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

Jingyu Li
Ant Group
Hangzhou, China
ljy404490@antgroup.com

Zhicong Huang
Ant Group
Hangzhou, China
zhicong.hzc@antgroup.com

Min Zhang
Institute of Software, Chinese Academy of Sciences
Beijing, China
zhangmin@iscas.ac.cn

Cheng Hong*
Ant Group
Beijing, China
vince.hc@antgroup.com

Jian Liu
Zhejiang University
Hangzhou, China
jian.liu@zju.edu.cn

Tao Wei
Ant Group
Hangzhou, China
lenx.wei@antgroup.com

Wenguang Chen
Ant Group
Beijing, China
yuanben.cwg@antgroup.com

Abstract

Approximate nearest neighbor search (ANNS), also known as vector search, is an important building block for various applications, such as recommendation systems, biometric authentication, 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 suffer from high communication cost (Chen et al., USENIX Security 2020) or work under a stronger 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, secret-sharing, garbled circuits, and homomorphic encryption. We made extensive experiments using Panther on four public datasets, showing that Panther could answer an ANNS query on 10 million points in 18 seconds with 284 MB of communication. This is more than 7.8× faster and 20× more compact than Chen et al..

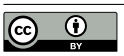
CCS Concepts

• **Security and privacy → Privacy-preserving protocols**; *Cryptography*; Database and storage security.

Keywords

privacy, approximate nearest neighbor search

*Cheng Hong is the corresponding author.



This work is licensed under a Creative Commons Attribution 4.0 International License.
CCS '25, Taipei
© 2025 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-1525-9/2025/10
https://doi.org/10.1145/3719027.3765190

ACM Reference Format:
Jingyu Li, Zhicong Huang, Min Zhang, Cheng Hong, Jian Liu, Tao Wei, and Wenguang Chen. 2025. Panther: Private Approximate Nearest Neighbor Search in the Single Server Setting. In *Proceedings of the 2025 ACM SIGSAC Conference on Computer and Communications Security (CCS '25), October 13–17, 2025, Taipei.* ACM, New York, NY, USA, 18 pages. https://doi.org/10.1145/3719027.3765190

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 aims to find the top-k closest points to a query point. k-NNS has numerous applications, such as recommendation systems [9, 56], image recognition [31], biometric authentication [7], and Retrieval-Augmented Generation (RAG) [23, 32] in large language models. In the case of large, high-dimensional databases where exact k-NNS is prohibitively expensive, the problem is often relaxed to approximate nearest neighbors search (ANNS), which returns points highly likely to be among the top-k closest neighbors, rather than guaranteeing the exact top-k. Compared to exact k-NNS, there exist several more efficient algorithms [22, 28, 35, 37–39, 45, 54, 57, 59, 63] for ANNS.

Normally, ANNS algorithms require knowing the database and the query point in plaintext 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 [4] and fingerprint recognition [16, 24, 34]. To this end, proposing private ANNS services based on secure multi-party computation (MPC) would be beneficial.

A considerable number of studies have been conducted along this line. Some of them [4, 51, 53, 67] use a full database distance computation, and thus are only practical for small or low-dimensional databases. Some works [50] try to increase efficiency at the cost of extra information leakage. Another popular choice is to build the system under a multi-server setting [13, 48, 52, 60], 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 the additional assumption of non-colluding participants. However, for clients seeking private ANNS services, systems relying on such security assumptions may be less trustworthy than those in a single-server setting.

In USENIX Security 2020, Chen et al. [11] propose the milestone framework SANNS that scales to a database of 10 million entries in the single server setting. However, SANNS relies on several secure computation techniques with high communication overhead, such as distributed oblivious RAM (DORAM) [17] and garbled circuits (GC) [61]. As a result, SANNS requires several gigabytes of communication per query, which is not deployable over realistic network connections (Section 8.3, [48]).

Designing an efficient single server private ANNS is challenging due to two main reasons. First, a common approach in ANNS is to partition the database into clusters, so that top-k distance computations are only required on those clusters close to the query point, rather than the entire database. While reading the content of these clusters is straightforward in plaintext, it needs to be done obliviously in MPC, often necessitating costly primitives like DORAM. Second, computing the top-k elements in MPC is resource-intensive. For instance, secret sharing (SS)-based methods [20] can require numerous communication rounds, while GC-based methods, as used in SANNS, can lead to high communication costs. These challenges underscore the difficulty of the secure ANNS problem.

1.1 Our Contributions

This work focuses on an efficient pipeline for addressing the private ANNS problem in the single server setting. Through a careful integration of private information retrieval (PIR), secret sharing, 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 efficiently address the oblivious read problem in secure ANNS, leveraging batch PIR instead of the costly DORAM. This approach reduces the communication cost for this step by 97% $\sim$ 99%.
- We propose several efficient and nontrivial methods for the "PIR to secret shares" problem, including post-processing PIR results (which are homomorphic ciphertexts in special format), and operating under the ring Z_{2^k} in PIR. These are important building blocks when one wants to securely evaluate a function on the PIR results without disclosing the results, and could be of broader interest.
- We propose a mixed-primitive approach (combining SS and GC) for top-k selection, paired with a carefully designed selection network, reducing the communication cost by 85% $\sim$ 90%.
- We open-source Panther[1], a fully implemented framework for secure ANNS. We conducted extensive evaluations on the datasets SIFT-1M, Deep1B-1M, Deep1B-10M and Amazon, which are commonly used benchmarks in the ANNS literature. Results show that Panther achieves up to 7.8× faster than the state-of-the-art method [11] with the same accuracy, and the communication cost is reduced by more than one order of magnitude.

[1] https://github.com/AntCPLab/OpenPanther.git

2 Related Work

Approximated Nearest Neighbors Search. Approximated nearest neighbors (ANN) search has broad applications. In the "plaintext world", many studies aim to enhance search efficiency while maintaining high accuracy. Including(1) graph-based[22, 37, 38, 45], (2) tree-based[57], (3) quantization-based [28, 39, 59] and (4) hashing-based [35, 54, 63]. Most of the algorithms employ the 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 lies in their ways of generating candidate neighbors. Referring to the benchmark [6], the graph-based algorithm usually performs better. It introduces a more complex structure to organize the vectors. The complex structure brings a 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 ANNS under Two Non-colluding Servers. Servan-Schreiber et al. [48] propose an efficient private k-NNs called Preco in two non-colluding servers. In the two-server setting, it is well-known that there exist many significantly more efficient protocols. However, the assumption that two servers hold the same dataset but do not collude with each other may be impractical in many real-world scenarios. This protocol utilizes local sensitive hash (LSH) to calculate the index of the approximated nearest subsets. Then it used the distributed point function to retrieve a subset with all 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 that each subset has only one point. In the single-server setting, the cost in the retrieval process will become intolerable. In addition, this scheme will leak part of the information about the dataset. Song et al. [52] follow the Preco direction but retain the comparison operations, which can reduce the number of subsets. However, the comparison method in their scheme does not consider network latency. They only show performance under a low-latency network. What's more, even using comparison after retrieving subsets, the number of subsets is still too large (e.g. $n = 2^{25}$) for single server setting.

