PANTHER: Private Approximate Nearest Neighbor Search in the Single Server Setting

Jingyu Li¹, Zhicong Huang¹, Min Zhang², Jian Liu³, Cheng Hong¹, Tao Wei¹, and Wenguang Chen¹

¹Ant Group ² Institute of Software, Chinese Academy of Sciences ³Zhejiang University

Abstract

Approximate nearest neighbor search (ANNS), also known as vector search, is an important building block for varies applications, such as databases, biometrics, and machine learning. In this work, we are interested in the private ANNS problem, where the client wants to learn (and can only learn) the ANNS results without revealing the query to the server. Previous private ANNS works either suffers from high communication cost (Chen et al., USENIX Security 2020) or works under a weaker security assumption of two non-colluding servers (Servan-Schreiber et al., SP 2022). We present Panther, an efficient private ANNS framework under the single server setting. Panther achieves its high performance via several novel co-designs of private information retrieval (PIR), secretsharing, garbled circuits, and homomorphic encryption. We made extensive experiments using Panther on four public datasets, results show that Panther could answer an ANNS query on 10 million points in 23 seconds with 318 MB of communication. This is more than $6\times$ faster and $18\times$ more compact than Chen et al..

1 Introduction

The *k*-nearest neighbors search (*k*-NNS) is a fundamental problem in the fields of machine learning, data mining, and pattern recognition. Given a database of multidimensional points, *k*-NNS aim to find the top-*k* closest points to a query point. *k*-NNS has numerous applications, such as recommendation systems [\[6,](#page-13-0) [43\]](#page-15-0), image recognition [\[23\]](#page-14-0), anomaly detection [\[8\]](#page-13-1), and Retrieval-Augmented Generation (RAG) [\[18,](#page-13-2) [24\]](#page-14-1) in large language models. In the case of large high-dimensional databases where exact *k*-NNS could be prohibitively expensive, the problem is often relaxed to approximated nearest neighbors search (ANNS), which allows the top-*k* closest neighbors to be returned with a high probability rather than exactly. Compared to exact *k*-NNS, there exist several more efficient algorithms [\[17,](#page-13-3) [22,](#page-14-2) [26,](#page-14-3) [28](#page-14-4)[–30,](#page-14-5) [34,](#page-14-6) [42,](#page-15-1) [44,](#page-15-2) [46,](#page-15-3) [50\]](#page-15-4) for ANNS.

Normally, ANNS algorithms require knowing the database and the query point in clear to proceed. However there exist scenarios where both the database server and the query client are reluctant to reveal their sensitive data to each other. Examples include genome similarity search [\[3\]](#page-13-4) and fingerprint recognition [\[19\]](#page-13-5). To this end, it would be promising if one can propose private ANNS services based on secure multi-party computation (MPC).

A considerable amount of researches have been conducted along this line. Some of them [\[3,](#page-13-4)[39,](#page-15-5)[41,](#page-15-6)[52\]](#page-15-7) use a full database distance computation, thus are only practical for small or low-dimensional databases. Some works [\[38\]](#page-14-7) try to increase efficiency at the cost of extra information leakage. Another popular choice is to build the system under a multi-server setting [\[10,](#page-13-6) [36,](#page-14-8) [40,](#page-15-8) [47\]](#page-15-9), where the client's privacy only holds if the servers do not collude. They adopt such setting because the complexity of MPC could be greatly reduced given extra non-colluding assumption. However, for clients looking for private ANNS services, systems under such security assumption could be less convincing than the single server ones.

In USENIX Security 2020, Chen et al. [\[9\]](#page-13-7) propose the milestone framework SANNS that scales to a database of 10 million entries in the singe server setting. However, SANNS relies on several secure computation techniques with high communication overhead, such as distributed oblivious RAM (DORAM) [\[12\]](#page-13-8) and garbled circuits [\[48\]](#page-15-10). As a result, SANNS requires several gigabytes of communication per query, which is not deployable over realistic network connections (Section 8.3, [\[36\]](#page-14-8)).

After carefully examining the problem, we find two reasons that make it difficult to design an efficient single server private ANNS: 1) One of the best practice in ANNS is to divide the database into several clusters, so that the top-*k* distance computations are only required for some close clusters, avoiding the full database scan. Retrieving points in those close clusters is trivial in plaintext, but it needs to be done obliviously in MPC, requiring the expensive DORAM. 2) Computing top-*k* is one of the most challenging algorithms in MPC. Generally there are two kinds of MPC primitives, secret-sharing [\[15\]](#page-13-9)

and garbled circuits. Using the former for top-*k* computation will incur too many communication rounds, and the latter (which was used in SANNS) will incur high communication. These two difficulties are hard to circumvent, an evidence is that SANNS is still the state-of-the-art, more than four years after its publication.

1.1 Our Contributions

This work focuses on an efficient pipeline for addressing the private ANNS problem in the single server setting. By a deep co-design of private information retrieval (PIR), secretsharing, garbled circuits, and homomorphic encryption (HE), we successfully designed a system Panther that is scalable to datasets with tens of millions of samples, and is much more efficient than previous works. Our contributions in this work are four-fold:

- We design a novel "shuffle-and-reveal" protocol that enables us to solve the points retrieval problem with efficient batch PIR instead of distributed ORAM, reducing the communication cost of this step by 93% \sim 97%. We also propose efficient methods for post-processing the PIR results (they are homomorphic ciphertexts in special format) to make them compatible for subsequent MPC operations, which are nontrivial and could be of broader interest.
- We propose a mixed-primitive top-*k* approach paired with a carefully designed selection network to optimize the top-*k* selection process, reducing the communication cost by $86\% \sim 90\%$.
- We carefully design the encoding of homomorphic ciphertexts used in PIR and distance computation, enabling us to use a smaller HE parameter to mitigate the privacy issues related to HE noises, increasing the speed of HE calculations by $2 \sim 10 \times$.
- We introduce Panther, a fully implemented end-to-end framework for secure ANNS. We made extensive evaluations on the datasets SIFT-1M, Deep1B-1M, Deep1B-10M and Amazon, which are popularly used as benchmarks in the ANNS literature. Results show that Panther is $6 \times$ faster than the state-of-the-art method [\[9\]](#page-13-7) with the same accuracy, and the communication cost is reduced by more than one order of magnitude.

2 Preliminaries

In this section, we briefly introduce some building blocks that are used in Panther. The summary of the frequent notation used in this paper is summarized in Table [1.](#page-1-0)

Table 1: Summary of frequent notations.

2.1 Cryptographic Primitives

2.1.1 Additive secret sharing

Throughout this manuscript, we use 2-out-of-2 additive secret sharing schemes over the ring $\mathbb{Z}_{2^{\ell}}$. An ℓ -bit ($\ell \geq 2$) value *x* is additively shared as $\llbracket x \rrbracket_c$ and $\llbracket x \rrbracket_s$, where $\llbracket x \rrbracket_c$ is a random
share of x hald by client and $\llbracket x \rrbracket$ is a random share of x hald by share of x held by client and $\llbracket x \rrbracket_c$ is a random share of x held by server. To reconstruct the value *x*, we compute the modulo addition, i.e., $x \equiv [x]_c + [x]_s \mod 2^{\ell}$. For a real value $\tilde{x} \in \mathbb{R}$, we first ancode it as a fixed point value $x = |\tilde{x}2^{\ell}| \subset [-2^{\ell-1}, 2^{\ell-1}]$ first encode it as a fixed-point value $x = \lfloor \tilde{x}2^f \rfloor \in [-2^{\ell-1}, 2^{\ell-1})$ under a specified precision $f > 0$ before secretly sharing it. For a boolean value $z \in \{0, 1\}$, we use $[\![z]\!]_B^B$ and $[\![z]\!]_S^B$ to denote the shares of z such that $z = \mathbb{I} \in \mathbb{I}^B \cap \mathbb{I} \in \mathbb{I}^B$. Also we omit the the shares of *z* such that $z = [\![z]\!]_B^B \oplus [\![z]\!]_S^B$. Also we omit the subscript and only write $[\![x]\!]_B$ or $[\![x]\!]_B^B$ when the ownership is subscript and only write $\llbracket x \rrbracket$ or $\llbracket z \rrbracket^B$ when the ownership is irrelevent from the context irrelevant from the context.

2.1.2 Garbled circuit

Garbled circuit is a technique proposed by [\[49\]](#page-15-11) to achieve general secure two-party computation for boolean circuits. Essentially, one party, called *garbler*, generates truth tables for gates of the circuit in an encrypted format that can be evaluate by the other party, called *evaluator*. The major advantage of garbled circuit is *constant round* for network interaction, which is beneficial for deep circuits, such as *min/max* and *sorting*.

2.1.3 Lattice-based additive homomorphic encryption

A homomorphic encryption (HE) of *x* enables computing the encryption of $F(x)$ without the knowledge of the decryption key. In this work, we use HE based on Ring Learning with Errors (RLWE), specifically, the BFV scheme [\[16\]](#page-13-10). The scheme has a set of public parameters HE.pp = $\{N, q, p\}$.

We leverage the following functions.

