 and JST CREST JPMJCR19F6.

References

[1] A. C. Yao: "How to generate and exchange secrets (extended abstract)", 27th Annual Symposium on Foundations of Computer Science, Toronto, Canada, 27-29 October 1986, pp. 162–167 (1986).

[2] O. Goldreich, S. Micali and A. Wigderson: "How to play any mental game or A completeness theorem for protocols with honest majority", Proceedings of the 19th Annual ACM Symposium on Theory of Computing, 1987, New York, New York, USA, pp. 218–229 (1987).

[3] R. L. Rivest, L. Adleman and M. L. Dertouzos: "On data banks and privacy homomorphisms", Foundations of Secure Computation, Academia Press, pp. 169–179 (1978).

[4] C. Gentry: "Fully homomorphic encryption using ideal lattices", Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, May 31 - June 2, 2009, pp. 169–178 (2009).

[5] T. Araki, J. Furukawa, Y. Lindell, A. Nof and K. Ohara: "High-throughput semi-honest secure three-party computation with an honest majority", Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016, pp. 805–817 (2016).

[6] K. Chida, D. Genkin, K. Hamada, D. Ikarashi, R. Kikuchi, Y. Lindell and A. Nof: "Fast large-scale honest-majority MPC for malicious adversaries", Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part III, pp. 34–64 (2018).

[7] I. Damgård, M. Fitzi, E. Kiltz, J. B. Nielsen and T. Toft: "Unconditionally secure constant-rounds multi-party computation for equality, comparison, bits and exponentiation", Theory of Cryptography, Third Theory of Cryptography Conference, TCC 2006, New York, NY, USA, March 4-7, 2006, Proceedings, pp. 285–304 (2006).

[8] T. Nishide and K. Ohta: "Multiparty computation for interval, equality, and comparison without bit-decomposition protocol", Public Key Cryptography - PKC 2007, 10th Inter-

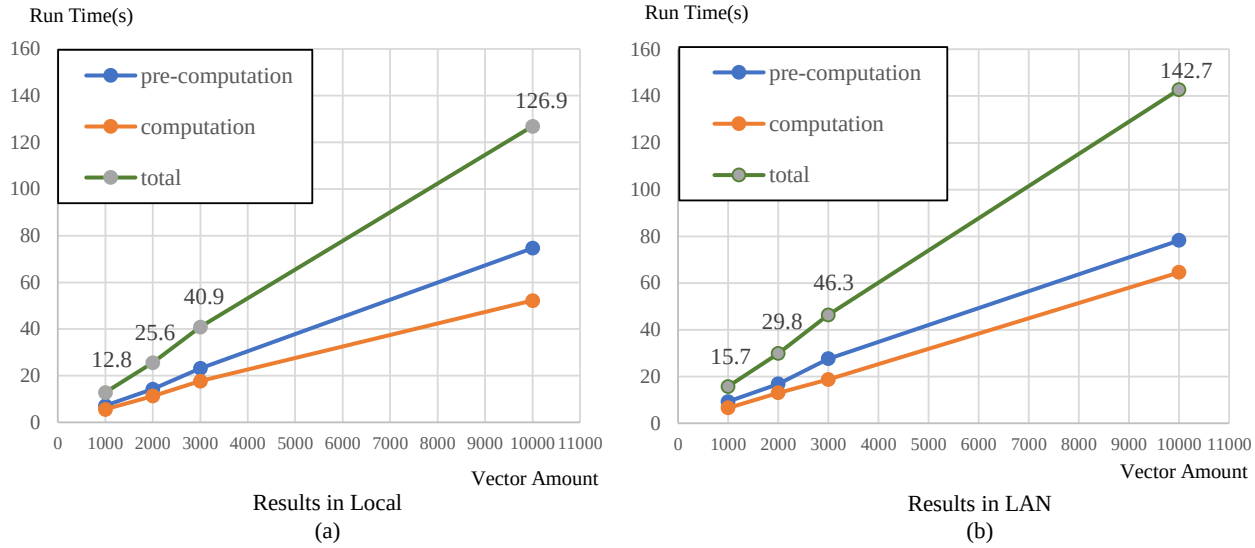

Fig. 4 Run time of privacy-preserving ANN with different amount of database vector.

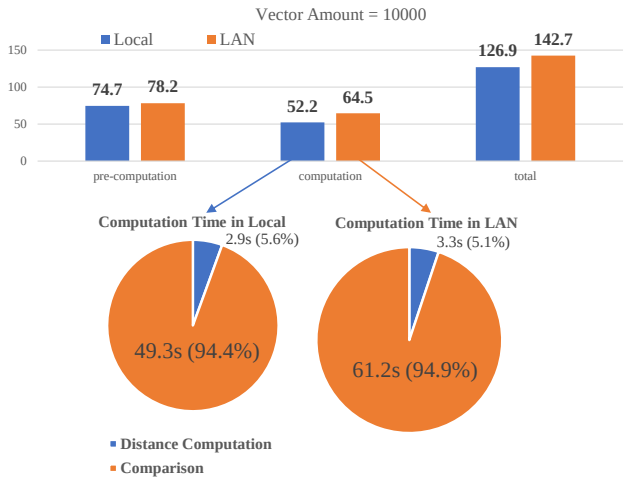

Fig. 5 Comparison of run time in Local and LAN environments and the proportion of time spent on each part of the calculations.

national Conference on Practice and Theory in Public-Key Cryptography, Beijing, China, April 16-20, 2007, Proceedings, pp. 343–360 (2007).