Secure ANNS without Server Privacy. Some studies only focus on ensuring query privacy, without protecting the privacy of the database on the server. Henzinger et al. [25] propose Tiptoe, a system that allows clients to search over hundreds of millions of documents, while revealing no information about their search query to the search engine's servers. Asi et al. [5] propose Wally improving the performance of Tiptoe by relaxing the privacy notion to differential privacy. This approach allows for the amortization of server computation and communication costs across multiple client requests. Both Tiptoe and Wally utilize clustering-based algorithms. Given that the database is public, it can let the client locally save and compute a mount of information to improve performance. Zhu et al. [66] propose a graph-based ANNS scheme called Compass in the setting where the client owns the database and outsources it to the server in encrypted form. Zhou et al. [65] propose a graph-based ANNS scheme called PACMANN in the setting where the database

is public, which achieves higher search quality and better performance. When database privacy is not a concern, the solution would naturally be more efficient, yet it cannot meet the requirements of all scenarios.

Non-interactive Secure k-NNs. Shaul et al. [51] propose a k-NNs classifier based on FHE. It is non-interactive but takes hours to query a database containing thousands of points. Zuber et al. [67] 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 the clustering-based secure k-NNs algorithm has only $O(\sqrt{n})$ complexity.

Although using multi-thread CPU, they are much less efficient than SANNS [11], and cannot efficiently support queries in a database of millions because of the high computational cost.

3 Preliminaries

In this section, we briefly introduce some building blocks used in Panther. The frequently used notation is summarized in Table 1.

3.1 Cryptographic Primitives

3.1.1 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 $[\![x]\!]_c$ and $[\![x]\!]_s$, where $[\![x]\!]_c$ is a random share of x held by client and $[\![x]\!]_s$ is a random share of x held by server. To reconstruct the value x, we compute the modulo addition, i.e., $x = [\![x]\!]_c + [\![x]\!]_s \bmod 2^\ell$.

For a boolean value $z \in \{0,1\}$, we use $[\![z]\!]_c^B$ and $[\![z]\!]_s^B$ to denote the shares of z such that $z = [\![z]\!]_c^B \oplus [\![z]\!]_s^B$. Also we omit the subscript and only write $[\![x]\!]$ or $[\![z]\!]^B$ when the ownership is irrelevant from the context.

3.1.2 Garbled circuit. Garbled circuit is a technique proposed by [62] 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 Sort.

3.1.3 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 [21]. The scheme has a set of public parameters $\mathsf{HE.pp} = \{N, q, p\}$. We leverage the following functions.

- **KeyGen.** Generate the RLWE key pair (sk, pk) where the secret key $\mathsf{sk} \in \mathbb{A}_{N,q}$ and the public key $\mathsf{pk} \in \mathbb{A}_{N,q}^2$.
- **Encryption.** We use $\mathsf{RLWE}_{\mathsf{pk}}^{N,q,p}(\hat{m})$ and $\mathsf{RLWE}_{\mathsf{sk}}^{N,q,p}(\hat{m})$ to denote the RLWE ciphertext of $\hat{m} \in \mathbb{A}_{N,p}$ under the keys pk and sk, respectively. An RLWE ciphertext is given as polynomial tuple $(\hat{b}, \hat{a}) \in \mathbb{A}_{N,q}^2$.
- **Addition** ($\boxplus$)**.** Given two RLWE ciphertexts $\mathsf{ct}_0 = (\hat{b}_0, \hat{a}_0)$ and $\mathsf{ct}_1 = (\hat{b}_1, \hat{a}_1)$ that encrypt $\hat{m}_0 \in \mathbb{A}_{N,p}$ and $\hat{m}_1 \in \mathbb{A}_{N,p}$ respectively, the operation $\mathsf{ct}_0 \boxplus \mathsf{ct}_1$ computes the RLWE tuple $(\hat{b}_0 + \hat{b}_1 \bmod q, \hat{a}_0 + \hat{a}_1 \bmod q)$ which can be decrypted to $\hat{m}_0 + \hat{m}_1 \bmod p$.

Notation	Description
C	client
$\mathcal{S}$	server
n	number of data points in the dataset
t	number of clusters in the dataset
m	number of points in each cluster
u	number of clusters to retrieve
B	number of buckets in batch PIR ($B = 1.5u$ in our setting)
d	dimensionality of the data points
k	number of retrieved data points as answer
N	polynomial modulus degree in BFV scheme
p	plaintext modulus in BFV scheme
q	ciphertext modulus in BFV shceme
$\mathbb{A}_{N,q}$	the set of integer polynomials $\mathbb{Z}_q[X]/(X^N+1)$
$\hat{x}$	a polynomial
$\vec{x}$	a vector
$\vec{x}[i]$	the i-th component of $\vec{x}$
$\|\|\vec{x}\|\|_2^2$	$\vec{x}[1]^2 + \vec{x}[2]^2 + ... + \vec{x}[n]^2$
$\langle \vec{x}, \vec{y} \rangle$	$\vec{x}[1] \cdot \vec{y}[1] + \vec{x}[2] \cdot \vec{y}[2] + ... + \vec{x}[n] \cdot \vec{y}[n]$
$\vec{x}[i{:}j]$	$\langle \vec{x}[i], \vec{x}[i+1], ..., \vec{x}[j] \rangle$
Π	secure protocol
$[\![x]\!]$	additive secret-shared value x
ℓ	bitwidth
$[n]$	$\{1, 2, ..., n\}$

Table 1: Summary of frequent notations.

- **Constant multiplication** $(\cdot)$**.** Given an RLWE ciphertext ct that encrypts $\hat{m}$ and a plaintext $\hat{c}$, the operation $\mathsf{ct} \cdot \hat{c}$ results in a ciphertext that encrypts $\hat{m} \cdot \hat{c}$.

In state-of-the-art MPC protocols, it is generally good practice to mix the usage of different cryptographic primitives to achieve best performance. In this work, we use the lattice-based 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 $[\![\vec{v}]\!]_s^H$ to denote an RLWE ciphertext held by $\mathcal{S}$ but encrypted under C's key, where the vector $\vec{v}$ is arranged into a polynomial $\hat{v} = \sum_{i=1}^N \vec{v}[i] X^{i-1}$ before encryption (i.e., *coefficient encoding*).

HE to Arithmetic Share H2A**[27].** To protect the privacy of P_l's input during the computation phase, we convert the result to secret sharing. The conversion from $[\![\vec{v}]\!]_s^H = (\hat{b}, \hat{a})$ to an arithmetic share $[\![\vec{v}]\!]$ is done as follows: $\mathcal{S}$ samples $\hat{r} \in_R \mathbb{A}_{N,q}$ and computes $\mathsf{ct}' = (\hat{b} + \hat{r}, \hat{a}) \boxplus \mathsf{RLWE}_{\mathsf{pk}}^{N,q,p}(0)$. The addition of a fresh ciphertext $\mathsf{RLWE}_{\mathsf{pk}}^{N,q,p}(0)$ is intended to re-randomize $\hat{a}$, while the addition of $\hat{r}$ to $\hat{b}$ serves to mask the first polynomial. $\mathcal{S}$ sends back the result ct′ to the opposite. C decrypts ct′ and outputs the coefficients vector as the share $[\![\vec{v}]\!]_c$. On the other hand, $\mathcal{S}$ sets $[\![\vec{v}[i]]\!]_s = -\lfloor p \cdot \hat{r}[i]/q \rceil \bmod p$ for all $i \in [N]$. According to analysis in [27], the uniform random $\hat{r}$ will introduce a 1-bit error into the arithmetic share, but

it effectively protects $\mathcal{S}$'s input privacy without expensive noise flooding[2].