- KeyGen. Generate the RLWE key pair (sk, pk) where the secret key sk $\in \mathbb{A}_{N,q}$ and the public key pk $\in \mathbb{A}_{N,q}^2$.
- Encryption. We write RLW $E_{pk}^{N,q,p}(\hat{m})$ to denote the RLWE ciphertext of $\hat{m} \in A_{N,p}$ under the key pk. An RLWE ciphertext is given as polynomials tuple $(\hat{b}, \hat{a}) \in$ $\mathbb{A}^2_{N,q}$.
- Addition (\boxplus). Given two RLWE ciphertexts $ct_0 =$ (\hat{b}_0, \hat{a}_0) and $ct_1 = (\hat{b}_1, \hat{a}_1)$ that encrypts $\hat{m}_0 \in A_{N,p}$ and $\hat{m}_1 \in A_{N,p}$ respectively, the operation ct₀ ⊞ct₁ computes the RLWE tuple $(\hat{b}_0 + \hat{b}_1 \mod q, \hat{a}_0 + \hat{a}_1 \mod q)$ which can be decrypted to $\hat{m}_0 + \hat{m}_1$ mod *p*.
- Constant multiplication (\cdot). Given a RLWE ciphertext ct that encrypts \hat{m} and a plaintext \hat{c} , the operation ct $\cdot \hat{c}$ results in a ciphertext that encrypts $\hat{m} \cdot \hat{c}$.

In state-of-the-art MPC protocols, it is generally good practice to mixed the usage of different cryptogrpahic primitives to achieve best performance. In this work, we use the latticebased HEs to utilize the players' local computation power as much as possible, and switch to secret sharing where HE is unsuitable in terms of functionality. We write $\llbracket \vec{a} \rrbracket_l^H$ to de-
note an PIWE ciphertext held by *P*, but encrypted under note an RLWE ciphertext held by P_l but encrypted under P_{1-l} 's key, where the vector \vec{a} is arranged into a polynomial $\hat{a} = \sum_{i=0}^{N-1} \vec{a}[i]X^i$ before encryption (i.e., *coefficient encoding*).

HE to Arithmetic Share H2A [\[21\]](#page-14-9). A conversion from $\llbracket \vec{a} \rrbracket^H$ to the arithmetic share $\llbracket \vec{a} \rrbracket$ is done as follows: P_l
samples $\hat{\epsilon} \subset \mathbb{A}$ is and sends the sum $\epsilon t = \llbracket \vec{a} \rrbracket^H \sqcup \hat{\epsilon}$ to samples $\hat{r} \in_R \mathbb{A}_{N,q}$, and sends the sum $ct = [[\vec{a}]]_H^H \boxplus \hat{r}$ to the opposite Then $P_{k,q}$ decreate ct and outputs the coef the opposite. Then *P*1−*^l* decrypts ct and outputs the coefficients vector as the share $[\vec{a}]_{1-I}$. On the other hand, P_l sets $[\vec{a} \cdot \vec{a}]_{1}-[\vec{a} \cdot \vec{a}]_{1/A}$ mod 2^{ℓ} for all $i \in [M]$ $[\vec{a}[i]]_l = -[2^{\ell} \cdot \hat{r}[i]/q] \mod 2^{\ell}$ for all $i \in [N]$.

2.1.4 Batch private information retrieval

Private information retrieval (PIR) is a family of protocols that enable a client to query a server's database db with a private index *i*, such that the client obtains $db[i]$ without the server learning *i*. Researchers have proposed an extended variant where a client can query a set of indices $(I = \{i_1, i_2, \dots, i_m\})$ simultaneously, which is more efficient than naively issuing *m* independent queries on db, with the help of probabilistic batch codes (PBC) [\[2\]](#page-13-11).

The efficient PBC construction is based on cuckoo hashing. Using a 3-way cuckoo hashing we can encode the database into $M = 3n$ codewords distributed in $B = 1.5L$ buckets, with a small failure probability ($p = 2^{40}$), where *L* is the number of queries. After encoding, each query only requires to search in one bucket with the size of $\approx 2n/L$. Note that each query

requires to search the entire database with the size of *n* without PBC. As a result, batch PIR can significantly reduce the computation cost of each query.

For more details, the server computes all hash results for every index and duplicate the items to their index's hash result position. It will improve the size of database from *n* to 3*n*. When length of the hashing table is *B*, and the item in database will be mapped to *B* positions. Each of the table position has $\approx 3n/B$ items, which can be seen as a bucket. The client compute the cuckoo hashing function for its *L* queries, the *L* queries can be mapped into different positions in *B*. The client sends *B* queries and then each query can only be searched in the database of one bucket. The total workload of the server is 3*n*, but it can response many queries together. The amortized computation cost of each query is much lower than doing *L* independent queries.

2.2 ANNS based on clustering

There has been a lot of studies on ANNS to improve the search efficiency. They can be summarized as graph or tree based approaches, local sensitive hash based approaches, and clustering based approaches. In plaintext practices, graph-based approaches such as HNSW [\[29\]](#page-14-10) usually yield the best performance. However, HNSW is unfriendly for secure computation because it heavily relies on hierarchical RAM operations that cannot be run in parallel. Although clustering based algorithm performs averagely in plaintext, we find that it's quite efficient in secure computation (This fact is also observed in SANNS). To this end, we choose ANNS as the underlying plaintext approach for Panther.

Algorithm [1](#page-3-0) shows a typical workflow of clustering-based ANNS. It can be briefly described as follows:

- 1. Classify the database items into *t* clusters;
- 2. Select the top-*u* clusters whose cluster centers are nearest to the query point;
- 3. Retrieve points in the *u* clusters.
- 4. Select the top-*k* nearest points to the query point.

The algorithm above is friendly for secure computation because it could reduce a large amount of the searching space by dropping out distant clusters.

3 Problem Setting

3.1 Threat Model

Following [\[3,](#page-13-4) [9\]](#page-13-7), we provide 2-party computation security against a *static semi*-*honest* probabilistic polynomial time (PPT) adversary *A*. We consider a computationally bounded adversary *A* which corrupts with server or client at the start of protocol execution. Semi-honest security means that as long

Algorithm 1 Clustering based ANNS algorithm

Input Client inputs query: $\vec{q} \in \mathbb{R}^d$; Server inputs *t* clusters with each cluster with up to *m* points. A list of *n* IDs \vec{u} . All cluster centers $\{\vec{c}_1, ..., \vec{c}_t\}$.

Output *ID*1,...,*ID^k*

1: for $i := 1, ..., t$ do 2: $d_i := ||\vec{q} - \vec{c}_i||_2^2$ ▷ compute distance between query and cluster centers 3: end for 4: $V_c, I_c = \{v_1, ..., v_u\}, \{id_1, ..., id_u\} \leftarrow$ 5: ←TopK({*d*₁,...,*d*_{*t*}}, {1,2,...,*t*}, *u*) 6: ρ compute the ID of the top-*u* nearest clusters $P_c \leftarrow \bigcup_{i \in I_c} C$ ▷ retrieve points in nearest clusters 8: for $\vec{p} \in P_c$ do 9: $d_p := ||\vec{q} - \vec{p}||_2^2$

▷ compute distance between query and points

- 10: end for 11: $\{v_1, ..., v_k\}, \{\overline{ID}_1, ..., \overline{ID}_k\}$ ← 12: ← TopK({ d_p }_{$p \in P_c$}, { $ID(p)$ }_{$p \in P_c$}, k)
- 13: \triangleright compute the top- k nearest points in retrieved points

as the adversary *A* follows the protocol specification, all of its view could be simulated by a simulator given the corrupted party's input and output. It is suitable for many practical scene, e.g., both parties have the willing to collaborate, but need to run a secure protocol due to legal restrictions. Our protocol aims to improve the performance over prior works under the same threat model. Any semi-honest protocol can be transformed to be maliciously secure, although that may bring huge overhead.

3.2 Overview of private ANNS based on clustering

As mentioned in Section [2.2,](#page-2-0) we would like to build our private ANNS solution using the clustering based algorithm following SANNS [\[9\]](#page-13-7). In this way, the workflow of private ANNS is similar with the one described in Algorithm [1,](#page-3-0) but replaces all the operations with their MPC counterparts, namely, distance computation (Line 2,9), top-*k* selection (Line $4 \sim 5,11 \sim 12$) and point retrieval (Line 7). All the intermediate results between the operations are kept in secretshared form. In this way, no extra information will be leaked beyond the protocol output and the hyper parameters. This is known as the sequential modular composition theorem [\[7\]](#page-13-12), and is the rule-of-thumb in designing MPC protocols.

Among the three operations, distance computation could be done efficiently using leveled HE such as BFV. Indeed, it only

Operations In Algorithm 1	SANNS	Panther
Distance Compute (Line 2.9)	HF with Noise Flooding	HE with H2A
Point Retrieval (Line 7)	DORAM	PIR
$Top-k$ $(Line 4-5.11-12)$	Garbled Circuits	Secret Share + Garbled Circuits + Top-k Selection Network

Figure 1: Comparison between SANNS and Panther

constitute $10 \sim 20\%$ of the total time cost in SANNS. However the other two operations are quite expensive: point retrieval takes more than 50%, and top- k selection takes \sim 30%.

We briefly compare the methods used by Panther and SANNS for each operation in Figure [1,](#page-3-1) and explain them in detail in the next sections.

4 Point Retrievals via Batch PIR

In this section, we demonstrate how Panther avoids the oblivious RAM, which contributes more than 50% to the end-to-end improvement.

Recall the "point retrieval" procedure in Line 7, Algorithm [1.](#page-3-0) It is composed of multiple oblivious database reads whose query IDs (the cluster addresses) are in secret shared form, and the database (points for each cluster) is held solely by the server. SANNS [\[9\]](#page-13-7) uses DORAM [\[12\]](#page-13-8) to solve the problem. However, we notice that using DORAM is an overkill because it's designed for scenarios where both the query IDs and database are secret-shared. Indeed, if the server knows the database in clear, there exist more efficient methods such as PIR. Compared to distributed ORAM, PIR has a much lower communication cost at the price of slightly higher computation cost, which could easily be parallelized by multithreading. Nevertheless, we cannot directly adopt PIR because PIR additionally requires the client to know the cluster addresses in clear. To this end, we let the server locally shuffle the clusters before each query, so that the cluster addresses are just independent random values and safe to be revealed to the client without harming the database privacy. By employing such a "shuffle-and-reveal" strategy, we successfully reduce the problem to batch PIR, drastically reducing the communication cost.