[9] D. Bogdanov, M. Niitsoo, T. Toft and J. Willemson: "High-performance secure multi-party computation for data mining applications", Int. J. Inf. Sec., **11**, 6, pp. 403–418 (2012).

[10] H. Morita, N. Attrapadung, T. Teruya, S. Ohata, K. Nuida and G. Hanaoka: "Constant-round client-aided secure comparison protocol", Computer Security - 23rd European Symposium on Research in Computer Security, ESORICS 2018, Barcelona, Spain, September 3-7, 2018, Proceedings, Part II, pp. 395–415 (2018).

[11] S. Ohata and K. Nuida: "Towards high-throughput secure MPC over the internet: Communication-efficient two-party protocols and its application", CoRR, **abs/1907.03415**, (2019).

[12] D. Demmler, T. Schneider and M. Zohner: "ABY - A framework for efficient mixed-protocol secure two-party computation", 22nd Annual Network and Distributed System Security Symposium, NDSS 2015, San Diego, California, USA, February 8-11, 2015 (2015).

[13] 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", Proceedings of the 2018 on Asia Conference on Computer and Communications Security, AsiaCCS 2018, Incheon, Republic of Korea, June 04-08, 2018, pp. 707–721 (2018).

[14] P. Mohassel and P. Rindal: "Aby^3: A mixed protocol framework for machine learning", Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018, pp. 35–52 (2018).

[15] P. Mohassel and Y. Zhang: "Secureml: A system for scalable privacy-preserving machine learning", 2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017, pp. 19–38 (2017).

[16] J. Liu, M. Juuti, Y. Lu and N. Asokan: "Oblivious neural network predictions via minionn transformations", Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017, pp. 619–631 (2017).

[17] P. Mohassel, O. Orobets and B. Riva: "Efficient server-aided 2pc for mobile phones", PoPETs, **2016**, 2, pp. 82–99 (2016).

[18] Z. Erkin, M. Franz, J. Guajardo, S. Katzenbeisser, I. Lagendijk and T. Toft: "Privacy-preserving face recognition", Privacy Enhancing Technologies, 9th International Symposium, PETS 2009, Seattle, WA, USA, August 5-7, 2009. Proceedings, pp. 235–253 (2009).

[19] A. Sadeghi, T. Schneider and I. Wehrenberg: "Efficient privacy-preserving face recognition", Information, Security and Cryptology - ICISC 2009, 12th International Conference, Seoul, Korea, December 2-4, 2009, Revised Selected Papers, pp. 229–244 (2009).

[20] Y. Huang, L. Malka, D. Evans and J. Katz: "Efficient privacy-preserving biometric identification", Proceedings of the Network and Distributed System Security Symposium, NDSS 2011, San Diego, California, USA, 6th February - 9th February 2011 (2011).

[21] M. Barni, T. Bianchi, D. Catalano, M. D. Raimondo, R. D. Labati, P. Failla, D. Fiore, R. Lazzeretti, V. Piuri, F. Scotti and A. Piva: "Privacy-preserving fingercode authentication", Multimedia and Security Workshop, MM&Sec 2010, Roma, Italy, September 9-10, 2010, pp. 231–240 (2010).

[22] G. Asharov, S. Halevi, Y. Lindell and T. Rabin: "Privacy-preserving search of similar patients in genomic data", PoPETs, **2018**, 4, pp. 104–124 (2018).

[23] H. Chen, I. Chillotti, Y. Dong, O. Poburinnaya, I. P. Razenshteyn and M. S. Riazi: "SANNS: scaling up secure approximate k-nearest neighbors search", CoRR, **abs/1904.02033**, (2019).

[24] Y. Weiss, A. Torralba and R. Fergus: "Spectral hashing", Advances in Neural Information Processing Systems 21, Proceedings of the Twenty-Second Annual Conference on Neural Information Processing Systems, Vancouver, British Columbia, Canada, December 8-11, 2008, pp. 1753–1760 (2008).

[25] M. O. Rabin: "How to exchange secrets with oblivious transfer", IACR Cryptology ePrint Archive, **2005**, p. 187 (2005).

[26] B. Chor, O. Goldreich, E. Kushilevitz and M. Sudan: "Private information retrieval", 36th Annual Symposium on Foundations of Computer Science, Milwaukee, Wisconsin, USA, 23-25 October 1995, pp. 41–50 (1995).