3.1.4 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, ensuring that the client obtains db$[i]$ without revealing i to the server. Some works have designed protocols using probabilistic batch codes (PBC) [2] to allow a client to query a set of indices ($I = \{i_1, i_2, \cdots, i_L\}$) simultaneously. It can amortize computation costs to greatly improve efficiency compared to making L separate queries in db.

Angel et al.[2] proposed an efficient PBC construction based on cuckoo hashing. For a database of size n, if C makes L batch queries, $\mathcal{S}$ can use a 3-way cuckoo hashing to encode the database into $B = 1.5L$ buckets. Each bucket can be viewed as a subset of the database, with the total number of items across all buckets being $M = 3n$. Then C sends one query in each bucket and then receives a total of B responses. It can recover all results for arbitrary L queries with a small failure probability ($p = 2^{-40}$). Each query on the server only needs to search within a subset of a database, approximately $2n/L$ in size. It is important to note that each query traditionally requires scanning the entire database with size n. Meanwhile, the client needs to run B ($> L$) queries, which might increase communication overhead. However, in our scenario, getting more responses means inputting additional points during the final top-k computation, which could potentially enhance the accuracy of the search results. Despite a larger response size, server privacy could be enhanced by converting responses to secret sharing, which we will explain in Section 5.2.

3.2 ANNS based on Clustering

There have been many studies on Approximate Nearest Neighbor Search (ANNS) aimed at improving search efficiency. These studies can be broadly categorized into three types of approaches: graph- or tree-based, locality-sensitive hashing (LSH) based, and clustering-based. In plaintext settings, graph-based methods, such as Hierarchical Navigable Small World (HNSW) graphs[38], typically deliver the best performance. However, HNSW is not well suited for secure computation, due to its heavy reliance on hierarchical RAM operations, which cannot be executed in parallel. In contrast, while clustering-based algorithms exhibit average performance in plaintext settings, we observe that they are highly efficient in secure computation environment, a finding also noted in SANNS [11].

A typical workflow of clustering-based ANNS can be briefly described as follows:

I. Partition the database into clusters based on similarity.
II. Select a subset of clusters whose centroids are closest to the query point.
III. Retrieve the points belonging to the selected clusters.
IV. Search the top-k points closest to the query point from the retrieved points.

The algorithm described above is particularly advantageous for secure computation because it can significantly reduce the search space by excluding distant clusters. Therefore, following the approach in SANNS, we adopt clustering-based ANNS as the underlying plaintext method for Panther.

Panther assumes a dataset pre-partitioned into clusters, and its correctness is dependent on the quality of the plaintext clustering algorithm performed by the server. While compatible with any clustering algorithm, we use k-means clustering of [18] in this paper.

[2]It's to note that HE does not automatically provide privacy for the P_I's input in its computation phase. Noise flooding [19]can be a viable but expensive solution to address the problem.

4 Technical Overview

4.1 Threat Model

Similarly to prior private ANNS works [4, 11], Panther provides secure two-party computation against a *static semi-honest* adversary $\mathcal{A}$. We consider a computationally bounded adversary $\mathcal{A}$, which corrupts either the server or the client at the start of a protocol execution and honestly follows the protocol specification. Semi-honest security is suitable for many practical scenarios, such as when both parties are willing to collaborate but cannot directly share their data in plaintext due to legal restrictions. We describe Panther under the *hybrid model*, where the subroutines are invocations of corresponding "ideal" functionalities. In this way, as long as the ideal functionalities are securely instantiated, the full protocol reveals no information beyond the hyperparameters (e.g., the number of clusters or the number of points in clusters) of the ideal functionalities. This adheres to the sequential modular composition theorem [10], and is the rule of thumb in designing MPC protocols.

Panther protects the privacy of both the client and the server. The server learns nothing about the query or the result, while the client learns nothing about the dataset beyond the ANNS result. The protocol is designed such that it reveals nothing beyond the output of the corresponding plaintext ANNS algorithm. While some studies [29] have explored whether a database can be reconstructed from ANNS query results, such attacks typically require access to a large number of queries. In our work, we do not consider leakage from the query results themselves. This assumption is reasonable, as the server could charge the client per query, which naturally limits the query rate and discourages adversarial querying.

4.2 Analysis of Prior Approaches and Insights

A key step in clustering-based ANNS involves retrieving points belonging to the selected closest clusters (Step III in Section 3.2). Performing this retrieval in MPC is a nontrivial task, as both the identifiers of the points to retrieve and the retrieved results (i.e., the coordinates of the points) must remain in secret-shared form to prevent extra leakage. This resembles a special oblivious read problem. Indeed, SANNS employs the DORAM protocol named Floram [17] to resolve the challenge.

However, Floram is inherently costly: according to [17], a single oblivious query on a database of ten thousand entries requires communication on the order of several megabytes. In the context of secure ANNS, where we need to retrieve hundreds or thousands of entries using such queries, such communication costs become prohibitive.

First failed attempt. A natural idea is to consider whether recent advancements in DORAM, such as DUORAM [55], could optimize

the oblivious read process. Compared to Floram, DUORAM significantly lowers communication costs. However, this attempt proved unfeasible because DUORAM relies on a protocol by Naor and Pinkas [43] that requires re-encrypting the entire database for each oblivious read, rendering the process highly inefficient for batch retrieval. We defer the detailed explanations to Appendix E.

Our observation. We found that employing DORAM is overkill for our purposes: DORAM is designed for scenarios where both the query indices and the database are secret-shared. However, in the context of secure ANNS, the server owns the database and has full knowledge of its contents in plaintext, rendering DORAM's additional complexity unnecessary. For such server-owned databases, there exist other methods to privately retrieve items such as PIR, which is significantly more efficient than DORAM. Nevertheless, we cannot directly apply PIR because it requires the client to know both the query indices and query results in clear, while in secure ANNS we require the query indices and query results to be secret-shared.

Second failed attempt. [15] proposed a method to query a secret-shared index in PIR. Let $[\![idx'_i]\!]$ denote one of these secret-shared query indices. We have $idx'_i = [\![idx'_i]\!]_s + [\![idx'_i]\!]_c \mod t$, where t is the number of clusters, and subscripts s and c indicate the shares held by the server $\mathcal{S}$ and the client C, respectively. [15] suggests asking the server $\mathcal{S}$ to rotate the entire database by $-[\![idx'_i]\!]_s$ steps, making the client's shared index $[\![idx'_i]\!]_c$ into a real index compatible with standard PIR protocols. However, ANNS requires batch PIR rather than single-query PIR. Batch PIR often leverages cuckoo hashing to reduce amortized costs, but this approach is incompatible with the database rotation method due to the challenges of obliviously computing cuckoo hashing over secret-shared values.

4.3 Design of Panther

Our "shuffle-and-reveal" approach. To address the challenges of secret-shared indices in batch PIR, we suggest utilizing the shuffle-and-reveal paradigm, which has been studied in prior studies [3, 30, 44][3]. We let $\mathcal{S}$ sample a random permutation to locally shuffle the database before serving each query. This allows the shuffled indices to be revealed to C, who can use them for subsequent PIR. The revealed indices are just pseudorandom values and thus do not disclose any information about the database, and a formal proof is given in Appendix F.2.