There still exist several issues to solve, such as how to select a PIR scheme that is friendly for database shuffles, how to optimize the PIR encoding so that the shuffling proccess could be done at low cost, etc. Subsections below describe our endeavors to overcome these difficulties.

4.1 Details of Database Shuffling

In the PIR protocol, the client C is aware of the index of the retrieved item. However, in the secure ANNS scenario, if *C* knows which clusters have been retrieved, it may leak information of the dataset. E.g. if cluster 1 and cluster 2 are returned, *C* knows the two clusters are close to each other. To address this issue, we have to return the secret-shared indices of the top-*u* nearest clusters (Step 4-6 in Algorithm [1\)](#page-3-0). Let $\lbrack \lbrack id \rbrack$ denote one of these secret-shared indices. Note that we have $id = [id]_s + [id]_c \mod t$, where *t* is the number of clusters, and subscripts *s* and *c* denote the share holder *S* and *C*, respectively.

Querying a secret-shared index in a single-query PIR is manageable. A first attempt is to ask the server *S* to rotate the entire database $\lceil id \rceil_s$ steps, making the client's shared index $\llbracket id \rrbracket_c$ into a real index that can be used with any standard PIR protocols. This technique is used in [\[11\]](#page-13-13). However, what we need in ANNS is batch PIR rather than single-query PIR. Batch PIR often employs cuckoo hashing to reduce the amortized cost, which is not compatible with the database rotating method because of the difficulties to obliviously compute the cuckoo hashing on secret-shared values. In the plaintext cuckoo hashing algorithm, collision detection is both necessary and straightforward, but when computing obliviously, the collision detection outcomes must remain private.

To address the challenges of secret-shared indices in batch PIR, we propose employing a shuffle-and-reveal approach. In detail, let the indices of database be $ID =$ $(id_1, id_2, ...id_n)$. Before each query, *S* selects a new random permutation $\delta : \mathbb{Z}_n^n \mapsto \mathbb{Z}_n^n$, and locally applies δ on *ID*, obtaining $\delta(ID) = (id_{1}, id_{2}, ...id_{n})$. Then *S* and *C* together run some selection (e.g. top-*k*) function. Let the output be $\llbracket \mathbf{O} \rrbracket = (\llbracket idI_{o_1} \rrbracket, \llbracket idI_{o_2} \rrbracket, \ldots \llbracket idI_{o_k} \rrbracket).$ *S* discloses its share $\llbracket \mathbf{O} \rrbracket$ to C . We have the following Lemma: to *C*. We have the following Lemma:

Lemma 1. *For any adversary A that corrupts the client C, the simulator randomly samples* $P = (p_1, p_2, \ldots, p_k)$ *from* (1,2,..*n*) *without replacement, and hands P over to A. A cannot distinguish O from P.*

The proof is easy to see since random sampling without replacement is the same with random permutation. In other words, our shuffle-and-reveal approach does not decrease security, thus *O* could safely be used by *C* for subsequent PIR queries.

One more issue is that PIR protocols often return additional items that may affect the database privacy. This is allowed in PIR but not allowed in ANNS because by definition PIR does not guarantee anything about the database privacy. In Section [6](#page-7-0) we will show how to transfer the PIR results into secret-shared form to protect both parties' privacy. Interestingly, since we will pick top-*k* items in the end, returning additional items will only make ANNS more accurate.

Figure 2: Different encoding methods. If we encode different items into a single polynomial (up), we need to re-encode each time we shuffle. If we encode different items into separate polynomials (down), we only need to save an index map $\{4,3,1,2\}$ to implement efficient shuffling.

4.2 Choosing a Shuffling-friendly PIR Protocol

From a practical perspective, we aim to ensure that the shuffle operation does not significantly impact performance. Unfortunately, for efficient PIR protocols with preproccessing such as SimplePIR [\[20\]](#page-14-11) and PIANO [\[51\]](#page-15-12), they need a considerable amount of offline hints stored at the client side, which need to be updated according to any database changes. It is acceptable in their cases due to the assumption that the database is less frequently updated. However, in our scenario, we need to reshuffle the entire database for each query, essentially making the "offline" cost an "online" cost, completely undermining their performance advantages.

Other batch PIR protocols, such as PIRANA [\[25\]](#page-14-12) and BatchPIR [\[33\]](#page-14-13), demonstrate high efficiency but are not friendly to shuffle operations, because both protocols encode multiple items into a single polynomial in a SIMD manner, necessitating a complete re-encoding of the polynomial if item positions are to be shuffled. The SIMD encoding in homomorphic encryption could be time-consuming, so we wish to avoid redoing such encoding process for each query.

Consequently, an appropriate PIR protocol for our scheme must facilitate both efficient shuffling and encoding without leaking additional database information. Fortunately, there exists a line of PIR protocols that do not need preprocessing, and allow direct coefficient encoding instead of SIMD encoding, such as SealPIR [\[2\]](#page-13-11), OnionPIR [\[32\]](#page-14-14), and Spiral [\[31\]](#page-14-15). Although Spiral is recognized as the state-of-the-art RLWE-based PIR protocol, its advantage lies in the size of PIR results, which is not the bottleneck of our protocol since we already have low communication cost. Under our context of batch PIR, where

the size of each bucket is approximately 2^{11} , we found that the earlier SealPIR protocol performs best. Keeping our requirements in mind, we present a comprehensive comparison of various PIR protocols in Table [2,](#page-6-0) and discuss necessary customizations in the subsequent section.

4.3 Further Optimizing the PIR Protocol

Item encoding. We encode the points of each cluster into coefficients of a single BFV plaintext polynomial. As a result, shuffling the database is just modifying the index map, and does not require any polynomial modification. Figure [2](#page-4-0) gives a toy example. In more detail, for each point vector $\vec{p} = (p_1, p_2, ..., p_d)$, we encode each element of the vector into one separate coefficient, while lots of PIR protocols consider the data as a string and ignore its own structures. Separate each element is helpful for additive secret sharing, as will be shown in [6.1.](#page-8-0) In one plaintext, we can encode *m* points with *d* dimensions. Because of $m \times d$ is large enough in our scenarios, this encoding method will not waste much space. For larger datasets, if there are more points in each cluster, we can encode them into multiple plaintext polynomials.

Support 2^k plaintext modulus. After retrieving the data, we utilize the additively shared data to compute distances. Our goal is to represent these distances within the ring \mathbb{Z}_{2^k} for the comparison phase. This choice is motivated by the observation that comparison protocols operate more efficiently in the ring \mathbb{Z}_{2^k} than in the prime field \mathbb{Z}_p , particularly since modulo reduction in \mathbb{Z}_{2^k} is nearly free. The SealPIR protocol [\[2\]](#page-13-11) inherently supports coefficient encoding without the constraint of having a prime plaintext modulus. However, a naive approach necessitates that the plaintext modulus be set as an odd number to facilitate the inverse operation in the expansion step of SealPIR. After expansion, the constant coefficient of the plaintext polynomial will be $m = 2^l$, which must be multiplied by the modular inverse of *m* modulo *p* to recover the value 1; this process is referred to as the reverse operation. If we set the plaintext modulus to 2^k , we are unable to find the inverse of *m*.

Fortunately, we have discovered that multiplying the entire ciphertext by the inverse of *m* modulo the ciphertext modulus *q* can also be effective. This approach implicitly requires that *q* be an odd modulus, a condition typically satisfied in RLWEbased homomorphic encryption, which uses moduli $q \equiv 1$ mod 2*N*. We outline this idea in Algorithm [2](#page-5-0) and present our findings below.

Lemma 2. *Algorithm [2](#page-5-0) securely expands the input ciphertext* $\llbracket x^i \rrbracket^H$ *to a vector of ciphertexts* $\langle \llbracket 0 \rrbracket^H, \cdots, \llbracket 1 \rrbracket^H, \cdots, \llbracket 0 \rrbracket^H \rangle$, K *where the i-th entry encrypts* 1 *and all other entries encrypt* 0*.*

Proof. We only show the correctness argument here. Let *e* be the noise of input query $\llbracket x^i \rrbracket$ \mathbb{I}^H , and B_{sub} be the noise growth

Input *query* = $\llbracket x^i \rrbracket^H$
Quiput a vector of *n* **Output** a vector of *n* ciphertexts $\langle [\![0]\!]^H, \cdots, [\![1]\!]^H, \cdots, [\![0]\!]^H$,
where the *i* th entry enery to 1 and all others enery of 0. where the *i*-th entry encrypts 1 and all others encrypt 0. 1: Find smallest $m = 2^l$ such that $m \ge n$ 2: *inverse* ← m^{-1} (mod *q*) \triangleright where *q* is ciphertext modulus 3: *query*′ ← *query* ·*inverse* 4: return NAIVEEXPAND(*query*′)

▷ The *NaiveExpand* proposed in [\[2\]](#page-13-11) with further details in appendix [A.](#page-15-13)

of *EXPAND*, which depends only on the HE parameters. If we multiply the ciphertext with the inverse of *m* mod *q*, then the noise v_{in} can be represented as $m^{-1}e$. After *EXPAND* operation v_{out} will not be influenced because m^{-1} can be eliminated.

$$
v_{out} = 2^{l} v_{in} + 2(2^{l} - 1)B_{sub}
$$

= $mv_{in} + 2(2^{l} - 1)B_{sub}$
= $mm^{-1}e + 2(2^{l} - 1)B_{sub}$
= $e + 2(2^{l} - 1)B_{sub}$

 $\llbracket x^i \rrbracket^H$ can be represented as $(\Delta x^i + \hat{e} - \hat{a} \cdot \text{sk}, \hat{a})$. After mul-
ligation we get $(\Delta m^{-1}x^i + m^{-1}\hat{a} - m^{-1}\hat{a})$ ok $m^{-1}\hat{a}$. After tiplication, we get $(\Delta m^{-1}x^{i} + m^{-1}\hat{e} - m^{-1}\hat{a}\cdot\text{sk},m^{-1}\hat{a})$. After *EXPAND*, the coefficient of x^i will multiplied by *m*, then m^{-1} can be used to inverse it to get $\llbracket 1 \rrbracket^H$. \Box

5 Optimized Top-*k* Selection

5.1 Mixed Primitives Approximate Top-*k*

The performance of the top-*k* selection protocol is highly dependent on the efficiency of the comparison protocol Π*Compare*. There are generally two kinds of MPC primitives to facilitate secure comparisons: garbled circuits (GC) and secret sharing (SS). The performance metrics of state-of-the-art comparison protocols in two-party computation are presented in Table [3.](#page-6-1) State-of-the-art SS-based protocols utilize oblivious transfer (OT) to compute comparison results, and it benefits from advancements in silent OT [\[5\]](#page-13-14), resulting in significantly reduced communication costs. However, it still necessitates multiple communication rounds, which could become a bottleneck in scenarios with high latency, particularly for tasks with high depths such as top-*k*. In contrast, GC-based protocols incur substantial communication costs but require a constant number of communication rounds, regardless of circuit complexity. By effectively balancing communication costs with the number of communication rounds and leveraging the advantages of both protocol types, we can substantially enhance comparison efficiency.

	Coefficient Encoding	Shuffle friendly	Comm.	Comp.
SealPIR $[2]$			ж	ж
Spiral $[31]$				
OnionPIR [32]				
FastPIR $[1]$				
SimplePIR $[20]$, PIANO $[51]$	٠			
BatchPIR [33], PIRANA [25]				

Table 2: Advantages and disadvantages of Related PIR Protocols. The communication and computation columns indicate the comparison against SealPIR under database with a small size.

	Comm. cost	Rounds
GC-based Π_{cmp} [35]	$4\lambda\ell$ bits	O(1)
SS-based Π_{cmp} (<i>m</i> = 4) [21]	$< 11\ell$ bits	$O(\log \ell)$
GC-based Naive Top- <i>k</i>	$O(nk)$ Π_{cmp}	O(1)
GC-based Top-k network	$O(n(\log k)^2) \prod_{cmp}$	O(1)

Table 3: Communication cost of different protocols. λ is the security parameter (usually $\lambda \ge 128$). *m* is the parameter in SS-based compare protocol.

SANNS proposed an effective method for enhancing the top-*k* algorithm by computing approximate top-*k* results. This approach reduces the number of comparison operations from $O(nk)$ to $O(n+1k)$. The approximate top-*k* algorithm can be summarized as follows:

- (a). Shuffle the input *n* items and split it into *l* bins.
- (b). Compute the minimum within each bin using GC.
- (c). Calculate top-*k* of the above *l* minimum items using GC to get the final result.

By carefully studying the algorithm flow and testing, we found that indeed Step (c) performs best with GC, since SS-based top-*k* will consume too many communication rounds. However, Step (b) is better suited for SS, because the operations in different bins can be parallelized, sharing the same round complexity. We employ an SS-based protocol to compute the minimum within each bin. This approach requires only $O((\log \ell + 1) \log(n/l))$ communication rounds and incurs a communication cost of 11*n*ℓ bits, which performs significantly better in various network conditions.

5.2 A Better Top-*k* Selection Network

Although the approximate top-*k* selection algorithm can significantly reduce the costs associated with the comparison phase, it still requires the computation of accurate top-*k* results at step (c). The SANNS algorithm employs a naive top-*k* method, which invokes the min function *k* times to obtain accurate top-*k* results, resulting in *O*(*nk*) comparison opera-

Figure 3: Top-*k* selection network demo ($k = 3, n = 15$)

tions. This approach is sub-optimal for both GC-based and SS-based protocols.

We build on the bitonic top-*k* selection network proposed by [\[37\]](#page-14-17) with two improvements:

- Extend the algorithm to support arbitrary *k* and *n* value, without potentially expensive padding to nearest powerof-two values;
- Integrate an odd-even merge sort network to bootstrap the initial states of top-*k* selection with optimal comparison gates.

Figure [3](#page-6-2) illustrates our construction with an example, with details described in Algorithm [6](#page-15-14) and Algorithm [4.](#page-7-1) We give a brief description of how we extend the bitonic sorting to top-*k* selection below. Suppose we have a bitonic sequence $\vec{S} = \langle S_1, S_2, \cdots, S_{2k} \rangle$, where $S_1 \leq \cdots \leq S_i \geq \cdots \geq S_{2k}$ for some $i \in [1, 2k]$. The first step is to compare the pairs of elements (S_1, S_{k+1}) , (S_2, S_{k+2}) , \cdots , (S_k, S_{2k}) , if the first element is larger than the second element, then swap them. After this step will get two bitonic sequences $\langle S_1, \dots, S_k \rangle$ and $\langle S_{k+1},\cdots,S_{2k}\rangle$, where all the elements in the first bitonic sequence are smaller than any of the elements in the second bitonic sequence. If we want to compute the top-*k* min elements, we can easily prune the second sequence in the result. Afterwards, we follow a standard bitonic merge process to

Algorithm 3 Top-*k* selection algorithm

Input $\vec{v} = \{v_1, ..., v_2\}, \vec{id} = \{id_1, ..., id_n\}, k$ Output $\{v_1, ..., v_k\}, \{ID_1, ..., ID_k\}$ 1: \vec{v} *,* \vec{u} ← LOCALSORT(\vec{v} *,* \vec{u} *, n, k*) 2: $l := n/k$ 3: while *l* > 1 do 4: **for** $i := 1, ..., |l/2|$ **do** 5: $l, r := (i-1)k + 1, ik$ 6: *l* $\langle f, r' := (i-1)k + \lceil l/2 \rceil + 1, ik + \lceil l/2 \rceil$ 7: $\vec{v}[l:r], \vec{id}[l:r] \leftarrow$ $\leftarrow k$ **MERGE(** $\vec{v}[l:r]$ **,** $\vec{id}[l:r]$, $\vec{v}[l';r'], \vec{id}[l';r'], k$) 8: $\vec{v}[l:r], \vec{id}[l:r] \leftarrow$ \leftarrow BITONICMERGE($\vec{v}[l:r], \vec{id}[l:r], k$) 9: end for 10: $l := [l/2]$ 11: \vec{v} , $\vec{id} := \vec{v}[1 : lk], \vec{id}[1 : lk]$ 12: end while 13: **return** \vec{v} , \vec{id}

Table 4: Comparison of the number of compare-and-swap operation between bitonic sort and odd-even merge sort, where *n* represents the number of value to be sorted.

completely sort the first sequence. With pruning, we reduce to a bitonic sequence with length *k*, which has been sorted by the bitonic sort algorithm. The next step is to concatenate two length-*k* bitonic sequences into one bitonic sequence with length 2*k*, and repeat the above process.

Finally, to bootstrap the selection network with sorted length-*k* sequences, we utilize odd-even merge sort network for the optimal instantiation of Π*sort* in Algorithm [4.](#page-7-1) Although it has the same asymptotic complexity as bitonic sort, oddeven merge sort produces less comparison operations, which we detail in Table [4](#page-7-2) for concrete parameters. Roughly speaking, we could achieve a 10% reduction in communication cost with this optimization in our scenario.

Our optimized top-*k* selection network has much better asymptotic complexity $O(n \log k \log k)$ compared to previous alternatives (Table [3\)](#page-6-1), with also higher concrete efficiency as we will show in the experiments.

Algorithm 4 Function in top-*k* selection algorithm

38: end function

6 Putting Everything Together

The distance computation, PIR and top-*k* operations needs to be connected with secret-shared intermediate values to form an end-to-end MPC application. However, multiple steps (Line 4,7,9 in Algorithm [1\)](#page-3-0) in Panther result in outputs that requires further treatment. First, the resulting ciphertext in SealPIR is in a "double encryption" format, presenting a challenge for post-processing into secret-shared values. Second, homomorphic ciphertexts contain noises that could potentially reveal information about the plaintext, requiring the noise flooding [\[14\]](#page-13-16) countermeasure (which will increase the HE parameters) before secret-shared. For efficiency reasons, the homomorphic computations in ANNS usually run on a relatively low plaintext bitwidth (e.g.,8), making the noise flooding step especially expensive.

This section delves into the intricacy of post processing these ciphertext in an optimal manner, seamlessly chaining all the operations together.

6.1 Post-processing PIR results

Section [4](#page-3-2) elaborates our proposal of using PIR to retrieve points. However, there remains a critical unresolved issue for its integration into the pipeline of the secure ANNS scheme. The retrieved points should not be revealed to the client, and should be converted into a form that is appropriate for subsequent distance computation. As one of the most efficient PIR protocols, SealPIR operates in a two-dimensional manner and produces doubly-encrypted ciphertext $\begin{bmatrix} \begin{bmatrix} \vec{a} \end{bmatrix}^H_I \end{bmatrix}^H_I$ as results. This restricts us from directly applying the H2A trick on such ciphertext, due to the one-bit error that H2A might introduce to the first-layer ciphertext $\llbracket \vec{a} \rrbracket^H_1$ that is segmented into several
reduncional components. As a consequence, $\llbracket \vec{a} \rrbracket^H$ might be polynomial components. As a consequence, $\llbracket \vec{a} \rrbracket^H$ might be destroyed and can not be destruined correctly when the one bit destroyed and can not be decrypted correctly when the one-bit error takes place in the middle of the assembled ciphertext.