We provide an overview of our full protocol Π_{Panther} in Figure 1. For further details, please refer to Appendix D. The workflow closely resembles the plaintext ANNS workflow outlined in Section 3.2, with the following modifications: a shuffle step is added in Step 1, a revealing step is added in Step 4, and all subroutines are replaced with their secure counterparts. The formal definitions of the ideal functionalities $\mathcal{F}_{\text{ApproxTopK}}$, $\mathcal{F}_{\text{ExactTopK}}$ and $\mathcal{F}_{\text{BatchPIR}}$ invoked are provided in Appendix B.

The subroutines presented in Figure 1 can be categorized into three groups: **distance computation**, **point retrieval**, and **top-k selection**. Among these, point retrieval and top-k selection (including both approximate and exact variants) are the most expensive.

[3]Their database is in secret-shared form, so they have to use an expensive oblivious shuffle. In contrast, our database is held solely by the server, so we only need a cheap local shuffle.

[4]Clusters with less than m points are padded to m points.

Public Parameters: number of elements n, number of final search results k, number of clusters t, number of retrieved clusters u, number of returned points B by batch PIR, and other hyperparameters in ANNS and batch PIR.
Input: C inputs a query $\vec{q}$. $\mathcal{S}$ inputs the database db = $\langle(\vec{p}_1, \text{ID}(\vec{p}_1)), ..., (\vec{p}_n, \text{ID}(\vec{p}_n))\rangle$, where $\vec{p}_i$ is a point and $\text{ID}(\vec{p}_i)$ is its identifier. Both the query and the points have been quantized to the same bit length.
Output: C receives the IDs of the top-k nearest points to $\vec{q}$.

Setup: $\mathcal{S}$ locally partitions db into t disjoint clusters $\langle C_1, ..., C_t\rangle$, with each cluster containing up to m points[4]. The cluster centroids $\langle\vec{c}_1, ..., \vec{c}_t\rangle$ are chosen to approximately minimize the sum of distances between each centroid and all points in its cluster (e.g., via k-means). ▷ One-time preprocessing

---

1: $\mathcal{S}$ randomly shuffles the clusters and their centroids to generate: $\text{db}' = \langle(idx_1, C_1), ..., (idx_t, C_t)\rangle$ and $\text{db}'_{ctr} = \langle(idx_1, \vec{c}_1), ..., (idx_t, \vec{c}_t)\rangle$, where idx_i represents the index of the cluster C_i after the shuffle. ▷ Shuffle

2: $\mathcal{S}$ and C use leveled HE to securely compute the Euclidean distances between $\vec{q}$ and the t centroids in db'_{ctr}, obtaining t secret-shared pairs of indices and distances: $\langle([\![idx_1]\!], [\![dst_1]\!]), ..., ([\![idx_t]\!], [\![dst_t]\!])\rangle$, $dst_i = ||\vec{q} - \vec{c}_i||_2^2$.

3: $\mathcal{S}$ and C input these shared distances and indices to $\mathcal{F}_{\text{ApproxTopK}}$, which returns the shared indices of the approximately u closest clusters to $\vec{q}$: $\langle[\![idx'_1]\!], [\![idx'_2]\!], ..., [\![idx'_u]\!]\rangle$.

4: Reveal the u shared indices to C. ▷ Reveal shuffled indices

5: $\mathcal{S}$ inputs db' and C inputs the u indices into $\mathcal{F}_{\text{BatchPIR}}$, obtaining $B = 1.5u$ secret-shared clusters, each containing m points: $\langle([\![\vec{p'}_i]\!], [\![\text{ID}(\vec{p'}_i)]\!], [\![||\vec{p}_i||_2^2]\!])\rangle_{i=1}^{B\cdot m}$

6: $\mathcal{S}$ and C use leveled HE to securely compute the Euclidean distances between $\vec{q}$ and the $B \cdot m$ secret-shared points, obtaining $B \cdot m$ secret-shared distances and their corresponding IDs.

7: $\mathcal{S}$ and C input the $B \cdot m$ shared distances and IDs to $\mathcal{F}_{\text{ExactTopK}}$, which returns secret-shared IDs of the exact top-k smallest distances in the input.

8: Reveal the above k IDs to C.

Figure 1: Protocol Π_{Panther} for private ANNS (sketch).

For instance, in SANNS, point retrieval constitutes over 50% of the total runtime, while top-k selection accounts for approximately 30%. As part of our design strategy, we aim to minimize performance bottlenecks. After adopting the "shuffle-and-reveal" approach for point retrieval, top-k selection becomes the new communication bottleneck. We explore protocols with lower communication complexity, but these typically incur a higher number of communication rounds, leading to increased latency. To address this, we employ a protocol-friendly selection algorithm and combine the strengths of different protocols. We also find that the server's homomorphic

Subroutines in Fig. 1	SANNS [11]	Panther
Distance Computation (Step 2, 6)	HE with Noise Flooding	HE with H2A (§7.3)
Point Retrieval (Step 4 - 5)	DORAM	Batch PIR (§5, § 7.1, §7.2)
Top-k Selection (Step 3, 7)	Garbled Circuit	Mixed Primitive + Top-k Network (§6)

Table 2: Comparison between SANNS and Panther.

	Coefficient Encode	Shuffle Friendly	Comm.	Comp.
SealPIR [2]	✓	✓	–	–
Spiral [40]	✓	✓	✓	✗
OnionPIR [41]	✓	✓	✓	✗
FastPIR [1]	✗	✗	✓	✓
SimplePIR [26]	-	✗	✓	✓
PIANO [64]	-	✗	✓	✓
BatchPIR [42]	✗	✗	✓	✓
PIRANA [33]	✗	✗	✓	✓

Table 3: Advantages and disadvantages of Related PIR Protocols. The communication and computation columns indicate the comparison against SealPIR under a database with a small size (✓ and ✗ mean more and less efficient respectively).

evaluation is the main computational bottleneck. The choice of homomorphic encryption parameters significantly affects efficiency. To mitigate this, we propose a series of optimizations that enable secure computation with smaller parameters, reducing overhead while preserving correctness and security.

Accordingly, we employ specific cryptographic techniques for each group: HE for distance computation, batch PIR for point retrieval, and a mixed protocol combining SS and GC for top-k selection. Table 2 compares the approaches employed by Panther and SANNS for each category. Notably, our optimized designs outperform the state-of-the-art approach SANNS [11] across all categories. We begin by describing the point retrieval subroutine and elaborate on each subroutine and its respective optimizations in the following sections.

5 Point Retrievals via Batch PIR

In this section, we demonstrate how to efficiently instantiate secure point retrieval (Step 4-5 in Fig. 1) using PIR, which contributes the most to our efficiency gains.

5.1 Choosing a Shuffle-friendly PIR Protocol

From a practical perspective, our goal is to ensure that the shuffle operation does not significantly impact performance. Unfortunately, for efficient PIR protocols with preprocessing such as SimplePIR [26] and PIANO [64], they need a considerable amount of offline hints stored on the client side, which need to be updated accordingly with any database change. 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 [33] and BatchPIR [42], 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], OnionPIR [41], and Spiral [40]. 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 costs. 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 3, and discuss necessary customizations in the subsequent sections.