The problem stems from the practice of parameter choices during usage of SealPIR. Let (*p*,*q*) denote the plaintext modulus and ciphertext modulus of the first dimension, and let (p', q') denote those of the second dimension. Generally speaking, in most applications, *q* would be larger than *p* ′ in order to accommodate sufficient precision for *p* and the computation, rendering it necessary to break the first-layer ciphertext into several components modulo *p* ′ .

In our scenario, we observe that with a careful selection of BFV parameters, we can still apply H2A to secret-share the retrieved points. The idea is to perform a modulus switch on the first-layer ciphertext from modulo q to modulo q_1 with the restriction of $p' = q_1$. Note that q_1 could be a sub modulus of *q* that is composed of a chain of RNS moduli $\{q_1, \dots, q_n\}$. In this setting, a first-layer ciphertext polynomial can be fully accommodated in a plaintext polynomial. Although H2A would still introduce one-bit error to the first-layer ciphertext, it is absorbed into the low-end noise. We present the details in Algorithm [5](#page-8-1) for combining a two-dimensional PIR with the above level-2 H2A conversion.

Fix one-bit error. Although the H2A protocol can successfully re-share the retrieved points between the server and client, there still exists one-bit error that can impact accuracy. This error appears in the point value, not the distance value,

Algorithm 5 PIR with Level-2 H2A

Input

- 1: First dim.: HE.pp₁ = $\{N, \mathbf{q}, p\}$, where and \mathbf{q} = ${q_1, \cdots, q_n}.$ $p = 2^t$
- 2: Second dim.: $\text{HE}.\text{pp}_2 = \{N, \mathbf{q}', p'\}, \text{let } p' = q_1.$
- 3: Client inputs index (id_1, id_2) ;
- 4: Server inputs the encoded dataset which has *n* plaintext pt. Each item \vec{x} _{*i*, *j*} has been encoded in coefficient as $pt_{i,j} \in \mathbb{A}_{N,q}$

Output Client gets $[\![\vec{x}_{id_1, id_2}]\!]_c$; Server gets $[\![\vec{x}_{id_1, id_2}]\!]_s$

- 5:
- 6: Generate Query:
- 7: *C* encrypts (id_1, id_2) to $ct^1 = RLWE_{pk}^{N, q, p}(x^{id_1})$ and $ct^2 =$ $RLWE_{pk}^{N, \mathbf{q}', p'}(x^{id_2});$
- 8: *C* sends ciphertexts to *S* as query.
- \mathbf{Q}
- 10: Response: 10: **Respon**
11: $m := \sqrt{}$ *n* ▷ the size of each dimension 12: $\{ct_1^1, ..., ct_m^1\} \leftarrow EXPAND((ct^1))$ 13: \triangleright *S* obviously expands 14: $\{s_0, s_1, ..., s_m\} \leftarrow \{\text{RLWE}_{\text{pk}}^{N, \textbf{q}, p}(0), ..., \text{RLWE}_{\text{pk}}^{N, \textbf{q}, p}(0)\}$ 15: **for** $i := 1, ..., m$ **do** 16: **for** $j := 1, ..., m$ **do** 17: $s_i \leftarrow s_i \boxplus (\text{ct}_i^1 \cdot \text{pt}_{i,j})$ 18: end for 19: end for 20: **for** $i := 1, ..., m$ **do** 21: $s_i \leftarrow \text{MODULUSSWITCH}(s_i, q_1) \Rightarrow \text{Now } s_i \in \mathbb{A}_{q_1}^2$ 22: end for $23: \{(\hat{a}_0, \hat{b}_0), ..., (\hat{a}_m, \hat{b}_m)\} \leftarrow \{s_0, ..., s_m\}$ 24: \triangleright where $\hat{a}_i, \hat{b}_i \in \mathbb{A}_{q_1}$ 25: $\{ct_1^2, ..., ct_m^2\} \leftarrow EXPAND((ct^2))$ 26: 27: { w_0, w_1 } ← {RLWE^{N,q,p}(0),RLWE^{N,q,p}(0)} 28: for $i := 1, ..., m$ do 29: $w_0 \leftarrow w_0 \boxplus (\text{ct}_i^2 \cdot \hat{a}_i)$ 30: $w_1 \leftarrow w_1 \boxplus (\mathsf{ct}_i^2 \cdot \hat{b}_i)$ 31: end for 32: $w_0 \leftarrow \text{MODULUSSWITCH}(w_0, q_1')$ 33: $w_1 \leftarrow \text{MODULUSSWITCH}(w_1, q_1')$ 34: \hat{r} , $w'_0 \leftarrow H2A(w_0)$ 35: *S* sends w'_0 and w_1 to *C* while saving \hat{r} . 36: 37: Extract: 38: $\hat{u} \leftarrow Dec(w_0)$ $\Rightarrow \hat{u} + \hat{r} = \hat{a} + \hat{e}$ 39: $\hat{b} \leftarrow Dec(w_1)$ $40: C: \llbracket \vec{x}_{id_1, id_2} \rrbracket_c \leftarrow Dec((\hat{u}, \hat{b}))$
 $41: S: \llbracket \vec{x}_{id_1}, \vec{d}_2 \rrbracket_{d_1} \cup \llbracket \hat{s}_{id_1}, \vec{p}_{id_2} \rrbracket_{d_1}$ $41: S: \llbracket \vec{x}_{id_1, id_2} \rrbracket_s \leftarrow \lceil \hat{r} \cdot \frac{\hat{p}}{q_1} \rceil$ $\frac{p}{q_1}$

and must not be ignored. To address this, we propose a protocol to fix one-bit error. The main idea is to pad the lower bits of the value with 0*b*01. If a one-bit error occurs, the lower bits

will become 0*b*00 or 0*b*10 (secret-shared), neither of which will affect the higher bits. We can then use a truncate and reduce protocol in [\[35\]](#page-14-16) to remove the two lower bits and obtain the correct value.

Extend with known MSB After retrieving multiple points with H2A protocols, the elements of points have been secret shared in mod $p = 2^{\ell_1}$, where p is the first dimension plaintext modulus. Then we input this data to compute distance, the distance result will be additive shared in 2^{ℓ_2} . If each element of a d -dimension point has been quantized to ℓ bits, then $\ell_2 = 2\ell + \lceil \log d \rceil$. A simple way to set $\ell_1 = \ell_2$ will incur higher cost for points retrieval to accommodate larger ℓ_2 . We use a smaller ℓ_1 to retrieve points and then extend them to ℓ_2 . The extension protocol in [\[27\]](#page-14-18) requires only two rounds of communication and $O(\ell_2-\ell_1)$ bits, when the most significant bit (MSB) is known.

6.2 Optimized distance computation

For *d*-coordinate points, we recall that the Euclidean distance $||\vec{u} - \vec{v}||_2^2 = ||\vec{u}||_2^2 + ||\vec{v}||_2^2 - 2\langle \vec{u}, \vec{v} \rangle$. When *P*₀ holds \vec{u} and *P*₁ holds a set of points $\{\vec{v}_j\}_{j \in [N]}, P_0$ would encode each coordinate as a constant polynomial $\hat{f}_i = \vec{u}[i]$ and sends the encryption of d polynomials to P_1 who encodes polynomials $\hat{g}_i = \sum_j \vec{v}_j[i]x^j$, for $i \in [d]$. The inner products $\langle \vec{u}, \vec{v}_j \rangle$ are given in coefficients of $\hat{h} = \sum_i \hat{f}_i \hat{g}_i$. A simple masking of \hat{h} by sending the encryption of $\hat{h} - \hat{r}$ back to P_0 with a randomly sampled $\hat{r} \in A_{N,p}$ is not sufficient to *securely* produce additive secret sharing, because the noise contained in the ciphertext reveals information about \hat{g}_i , the plaintext used in constant homomorphic multiplication. In the SANNS framework [\[9\]](#page-13-7), in order to eliminate such leakage, they rely on the heavy noise flooding technique [\[14\]](#page-13-16) to hide such information in large noise. However, this enforces them to use large parameters for the BFV scheme, i.e., $N = 2^{13}$ and 180-bit ciphertext modulus *q*, resulting in suboptimal performance.

To optimize the performance, we leverage the H2A technique introduced in [\[21\]](#page-14-9), which essentially combines masking and noise flooding into a single step by sampling a random polynomial $\hat{r} \in_R \mathbb{A}_{N,q}$ in the ciphertext domain and sends back $\llbracket \vec{h} \rrbracket^H \boxplus \hat{r}$ to P_0 . P_1 locally constructs its own share from \hat{r} as shown in Section [2.1.3.](#page-1-1) It is not difficult to argue the information hiding property of this approach since *r*ˆ completely randomizes the ciphertext, including the noise. The drawback is that this method will lead to a probabilistic one-bit error for a reconstructed plain value (Appendix C, [\[21\]](#page-14-9)), but it hardly affects our ANNS scenario, as also demonstrated by our experiments. After all, in most of the time, the low-end bits will be truncated before entering garbled circuits, the only exception is that if the values reach the maximum, the low end bits will also affect the high end bits. In the end, this optimization enables us to use a small parameter set with $N = 2^{11}$ and a

	N	D	q
Distance	2048	2^{24}	$\{54\}$ bits
PIR HE .pp ₁	4096	2^{12}	${24,36,37} bits$
PIR HE.pp2	4096	$\log p \approx 24$	$\{36,36,37\}$ bits

Table 5: BFV parameters HE.pp

54-bit modulus *q*. Details can be found in Table [5.](#page-9-0)

7 Evaluation

7.1 Implementation and Environment

We have fully implemented Panther in C++ using the Secret-Flow ^{[1](#page-0-0)} and EMP-toolkit [\[45\]](#page-15-15) libraries. The SPU library within SecretFlow was employed to develop the secret-sharing-based comparison and distance computation protocols. Additionally, we utilized the mpir part in the PSI library of SecretFlow to construct the PIR building block, and the emp-sh2pc framework [2](#page-0-0) to implement the top-*k* selection network using garbled circuits protocols. These components were integrated to create a comprehensive end-to-end implementation, which we subsequently evaluated in terms of performance.