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 provides 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 many PIR protocols consider the data as a string and ignore its own structures. Separating each element is helpful for additive secret sharing, as will be shown in Section 5.2. Within a single plaintext, we can encode m points with d dimensions. Because $m \times d$ is large enough in our scenarios, this encoding method efficiently utilizes available space. For larger datasets, if there are more points in each cluster, we can encode them into multiple plaintext polynomials.

5.2 Converting PIR Results to Secret Shares

Traditional PIR schemes assume that the client can access the results in clear, while we require the results to be private to both parties. One may think it is straightforward for the server to add random masks on the PIR results, so that the results would be in secret-shared form. However, SealPIR operates in a two-dimensional manner and produces doubly-encrypted ciphertext $[\![[\![\vec{v}]\!]_l^H]\!]_l^H$ as its output. This restricts us from directly applying H2A [27] on such ciphertext, due to the one-bit error that H2A might introduce to the first-layer ciphertext $[\![\vec{v}]\!]_l^H$ that is segmented into several polynomial components. As a consequence, $[\![\vec{v}]\!]_l^H$ would be destroyed and fail to decrypt correctly when a one-bit error takes place in the middle of the assembled ciphertext.

The problem stems from the practice of parameter choices during the usage of SealPIR. Let (p, q) and (p', q') denote the plaintext and ciphertext moduli of the first and second dimensions, respectively.

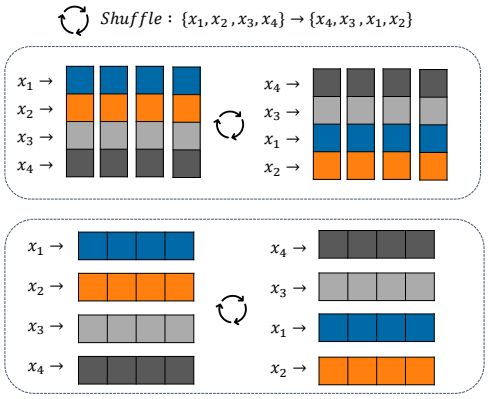


Figure 2: Different encoding methods. Encoding items into a single polynomial (up) requires re-encoding during each shuffle, while encoding into separate polynomials (down) only needs an index map $\{4, 3, 1, 2\}$ for efficient shuffling.

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 $\{q_1, \cdots, q_n\}$. In this setting, a first-layer ciphertext polynomial can be fully accommodated in a plaintext polynomial. Although H2A would still introduce a one-bit error into the first-layer ciphertext, it is absorbed into the low-end noise. We name the protocol Π_{PIR2A} (PIR to arithmetic secret shares) and describe its details in Figure 3.

In this way, we successfully instantiate $\mathcal{F}_{\text{BatchPIR}}$ using Π_{PIR2A}, avoiding the expensive DORAM in the point retrieval step. The security proof could be found in Appendix F.

6 Optimized Top-k Selection

We use approximate top-k to select the nearest clusters ($\mathcal{F}_{\text{ApproxTopK}}$ in Fig. 1) and exact top-k to select the nearest points ($\mathcal{F}_{\text{ExactTopK}}$ in Fig. 1). Both functions take a secret-shared database as input and produce k secret-shared outputs. In the following subsections, we describe how to efficiently instantiate these two functionalities.

6.1 Approximate Top-k with Mixed Primitives

SANNS proposed an effective approximate top-k selection method called ApproxTopK described in Algorithm 1. The idea is to randomly partition the input items into k' ($k' > k$) bins, compute the minimum within each bin, and perform an exact top-k selection from the k' minimum items.

ApproxTopK reduces the number of comparison operations from $O(nk)$ to $O(n + k'k)$, at the cost of possible errors. For example, if more than one of the top-k candidates is assigned into the same bin, only one of them will be selected. Fortunately, SANNS has proved that such chances are slim and that ApproxTopK satisfies the accuracy requirement in ANNS.

However, such $O(n + k'k)$ comparisons are still expensive in MPC. There are generally two kinds of MPC primitives to facilitate

Public Parameters:

- First dimension: $\text{HE.pp}_1 = \{N, q, p\}$, where $p = 2^t$ and q is a product of $\{q_1, \cdots, q_n\}$, dimension size n_1.
- Second dimension: $\text{HE.pp}_2 = \{N', q', p'\}$, let $p' = q_1$ and q' is a product of $\{q'_1, \cdots, q'_n\}$, dimension size n_2.
- Item size in database and encoded method.

Input: $\mathcal{C}$ inputs id and $\mathcal{S}$ inputs the encoded db.
▷ For simplicity, we assume db has $n{=}n_1 \cdot n_2$ plaintexts $\hat{pt}$, with each item $\vec{x}_i$ encoded in a separate $\hat{pt}_i \in \mathbb{A}_{N,p}$.
Output: $\mathcal{C}$ gets $[\![\vec{x}_{id}]\!]_c$ and $\mathcal{S}$ gets $[\![\vec{x}_{id}]\!]_s$.

1: $\mathcal{C}$ converts id into (id_1, id_2).
2: $\mathcal{C}$ generates queries $\text{ct}^1 \leftarrow \text{RLWE}^{N,q,p}_{\text{sk}}(x^{id_1})$ and to $\text{ct}^2 \leftarrow \text{RLWE}^{N',q',p'}_{\text{sk}'}(x^{id_2})$, and sends $(\text{ct}^1, \text{ct}^2)$ to $\mathcal{S}$.
3: $\mathcal{S}$ oblivious expands the query ciphertexts $(\text{ct}^1, \text{ct}^2)$:
$\langle \text{ct}^1_1, ..., \text{ct}^1_{n_1} \rangle \leftarrow \text{Expand}(\text{ct}^1)$
$\langle \text{ct}^2_1, ..., \text{ct}^2_{n_2} \rangle \leftarrow \text{Expand}(\text{ct}^2)$ ▷ Refer to §7.1 for details.
4: $\mathcal{S}$ computes first dimension result $\langle \text{s}_1, ..., \text{s}_{n_2} \rangle$, where $\forall i \in [n_2], \text{s}_i \leftarrow \sum_{j=1}^{n_1}(\text{ct}^1_j \cdot \hat{pt}_{i \cdot n_1 + j})$.
5: $\mathcal{S}$ adds a fresh $\text{RLWE}^{N,q,p}_{\text{pk}}(0)$ to each component of the first dimension result, then gets $\langle \text{s}'_1, ..., \text{s}'_{n_2} \rangle$.
6: $\mathcal{S}$ switches the modulus of the ciphertexts $\langle \text{s}'_1, ..., \text{s}'_{n_2} \rangle$ to q_1 and gets $\langle \text{s}''_1, ..., \text{s}''_{n_2} \rangle$, where $\text{s}''_i \in \mathbb{A}^2_{N,q_1}$.
7: $\mathcal{S}$ denotes $\langle \text{s}''_1, ..., \text{s}''_{n_2} \rangle$ as $\langle (\hat{b}_0, \hat{a}_0), ..., (\hat{b}_m, \hat{a}_m) \rangle$, where $\hat{a}_i, \hat{b}_i \in \mathbb{A}_{N,q_1}$.
8: $\mathcal{S}$ computes the second dimension result w_0, w_1, where $\text{w}_0 \leftarrow \sum_{i=1}^{n_2}(\text{ct}^2_i \cdot \hat{b}_i)$ and $\text{w}_1 \leftarrow \sum_{i=1}^{n_2}(\text{ct}^2_i \cdot \hat{a}_i)$.
9: $\mathcal{S}$ runs H2A [27] algorithm for only w_0 to mask the noise, then $\text{w}'_0 \leftarrow \text{w}_0 \boxplus \text{RLWE}^{N',q',p'}_{\text{pk}}(0) \boxplus (\hat{r}, 0)$, where $\hat{r} \in_R \mathbb{A}_{N',q'}$.
10: $\mathcal{S}$ switches the modulus of w'_0, w_1 to q'_1 and sends them back to $\mathcal{C}$, while holding $\hat{r}$.
11: $\mathcal{C}$ decrypts $\text{w}'_0, \text{w}_1$to obtain $\hat{u}, \hat{a}$. Then it treats the $(\hat{u}, \hat{a})$ as a ciphertext and decrypts it to get $[\![\vec{x}_{id}]\!]_c$.
12: $\mathcal{S}$ computes $\lfloor \lfloor -\hat{r} \cdot \frac{q_1}{q'_1} \rceil \cdot \frac{p}{q_1} \rceil$ to obtain $[\![\vec{x}_{id}]\!]_s$.