The experiments presented in this paper were conducted on cloud instances equipped with 128 processors (across two physical sockets), operating at 2.70 GHz and 256 GB of RAM. Both the client and server are running under 64 threads.

To simulate various network conditions, we adjusted the bandwidth using the traffic control utility in Linux. Following the SANNS network setting, we performed our benchmarks under two network conditions: a Local Area Network (LAN) with a bandwidth of 4000 Mbps and a 1ms round-trip time (RTT), and a Wide Area Network (WAN) with a bandwidth of 320 Mbps and a 74ms RTT.

7.2 Other Settings

BFV Parametes choices. In our end-to-end implementation, we utilize the BFV homomorphic encryption scheme as a foundational component at various stages. By leveraging the H2A protocol, we can employ more user-friendly parameters, resulting in enhanced performance. Table [5](#page-9-0) summarizes the parameters used in our experiments. To support the Level-2 H2A protocol, it is necessary to set the second dimension plaintext modulus *p* equal to the first dimension *q*1.

Datasets. We evaluate our scheme using the SANNS algorithm parameters in the context of the following datasets:SIFT $(n = 1 000 000, d = 128)$, Deep1B-1M $(n = 1 000 000,$

¹<https://github.com/secretflow>

²<https://github.com/emp-toolkit/emp-sh2pc>

Figure 4: Comparison of proposed protocols with SANNS in terms of running time and communication costs.

 $d = 96$, Deep1B-10M ($n = 10000000$, $d = 96$) and Amazon $(n = 2^{20}, d = 50)$. Our objective is to demonstrate the performance of Panther across datasets of varying sizes and dimensions.

Hyperparameters. To ensure fairness in our evaluation, we utilized the same hyperparameters (such as the number of clusters, and the number of bins in approximated top-*k*) as those used in SANNS [\[9\]](#page-13-7) for assessing our secure ANNS scheme. Our goal is to present the evaluation results to demonstrate that our secure building block is more efficient than SANNS. While using a more efficient plaintext clustering algorithm or changing the hyperparameters might enhance the performance of both Panther and SANNS, our focus remains on a consistent comparison in the aspect of secure computation.

7.3 Microbenchmarks

7.3.1 Distance Computation

In Section [6.2,](#page-9-1) we discuss our methods for mitigating noise leakage following homomorphic computation. Noise flooding can also achieve similar objectives. We compare the benchmark performance of our method with that of noise flooding in Table [6,](#page-10-0) which illustrates the advantages of our approach. For noise flooding, we adopt the same parameters as SANNS, specifically $N = 8192$, $p = 2^{24}$ and $q \approx 180$ bits. We conduct the micro-benchmarks using a single thread. To compute the

Distance Compute $(n = 100 000, d = 128)$					
	LAN	WAN	Comm.		
Noise Flooding [9]		549 ms 1858 ms	41.7 MB		
Ours	293 ms	541 ms	5.21 MB		
	PIR $(n = 2048)$				
	LAN	WAN	Comm.		
Noise Flooding	1064 ms	1138 ms	859 KB		

Table 6: Performance comparison of noise flooding and ours

distance, we analyze one point versus 100,000 points, each having a dimensionality of 128, quantized to 8 bits. Additionally, we evaluate the level-2 H2A protocol with PIR applied to a database containing 2048 items, each representing a separate bucket size in batch PIR. Each item is encoded into an individual polynomial.

7.3.2 Top-k selection benchmark

We evaluate our novel protocols for approximate top-*k* selection by combining garbled circuits with secret-sharingbased comparison protocols and exact top*k* by a better top-*k* selection network (refer to Section [5\)](#page-5-1). Figure [4](#page-10-1) shows the evaluation result. In this evaluation, we test $n = 1,000,000$

	Distances Dataset			$Top-k$		Points Retrieval		Total		
		SANNS	Panther	SANNS	Panther	SANNS	Panther	SANNS	Panther	
	LAN(s)	0.11	0.23	1.29	0.35	2.17	1.69	3.62	2.28	$1.6 \times$
INI	WAN(s)	1.41	0.50	16.1	6.34	27.1	3.64	45.3	10.7	$4.2 \times$
$\boldsymbol{\omega}$	Comm.	56.7 MB	5.57 MB	645 MB	66.4 MB	1.06 GB	45.4 MB	1.77 GB	118 MB	\downarrow 93 %
	LAN(s)	0.09	0.22	1.24	0.39	1.84	1.66	3.23	2.28	$1.4 \times$
D.1M	WAN(s)	1.10	0.41	15.5	6.80	23.0	3.58	40.4	11.0	$3.7 \times$
	Comm.	44.1 MB	4.57 MB	620 MB	74.9 MB	920 MB	46.3 MB	1.58 GB	126 MB	\downarrow 92 %
	LAN(s)	0.12	0.79	4.81	0.82	6.39	5.32	11.3	6.94	$1.6 \times$
M0T	WAN(s)	1.48	1.71	60.2	12.8	79.9	8.43	142	23.1	$6.1 \times$
\blacksquare	Comm.	59.4 MB	9.23 MB	2.35 GB	227 MB	3.12 GB	82.1 MB	5.53 GB	318 MB	\downarrow 94 %
	LAN(s)	0.05	0.09	1.06	0.38	1.23	1.29	2.29	1.78	$1.3 \times$
Amzn	WAN (s)	0.61	0.31	13.2	6.67	15.4	3.27	28.7	10.5	$2.7 \times$
	Comm.	24.4 MB	2.83 MB	528 MB	73.2 MB	617 MB	44.4 MB	1.12 GB	121 MB	\downarrow 89 %

Table 7: Performance comparison of Panther and SANNS. Since SANNS is not open-sourced, we estimate their time costs with communication/bandwidth, neglecting their computational cost, thus the numbers in this table are in great favor of them.

24-bit shared integers for mixed primitives approximate top-*k*. We assess for $k \in \{5, 10, 20, 50, 100\}$. The approximate top- k algorithm is designed to return $(1 - \delta) \cdot k$ entries accurately, where we set $\delta = 0.01$ to examine performance metrics. As detailed in SANNS, we utilize a single thread to compute the results. The blue line SANNS in Figure [4](#page-10-1) is data from their paper. Even in this single-threaded context, our protocol demonstrates a performance improvement by a factor of up to $5\times$. The grey line SANNS estimation in Figure [4](#page-10-1) is estimated by communication/cost. When using more threads, our protocol performance can be much better, while the GC-based algorithm can't break through the bandwidth limit.

We test $n = 10,000$ 24-bit shared integers for exact top*k* and assess for $k \in \{5, 10, 20, 50, 100\}$. These parameters are closer to the actual usage in clustering-based ANNS. We reimplement the exact top-*k* algorithm in SANNS by empsh2pc to get data under these parameters. As k increases, our solution has a significant advantage. The performance can be increased by $2.3 \times$ to $7.5 \times$.

7.4 End-to-End Evaluation

We present a comparison of end-to-end schemes in Table [7.](#page-11-0) For SANNS, the numbers in their paper (Table 2, [\[9\]](#page-13-7)) are based on a flexible bandwidth (Section 5.1, [\[9\]](#page-13-7)), thus cannot be reproduced for comparison. Instead, we report their communication cost as outlined in their paper, and estimate their runtime using the communication cost divided by bandwidth. Note that we completely ignore their computational cost, and put ourselves at a disadvantage.

As is shown, our approach greatly enhances the performance of each component. We summarize the combined distance computation costs in the column of "Distances". It is important to note that our estimated runtime appears slower than that of SANNS under LAN because we count in the

CPU computational time. However, we are still more efficient under WAN due to smaller HE parameters. For the top-*k* component, we merge the approximate top-*k* results into those of the top-*k* performance. This indicates that our method performs significantly better in both LAN and WAN settings. For point retrievals, our batch PIR approach significantly outperforms SANNS's DORAM-based approach, reducing the communication costs by up to 97%.

In summary, our scheme consistently outperforms SANNS in various datasets. We achieve a reduction in communication costs of up to 94%, while improving search times by up to $6.1 \times$. Note that if we take their computational cost into account, our improvements will be even more significant.

Comparison with two-server works. Our performance are even comparable with [\[36\]](#page-14-8) which works under a two-server setting. They answer a query on Deep1B-10M in 6.13 seconds, which is similar with us under LAN and $3.8\times$ faster than us under WAN. Note that their two servers reside in a same region of AWS regardless of WAN or LAN.

7.5 Accuracy

Since Panther and SANNS are both implementing the same plaintext routine of Algorithm [1,](#page-3-0) and share the same set of hyper parameters such as plaintext bitwidth, number of bins in approximate top-*k*, and shape of clusters, we expect that the accuracy of Panther directly follows SANNS, which achieves 10-NN accuracy of 0.9 (9 out of 10 points are correct on average, Section 5.3 of [\[9\]](#page-13-7)).