Figure 3: Protocol of PIR to secret shares Π_{PIR2A}.

Algorithm 1 ApproxTopK: Approximate top-k selection

Input: $(x_1, ID_1), \ldots, (x_n, ID_n), k, k'$
Output: $\langle id_1, ..., id_k \rangle$

function ApproxTopK($\vec{v}, n, k$)
  for $i \leftarrow 1 \ldots k'$ **do**
    $b_n \leftarrow n/k'$
    $(v_i, \widetilde{ID}_i) \leftarrow \text{Min}(\{(x_{(i \cdot b_n + j)}, ID_{(i \cdot b_n + j)})\}_{j=1}^{b_n})$
  end for
  return ExactTopK($(v_1, \widetilde{ID}_1), .., (v_{k'}, \widetilde{ID}_{k'}), k$)
end function

	Comm. cost	Rounds
GC Π_{cmp}[47]	$4\lambda\ell$ bits	$O(1)$
SS Π_{cmp} [27]	$< 11\ell$ bits	$O(\log \ell)$
GC Naive Top-k	$O(nk)\Pi_{cmp}$	$O(1)$
GC Top-k Network	$O(n(\log k)^2)\Pi_{cmp}$	$O(1)$
SANNS ApproxTopK	$6\lambda\ell(k'k+n)$ bits	$O(1)$
Panther ApproxTopK	$3k'(\log^2 k + 2)\cdot \lambda\ell + 13n\ell$ bits	$O(\log \ell \cdot \log(n/k'))$

Table 4: Communication cost of different protocols. λ is the security parameter (usually $\lambda \geq 128$). Π_{cmp} represents the comparison protocol of two ℓ-bit private integers.

secure comparisons: garbled circuit (GC) and secret sharing (SS). The performance metrics of state-of-the-art comparison protocols in two-party computation are presented in Table 4. State-of-the-art SS-based protocols utilize oblivious transfer (OT) to perform comparisons. Thanks to recent advancements in silent OT[8], it enjoys a significant reduction in communication costs. However, it requires multiple communication rounds, which can become a bottleneck in high-latency environments, especially for hierarchical comparison circuits such as top-k. In contrast, GC-based protocols incur higher communication costs but maintain a constant number of communication rounds, irrespective of circuit complexity.

By carefully studying the secure instantiations of ApproxTopK, we found that indeed the exact top-k part (Line 6, Algorithm 1) performs best with GC, since SS-based exact top-k requires excessive communication rounds. However, computing the minimum (Line 4, Algorithm 1) is better suited for SS. This is because the operations in different bins can be parallelized, sharing the same round complexity. We employ the SS-based comparison protocol Π_{cmp} [27] to compute the minimum within each bin. This approach requires only $O(\log \ell \cdot \log(n/k'))$ communication rounds and incurs a communication cost of $11n\ell$ bits, which is significantly better under various network conditions.

6.2 Exact Top-k with a Better Selection Network

SANNS employs a naive algorithm for exact top-k, which computes the minimum k times to obtain the exact top-k results from n input values, leading to $O(nk)$ compare-and-swap operations.[5] This approach is sub-optimal for both GC-based and SS-based protocols.

To improve the performance of exact top-k, we build it on the bitonic top-k selection network proposed by [49] with two improvements:

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

Figure 4 illustrates our construction with an example, with details described in Appendix C. We give a brief description of how we

[5]To avoid notational ambiguity in the Algorithm 1(ApproxTopK), when invoking ExactTopK, the number of input values is interpreted as k'.

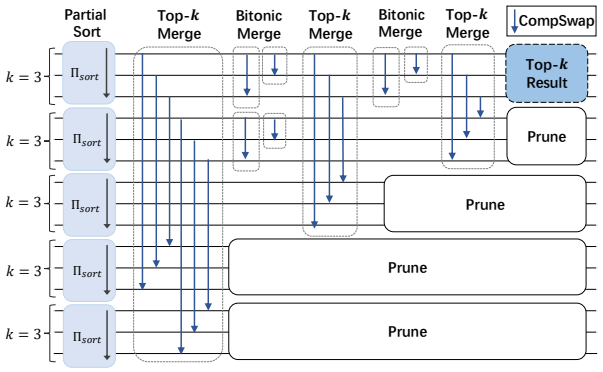


Figure 4: Top-k selection network demo ($k = 3, n = 15$)

Sort	CompSwap
Bitonic Sort	$n(\log n)(\log n + 1)/4$
Odd-even Merge Sort	$n(\log n)(\log n - 1)/4 + n - 1$
Reduction	$n((\log n)/2 - 1) + 1$

Table 5: 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.

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 $i \in [2k]$. 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 $\langle s_1, \cdots, s_k\rangle$ 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 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 $2k$, 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 Sort in partial sort. Although it has the same asymptotic complexity as bitonic sort, odd-even merge sort produces less compare and swap operations, which we detail in Table 5. The full algorithm is given in Appendix C. The functionality and security proof of the approximated top-k sub-protocol is given in Appendix F. Our optimized top-k selection network has much better asymptotic complexity $O(n \log k \log k)$ compared to previous alternatives (Table 4), with also higher concrete efficiency as we will show in the experiments.

By combining the techniques in Section 6.1 and 6.2, we successfully reduce the overall communication cost of secure ApproxTopK from roughly $6\lambda\ell(k'k+n)$ bits to $3k'(\log^2 k + 2)\cdot\lambda\ell + 13n\ell$ bits.

Algorithm 2 Plaintext modulus $p = 2^k$ expand algorithm.

Input: $\mathsf{RLWE}_{\mathsf{sk}}^{N,q,p}(x^i), n$ ▷ query
Output: $\langle \mathsf{ct}_1, \mathsf{ct}_2, ..., \mathsf{ct}_n \rangle$
▷ where the i^{th} entry is an encryption of 1 and all other entries are encryption 0.