There are only two points that might lead to differences in results between Panther and SANNS. The first is the possibility of one-bit error in the H2A step of distance computation (Section [6.2\)](#page-9-1). Note that the one-bit error in the H2A step of PIR has been addressed in Section [6.1.](#page-8-0) Although the

one-bit error happens in the least significant bit and is unlikely to affect the accuracy, we want to experimentally show the results. However, SANNS didn't provide enough details to reproduce the same experiment result in their paper, most importantly, we don't have the plaintext clustering algorithm they used. As a result, we pick the *k*-means clustering algorithm in [\[13\]](#page-13-17) and use Panther with noise flooding instead of H2A to simulate SANNS. Table [8](#page-12-0) shows its accuracy comparison with Panther. Results indicate that H2A indeed does not affect accuracy. Note that even we tried our best to tune the hyper parameters, either aligned or not aligned to the settings described in SANNS, the accuracy is still slightly below 0.9, that may because the plaintext clustering algorithm in [\[13\]](#page-13-17) is not optimized for the task.

The second difference is that the batch queries in PIR naturally return additional indices, specifically, Panther returns 50% more points in the points retrieval step. The extra points will be fed into the top-*k* step so the number of output samples does not change. It's likely that feeding more candidate clusters to top-*k* could improve the accuracy, but the beneficial effect is unpredictable because the additional 50% clusters are picked at random. We choose to stay conservative and do not claim the accuracy improvements.

Dataset	Panther with- Panther out H ₂ A error	
SIFT-1M $Deep1B-1M$	88.85% 89.16%	88.85% 89.16%

Table 8: Accuracy comparison between Panther without H2A errors (to simulate [\[9\]](#page-13-7)) and Panther

8 Related Work

Approximated Nearest Neighbors Search Approximated nearest neighbors (ANN) search has aboard applications. In the "plaintext world", lots of studies aim to enhance search efficiency while maintaining high accuracy. Including(1) graphbased [\[17,](#page-13-3) [28,](#page-14-4) [29,](#page-14-10) [34\]](#page-14-6), (2) tree-based [\[44\]](#page-15-2), (3) quantizationbased [\[22,](#page-14-2) [30,](#page-14-5) [46\]](#page-15-3) and (4) hashing-based [\[26,](#page-14-3) [42,](#page-15-1) [50\]](#page-15-4). Most of the algorithms employ same strategy: initially, they generate a set of candidate nearest neighbors, followed by identifying the actual *k* nearest neighbors from within this candidate set. The difference is their ways how to generate candidates neighbors. Refer to the benchmark [\[4\]](#page-13-18), the graph based algorithm usually performs better. It provides a more complex structures to organize the vectors. The complex structures bring big challenge for secure computation. The *k*-means clustering based algorithm may not be the optimal choice, but it is outperforming in secure computation.

Secure *k*-NNs under Other Security Models Servan-Schreiber et al. [\[36\]](#page-14-8) propose an efficient private *k*-NNs called Preco in two non-colluding server. In two server setting, it is known to us existing a lot of much more efficient protocols. But the assumption that two server hold the same dataset while don't collude to each other may be impractical in many real-world scenarios. This protocol utilize local sensitive hash (LSH) to calculate the index of the approximated nearest subsets. Then it used distributed point function DPF to retrieve a subset with all of the nearest points from all subsets. It can be free of comparison but need to retrieve from a large number of subsets (i.e. $n = 2^{45}$), which ensures each subset has only one point. For single server, the cost in retrieval process will become intolerable. Besides, this scheme will leak part of information about the dataset.

Song et al. [\[40\]](#page-15-8) follow the Preco's direction but retain the comparison operations, which can make the number of subset smaller. However, the comparison method in their scheme doesn't consider the network latency. They only show the performance under a low latency network. What's more, even using comparison after retrieve subsets, the number of subsets still too large (e.g. $n = 2^{25}$) for single server setting.

Non-interactive Secure *k*-NNs Shaul et al. [\[39\]](#page-15-5) propose a *k*-NNs classifier based on FHE. It's non-interactive but take hours to query a database containing thousands of points. Zuber et al. [\[52\]](#page-15-7) propose a fully homomorphic *k*-NNs algorithm based on TFHE. The computation is fully non-interactive but has quadratic complexity with regard to the size of the database (e.g. $O(n^2)$). Note that clustering based secure *k*-NNs algorithm has only $O(\sqrt{n})$ complexity.

It's intuitively the non-interactive algorithm can't run in sublinear time. It must search all data to avoid information leakage. It's much less efficient than SANNS [\[9\]](#page-13-7). Although using multi-thread CPU, it can't support to query in a database of millions because of the highest computation cost.

9 Conclusion

In this paper, we propose an efficient private ANNS framework Panther. It can significantly improve the performance than previous single server ANNS framework. We design novel protocols to retreive point with efficient batch PIR instead of ORAM. We propose a mixed-primitive top-*k* approach with optimized selection network to enhance the top-*k* selection performance. We carefully design optimized usage of homomorphic encryption. Evaluated the end-to-end implementation in different size of dataset, Panther is superior to the previous work in both communication cost and computation cost.