1: **function** Expand($\mathsf{RLWE}_{\mathsf{sk}}^{N,q,p}(x^i)$, n)
2: find smallest $m = 2^l$ such that $m \geq n$
3: $inverse \leftarrow m^{-1} \pmod q$
4: $query\prime \leftarrow query \cdot inverse$
5: **return** $output \leftarrow$ NaiveExpand($query'$, n)
▷ Appendix A details NaiveExpand[2].
6: **end function**

7 Further Optimizations

In this section we present more optimizations that further enhance the efficiency of Panther.

7.1 Supporting $\mathbb{Z}_{2^k}$ Plaintext Domain in PIR

Common PIR schemes are based on homomorphic encryption, thus their plaintext domain naturally works under a prime field. However, we want to increase the efficiency by representing the plaintext (the coordinates and distances) within the ring $\mathbb{Z}_{2^k}$. This choice is motivated by the observation that MPC protocols operate more efficiently in the ring $\mathbb{Z}_{2^k}$ than in the prime field $\mathbb{Z}_p$ (Table 2, [47]), since protocols under $\mathbb{Z}_{2^k}$ are easier to design, and modulo reduction in $\mathbb{Z}_{2^k}$ is nearly free in CPU. The SealPIR protocol [2] inherently supports coefficient encoding without the constraint of having a prime plaintext modulus.

However, the plaintext modulus $\mathbb{Z}_{2^k}$ is incompatible with the inverse operation in the expansion step (Figure 3, [2]) of SealPIR, which necessitates the plaintext modulus be an odd number. Essentially, let the coefficient of the plaintext polynomial after expansion be m. To recover the value to 1, the result must be multiplied by $m^{-1} \pmod p$. If we set the plaintext modulus $p = 2^k$, we cannot find $m^{-1} \pmod p$ successfully.

Fortunately, we have discovered that multiplying the entire ciphertext by $m^{-1} \pmod q$ can also be effective. This approach implicitly requires that q be an odd modulus, a condition typically satisfied in RLWE-based homomorphic encryption, where the ciphertext modulus q is chosen such that $q = 1 \pmod{2N}$. We outline this idea in Algorithm 2 and present our findings below.

Lemma 1. *Algorithm 2 securely expands the input* $\mathsf{RLWE}_{\mathsf{sk}}^{N,q,p}(x^i)$ *to a vector of ciphertexts* $\langle \mathsf{ct}_1, \mathsf{ct}_2, ..., \mathsf{ct}_n \rangle$*, where the* i^{th} *entry is an encryption of 1 and all other entries are encryption 0.*

The algorithm just does homomorphic operations on the ciphertext, so the privacy is guaranteed by the BFV scheme itself, and we only show the correctness argument of Lemma 1 below.

Proof. The query $\mathsf{RLWE}_{\mathsf{sk}}^{N,q,p}(x^i) = (\lfloor \frac{q}{p} x^i \rceil + \hat{e} - \hat{a} \cdot \mathsf{sk}, \hat{a})$. In line 3, we multiply the constant $m^{-1} \pmod q$, resulting in $(\lfloor \frac{q}{p} m^{-1} x^i \rceil + m^{-1}\hat{e} - m^{-1}\hat{a} \cdot \mathsf{sk}, m^{-1}\hat{a})$. If the algorithm NaiveExpand is applied to the ciphertext $query'$, the constant factor m is naturally eliminated. Then the phase of i-th ciphertext will be approximately equal to $m \cdot m^{-1} \pmod q = 1$. A similar technique has been used in previous work (Section 3.4, [12]).

We note that the multiply operation will influence the input noise, but the result can still be decrypted successfully. Let v_{in} be the input noise in $query'$ and v_{out} be the output noise of NaiveExpand. B_{sub} is a constant additive term, which depends only on the HE parameters. After the l-th iteration we have that

$$v_{out} = 2^l v_{in} + 2(2^l - 1)B_{sub} \pmod q [2]$$

Let $\hat{e}$ be the noise in $\mathsf{RLWE}_{\mathsf{sk}}^{N,q,p}(x^i)$. After multiplying the query ciphertext with $m^{-1} \pmod q$, the noise $v_{in} = m^{-1}\hat{e} \pmod q$. After calling NaiveExpand, the noise in v_{out} is small enough because the factor m^{-1} can be eliminated.

$$\begin{aligned} v_{out} &= 2^l m^{-1}\hat{e} + 2(2^l - 1)B_{sub} \\ &= m m^{-1}\hat{e} + 2(2^l - 1)B_{sub} \\ &= \hat{e} + 2(2^l - 1)B_{sub} \pmod q \end{aligned}$$

□

7.2 Faster and More Accurate PIR-to-shares

Fixing last level error. In the protocol Π_{PIR2A} (Fig. 3), although we solve the first level one-bit error and successfully re-share the retrieved points between the server and client, there remains a last level one-bit error that can affect accuracy. This error appears in the point value, not in the distance value, and should not be ignored. To address this, we propose a protocol to fix the one-bit error. The key idea is to pad the lower bits of the value with $0b01$. If a one-bit error occurs, the lower bits will become $0b00$ or $0b10$ (secret-shared), neither of which will affect the higher bits. We can then apply a truncate and reduce protocol in [46] to remove the lower two bits securely and obtain the correct value.

Efficient bitwidth conversion of PIR results. After retrieving multiple points by Π_{PIR2A}, the data are secret-shared under plaintext modulus p. Sometimes, it's necessary to represent the data such as $||\vec{q}||_2^2$ in a larger ring without increasing the HE parameters p. We hope to adjust the secret-shared data bitwidth. If we know the most significant bit (MSB) of the shared data, the extension protocol can be much cheaper. Note that we can make this assumption because all data have been quantized to ℓ bits unsigned int, we can set the plaintext modulus $\geq \ell + 1$ bits to make the response of PIR shared with MSB = 0.

The extension protocol with a known MSB in [27, 36] requires two rounds of communication and $O(\ell' - \ell)$ bits. One round is to compute a wrap bit and another round is to invoke a boolean to arithmetic (B2A) protocol. We find it can be optimized to require only one round. To extend an ℓ-bit shared $[\![x]\!]$, given the MSB of x is 0, let $\mathcal{S}$ and C locally split each share into two parts $[\![x]\!]_s = msb([\![x]\!]_s)||[\![x]\!]_s^{\ell-1}$ and $[\![x]\!]_c = msb([\![x]\!]_c)||[\![x]\!]_c^{\ell-1}$. We define w' as whether the sum of $[\![x]\!]_s^{\ell-1}$ and $[\![x]\!]_c^{\ell-1}$ wrap around $2^{\ell-1}$. When MSB is zero, we have $w' = msb([\![x]\!]_s) \oplus msb([\![x]\!]_c)$, which means we have boolean share $[\![w']\!]^B$ for free. Then the extended result $x = [\![x]\!]_s^{\ell-1} + [\![x]\!]_c^{\ell-1} - w' * 2^{\ell-1} \pmod{2^{\ell'}}$ and $w' * 2^{\ell-1}$ can be computed by a B2A protocol [47] in only one round.