References

- [1] Ishtiyaque Ahmad, Yuntian Yang, Divyakant Agrawal, Amr El Abbadi, and Trinabh Gupta. Addra: Metadataprivate voice communication over fully untrusted infrastructure. In Angela Demke Brown and Jay R. Lorch, editors, *15th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2021, July 14-16, 2021*. USENIX Association, 2021.
- [2] Sebastian Angel, Hao Chen, Kim Laine, and Srinath T. V. Setty. PIR with compressed queries and amortized query processing. In *2018 IEEE Symposium on Security and Privacy, SP 2018, Proceedings, 21-23 May 2018, San Francisco, California, USA*, pages 962–979. IEEE Computer Society, 2018.
- [3] Gilad Asharov, Shai Halevi, Yehuda Lindell, and Tal Rabin. Privacy-preserving search of similar patients in genomic data. *Proc. Priv. Enhancing Technol.*, 2018(4):104–124, 2018.
- [4] Martin Aumüller, Erik Bernhardsson, and Alexander Faithfull. Ann-benchmarks: A benchmarking tool for approximate nearest neighbor algorithms. *Information Systems*, 87:101374, 2020.
- [5] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, Peter Rindal, and Peter Scholl. Efficient two-round ot extension and silent non-interactive secure computation. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 291–308, 2019.
- [6] Rui Cai, Chao Zhang, Lei Zhang, and Wei-Ying Ma. Scalable music recommendation by search. In Rainer Lienhart, Anand R. Prasad, Alan Hanjalic, Sunghyun Choi, Brian P. Bailey, and Nicu Sebe, editors, *Proceedings of the 15th International Conference on Multimedia 2007, Augsburg, Germany, September 24-29, 2007*, pages 1065–1074. ACM, 2007.
- [7] Ran Canetti. Security and composition of multiparty cryptographic protocols. *Journal of CRYPTOLOGY*, 13:143–202, 2000.
- [8] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM computing surveys (CSUR)*, 41(3):1–58, 2009.
- [9] Hao Chen, Ilaria Chillotti, Yihe Dong, Oxana Poburinnaya, Ilya P. Razenshteyn, and M. Sadegh Riazi. SANNS: scaling up secure approximate k-nearest neighbors search. In Srdjan Capkun and Franziska Roesner, editors, *29th USENIX Security Symposium, USENIX Security 2020, August 12-14, 2020*, pages 2111–2128. USENIX Association, 2020.
- [10] Ningning Cui, Xiaochun Yang, Bin Wang, Jianxin Li, and Guoren Wang. Svknn: Efficient secure and verifiable k-nearest neighbor query on the cloud platform. In *2020 IEEE 36th International Conference on Data Engineering (ICDE)*, pages 253–264. IEEE, 2020.
- [11] Ghada Dessouky, Farinaz Koushanfar, Ahmad-Reza Sadeghi, Thomas Schneider, Shaza Zeitouni, and Michael Zohner. Pushing the communication barrier in secure computation using lookup tables. In *24th Annual Network and Distributed System Security Symposium, NDSS 2017, San Diego, California, USA, February 26 - March 1, 2017*. The Internet Society, 2017.
- [12] Jack Doerner and Abhi Shelat. Scaling ORAM for secure computation. In Bhavani Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*, pages 523–535. ACM, 2017.
- [13] Matthijs Douze, Alexandr Guzhva, Chengqi Deng, Jeff Johnson, Gergely Szilvasy, Pierre-Emmanuel Mazaré, Maria Lomeli, Lucas Hosseini, and Hervé Jégou. The faiss library. 2024.
- [14] Léo Ducas and Damien Stehlé. Sanitization of fhe ciphertexts. In *Advances in Cryptology–EUROCRYPT 2016: 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part I 35*, pages 294–310. Springer, 2016.
- [15] Daniel Escudero. An introduction to secret-sharingbased secure multiparty computation. *IACR Cryptol. ePrint Arch.*, page 62, 2022.
- [16] Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. *Cryptology ePrint Archive*, 2012.
- [17] Cong Fu, Chao Xiang, Changxu Wang, and Deng Cai. Fast approximate nearest neighbor search with the navigating spreading-out graph. *Proc. VLDB Endow.*, 12(5):461–474, 2019.
- [18] Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, Qianyu Guo, Meng Wang, and Haofen Wang. Retrieval-augmented generation for large language models: A survey. *CoRR*, abs/2312.10997, 2023.
- [19] Changhee Hahn and Junbeom Hur. Efficient and privacypreserving biometric identification in cloud. *ICT Express*, 2(3):135–139, 2016.
- [20] Alexandra Henzinger, Matthew M. Hong, Henry Corrigan-Gibbs, Sarah Meiklejohn, and Vinod Vaikuntanathan. One server for the price of two: Simple and fast single-server private information retrieval. In Joseph A. Calandrino and Carmela Troncoso, editors, *32nd USENIX Security Symposium, USENIX Security 2023, Anaheim, CA, USA, August 9-11, 2023*, pages 3889–3905. USENIX Association, 2023.
- [21] Zhicong Huang, Wen-jie Lu, Cheng Hong, and Jiansheng Ding. Cheetah: Lean and Fast Secure Two-Party Deep Neural Network Inference. *USENIX Security*, 2022.
- [22] Hervé Jégou, Matthijs Douze, and Cordelia Schmid. Product quantization for nearest neighbor search. *IEEE Trans. Pattern Anal. Mach. Intell.*, 33(1):117–128, 2011.
- [23] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proc. IEEE*, 86(11):2278–2324, 1998.
- [24] Patrick S. H. Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrievalaugmented generation for knowledge-intensive NLP tasks. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.
- [25] J. Liu, J. Li, D. Wu, and K. Ren. Pirana: Faster multiquery pir via constant-weight codes. In *2024 IEEE Symposium on Security and Privacy (SP)*, pages 43–43, Los Alamitos, CA, USA, may 2024. IEEE Computer Society.
- [26] Wanqi Liu, Hanchen Wang, Ying Zhang, Wei Wang, Lu Qin, and Xuemin Lin. EI-LSH: an early-termination driven I/O efficient incremental c-approximate nearest neighbor search. *VLDB J.*, 30(2):215–235, 2021.
- [27] Wen-jie Lu, Zhicong Huang, Zhen Gu, Jingyu Li, Jian Liu, Kui Ren, Cheng Hong, Tao Wei, and Wenguang Chen. Bumblebee: Secure two-party inference framework for large transformers. *IACR Cryptol. ePrint Arch.*, page 1678, 2023.
- [28] Yury Malkov, Alexander Ponomarenko, Andrey Logvinov, and Vladimir Krylov. Approximate nearest neighbor algorithm based on navigable small world graphs. *Inf. Syst.*, 45:61–68, 2014.
- [29] Yury A. Malkov and Dmitry A. Yashunin. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE Trans. Pattern Anal. Mach. Intell.*, 42(4):824–836, 2020.
- [30] Yusuke Matsui, Yusuke Uchida, Hervé Jégou, and Shin'ichi Satoh. A survey of product quantization. *ITE Transactions on Media Technology and Applications*, 6(1):2–10, 2018.
- [31] Samir Jordan Menon and David J. Wu. SPIRAL: fast, high-rate single-server PIR via FHE composition. In *43rd IEEE Symposium on Security and Privacy, SP 2022, San Francisco, CA, USA, May 22-26, 2022*, pages 930– 947. IEEE, 2022.
- [32] Muhammad Haris Mughees, Hao Chen, and Ling Ren. Onionpir: Response efficient single-server PIR. In Yongdae Kim, Jong Kim, Giovanni Vigna, and Elaine Shi, editors, *CCS '21: 2021 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, Republic of Korea, November 15 - 19, 2021*, pages 2292– 2306. ACM, 2021.
- [33] Muhammad Haris Mughees and Ling Ren. Vectorized batch private information retrieval. In *44th IEEE Symposium on Security and Privacy, SP 2023, San Francisco, CA, USA, May 21-25, 2023*, pages 437–452. IEEE, 2023.
- [34] Yun Peng, Byron Choi, Tsz Nam Chan, Jianye Yang, and Jianliang Xu. Efficient approximate nearest neighbor search in multi-dimensional databases. *Proc. ACM Manag. Data*, 1(1):54:1–54:27, 2023.
- [35] Deevashwer Rathee, Mayank Rathee, Rahul Kranti Kiran Goli, Divya Gupta, Rahul Sharma, Nishanth Chandran, and Aseem Rastogi. Sirnn: A math library for secure RNN inference. In *42nd IEEE Symposium on Security and Privacy, SP 2021, San Francisco, CA, USA, 24-27 May 2021*, pages 1003–1020. IEEE, 2021.
- [36] Sacha Servan-Schreiber, Simon Langowski, and Srinivas Devadas. Private approximate nearest neighbor search with sublinear communication. In *43rd IEEE Symposium on Security and Privacy, SP 2022, San Francisco, CA, USA, May 22-26, 2022*, pages 911–929. IEEE, 2022.
- [37] Anil Shanbhag, Holger Pirk, and Samuel Madden. Efficient top-k query processing on massively parallel hardware. In *Proceedings of the 2018 International Conference on Management of Data*, pages 1557–1570. ACM, 2018.
- [38] Mark Shaneck, Yongdae Kim, and Vipin Kumar. Privacy preserving nearest neighbor search. In *Machine Learning in Cyber Trust: Security, Privacy, and Reliability*, pages 247–276. Springer, 2009.
- [39] Hayim Shaul, Dan Feldman, and Daniela Rus. Secure k-ish nearest neighbors classifier. *Proc. Priv. Enhancing Technol.*, 2020(3):42–61, 2020.
- [40] Shang Song, Lin Liu, Rongmao Chen, Wei Peng, and Yi Wang. Secure approximate nearest neighbor search with locality-sensitive hashing. In Gene Tsudik, Mauro Conti, Kaitai Liang, and Georgios Smaragdakis, editors, *Computer Security - ESORICS 2023 - 28th European Symposium on Research in Computer Security, The Hague, The Netherlands, September 25-29, 2023, Proceedings, Part III*, volume 14346 of *Lecture Notes in Computer Science*, pages 411–430. Springer, 2023.
- [41] Ebrahim M Songhori, Siam U Hussain, Ahmad-Reza Sadeghi, and Farinaz Koushanfar. Compacting privacypreserving k-nearest neighbor search using logic synthesis. In *Proceedings of the 52nd Annual Design Automation Conference*, pages 1–6, 2015.
- [42] Yifang Sun, Wei Wang, Jianbin Qin, Ying Zhang, and Xuemin Lin. SRS: solving c-approximate nearest neighbor queries in high dimensional euclidean space with a tiny index. *Proc. VLDB Endow.*, 8(1):1–12, 2014.
- [43] Avery Wang. An industrial strength audio search algorithm. In *ISMIR 2003, 4th International Conference on Music Information Retrieval, Baltimore, Maryland, USA, October 27-30, 2003, Proceedings*, 2003.
- [44] Jingdong Wang, Naiyan Wang, You Jia, Jian Li, Gang Zeng, Hongbin Zha, and Xian-Sheng Hua. Trinaryprojection trees for approximate nearest neighbor search. *IEEE Trans. Pattern Anal. Mach. Intell.*, 36(2):388–403, 2014.
- [45] Xiao Wang, Alex J. Malozemoff, and Jonathan Katz. EMP-toolkit: Efficient MultiParty computation toolkit. <https://github.com/emp-toolkit>, 2016.
- [46] Xiang Wu, Ruiqi Guo, Ananda Theertha Suresh, Sanjiv Kumar, Daniel N. Holtmann-Rice, David Simcha, and Felix X. Yu. Multiscale quantization for fast similarity search. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 5745–5755, 2017.
- [47] Kun Yang, Chengliang Tian, Hequn Xian, Weizhong Tian, and Yan Zhang. Query on the cloud: improved privacy-preserving k-nearest neighbor classification over the outsourced database. *World Wide Web*, 26(4):1747–1774, 2023.
- [48] Andrew Chi-Chih Yao. Protocols for secure computations (extended abstract). In *23rd Annual Symposium on Foundations of Computer Science, Chicago, Illinois, USA, 3-5 November 1982*, pages 160–164. IEEE Computer Society, 1982.
- [49] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *27th Annual Symposium on Foundations of Computer Science, Toronto, Canada, 27-29 October 1986*, pages 162–167. IEEE Computer Society, 1986.
- [50] Yuxin Zheng, Qi Guo, Anthony K. H. Tung, and Sai Wu. Lazylsh: Approximate nearest neighbor search for multiple distance functions with a single index. In Fatma Özcan, Georgia Koutrika, and Sam Madden, editors, *Proceedings of the 2016 International Conference on Management of Data, SIGMOD Conference 2016, San Francisco, CA, USA, June 26 - July 01, 2016*, pages 2023–2037. ACM, 2016.
- [51] Mingxun Zhou, Andrew Park, Wenting Zheng, and Elaine Shi. Piano: Extremely simple, single-server pir with sublinear server computation. In *2024 IEEE Symposium on Security and Privacy (SP)*, pages 55–55. IEEE Computer Society, 2023.
- [52] Martin Zuber and Renaud Sirdey. Efficient homomorphic evaluation of k-nn classifiers. *Proc. Priv. Enhancing Technol.*, 2021(2):111–129, 2021.

Appendix

A Naive Expand

Algorithm 6 Naive expand in [\[2\]](#page-13-11) (Figure 3. line 6 - 12)

Input *query* = $\llbracket x^i \rrbracket^H$
Quiput a vector of *n* **Output** a vector of *n* ciphertexts $\langle [\![0]\!]^H, \cdots, [\![1]\!]^H, \cdots, [\![0]\!]^H$,
where the *i* th entry enery to 1 and all others enery of 0. where the *i*-th entry encrypts 1 and all others encrypt 0. 1: Find smallest $m = 2^l$ such that $m \ge n$ 2: $ciphertexts \leftarrow [query]$ 3: for $j = 0$ to $l - 1$ do 4: **for** $k = 0$ to $2^{j} - 1$ do 5: $c_0 \leftarrow ciphertexts[k]$ 6: $c_1 \leftarrow c_0 \cdot x^{-2j}$ 7: $c'_k \leftarrow c_0 + \text{Sub}(c_0, N/2^j + 1)$ 8: $c_{k+2j}^{j} \leftarrow c_1 + \text{Sub}(c_1, N/2^{j} + 1)$ 9: end for 10: $ciphertexts \leftarrow [c'_0, ..., c'_{2^{j+1}-1}]$ 11: end for 12: return *ciphertexts*