7.3 More Compact HE Ciphertext in Distance Computation

We provide a brief overview of the method for distance computation using leveled HE. Consider the Euclidean distance between d-dimensional points, which is given by: $||\vec{u}-\vec{v}||_2^2 = ||\vec{u}||_2^2 + ||\vec{v}||_2^2 - 2\langle\vec{u},\vec{v}\rangle$. We analyze it in two separate cases, which incorporate different optimization steps.

7.3.1 Case 1: $\vec{v}$ is private (Π_{Panther} step 2). Suppose C holds $\vec{u}$ and $\mathcal{S}$ holds a set of points $\{\vec{v}_j\}_{j\in[N]}$. To compute the distance securely, C encodes each coordinate of $\vec{u}$ as a constant polynomial $\hat{f}_i = \vec{u}[i]$ and sends the encryption of these d polynomials to $\mathcal{S}$. $\mathcal{S}$ then encodes its points into polynomials $\hat{g}_i = \sum_j \vec{v}_j[i]x^{j-1}$, for $i \in [d]$. The inner products $\langle\vec{u},\vec{v}_j\rangle$ are computed as the coefficients of the polynomial $\hat{h} = \sum_i \hat{f}_i\hat{g}_i$.

7.3.2 Case 2: $\vec{v}$ is secret-shared (Π_{Panther} step 6). Suppose C holds $\vec{u}$, but $\{\vec{v}_j\}_{j\in[N]}$ are a set of secret-shared points. Let ℓ_0, ℓ_1, and ℓ_2 denote the bit length of point coordinate, its secret share, and distance's secret share, respectively. Note that $\ell_2 = 2\ell_0 + \lceil\log d\rceil$ for d-dimensional points. C and $\mathcal{S}$ collaboratively compute $\langle\vec{u},[\![\vec{v}_j]\!]_s\rangle$ in the same way as in Case 1, while C locally computes $\langle\vec{u},[\![\vec{v}_j]\!]_c\rangle$. The final distance can be calculated as:

$$||\vec{u}||_2^2 + [\![||\vec{v}_j||_2^2]\!] - 2\langle\vec{u},[\![\vec{v}_j]\!]_s\rangle - 2\langle\vec{u},[\![\vec{v}_j]\!]_c\rangle \mod 2^{\ell_2}, \quad (1)$$

where $[\![||\vec{v}_j||_2^2]\!]$ is required to be pre-shared between the two parties. However, $\vec{v}_j = [\![\vec{v}_j]\!]_s + [\![\vec{v}_j]\!]_c$ only holds modulo 2^{ℓ_1}, leading to a potential error due to the wrap bits $\vec{w}$: $\vec{v}_j = [\![\vec{v}_j]\!]_s + [\![\vec{v}_j]\!]_c - \vec{w}\cdot 2^{\ell_1}$. One straightforward solution is to set $\ell_1 = \ell_2$, which leads to suboptimal performance because random shares $[\![\vec{v}_j]\!]_s$ in the larger ring $\mathbb{Z}_{2^{\ell_2}}$ would accelerate noise growth during ciphertext-plaintext multiplication $\langle\vec{u},[\![\vec{v}_j]\!]_s\rangle$, necessitating larger homomorphic parameters. Instead of extending the bit-width of the shared points, we opt to use OT to compute $\langle\vec{w},\vec{u}\rangle$ for result correction, where $\vec{w}$ can be obtained with the technique in 7.2.

7.3.3 Further optimizations. To share the output distance between the two parties for subsequent computations, we need to reshare $\hat{h}$. A simple masking of $\hat{h}$ by sending the encryption of $\hat{h}-\hat{r}$ back to C with a randomly sampled $\hat{r} \in_R \mathbb{A}_{N,p}$ is not sufficient to *securely* produce additive secret sharing, because the noise contained in the ciphertext reveals information about v_i, the private information of $\mathcal{S}$. In the SANNS framework [11], in order to eliminate such leakage, they rely on the expensive noise flooding technique [19] to hide such information using larger noise. However, this enforces them to use larger BFV parameters to accommodate noise, i.e., $N = 2^{13}$ and 180-bit q, resulting in suboptimal performance[6].

To optimize performance, we leverage the H2A technique introduced in [27], which essentially combines masking and noise flooding in a single step by sampling a random polynomial $\hat{r} \in_R \mathbb{A}_{N,q}$ in the ciphertext domain and sending back $[\![\vec{h}]\!]_1^H \boxplus \hat{r}$ to C. $\mathcal{S}$ locally constructs its own share from $\hat{r}$ as shown in Section 3.1.3. It is not difficult to argue the information hiding property of this approach since $\hat{r}$ completely randomizes the ciphertext, including the noise.

[6]Theoretically, they could adopt our small HE parameters for efficient distance computation and use the techniques in [14] to switch to larger HE parameters for noise flooding, but this would incur costly bootstrapping.

	N	p	$q : \{q_1, ..., q_n\}$
Distance Computation	2048	2^{24}	{54} bits
PIR HE.pp$_1$	4096	2^{13}	{24,36,37} bits
PIR HE.pp$_2$	4096	$\log p \approx 24$	{36,36,37} bits

Table 6: BFV parameters HE.pp. In the RNS variant of BFV, q is decomposed into smaller co-prime components $q_1, \ldots, q_n$.

This approach eliminates the need for large BFV parameters to accommodate the noise, introducing only a probabilistic one-bit error (Appendix C,[27]). This has minimal impact on our ANNS scenario, as confirmed by our experiments, since low-order bits are typically truncated post-multiplication. Consequently, this optimization enables us to use a small parameter set with $N = 2^{11}$ and a 54-bit q under the same level of RLWE security. Details can be found in Table 6.

Halving the communication cost in H2A. We discovered that optimizing the H2A protocol can effectively reduce the communication cost during the response phase by nearly 50%. At the end of H2A, $\mathcal{S}$ will hold $-\hat{r}$ and C will hold $(\hat{b'}+\hat{r}, \hat{a'})$. Here, $(\hat{b'}, \hat{a'})$ represents the response $(\hat{b}, \hat{a})$ added by a fresh encryption of zero, and $\hat{r} \in_R \mathbb{A}_{N,q}$. We suggest that C holds $(-\hat{r}, \hat{a'})$ and $\mathcal{S}$ holds $(\hat{b'}+\hat{r})$. It does not affect correctness and privacy. Instead of sending back $(-\hat{r}, \hat{a'})$, $\mathcal{S}$ can send back only $\hat{a'}$ and a random seed, allowing $\hat{r}$ to be locally sampled. This modification can reduce the communication cost by half during the response phase.

8 Evaluation

8.1 Implementation and Environment

We have fully implemented Panther in C++ using the SecretFlow[7] and EMP-toolkit [58] 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 SecretFlow PSI library to construct the PIR building block, and the emp-sh2pc framework[8] to implement the top-k selection network using garbled circuit protocols. These components were integrated to create a comprehensive end-to-end implementation, which we subsequently evaluated in terms of performance. Our code is available at https://github.com/AntCPLab/OpenPanther.git.

The experiments presented in this paper were conducted on cloud instances equipped with 64 vCPU, operating at 2.80 GHz and 256 GB of RAM. Both the client and the server are running under 32 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]https://github.com/secretflow
[8]https://github.com/emp-toolkit/emp-sh2pc

8.2 Other Settings

BFV Parameters. 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 6 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 ($n = 1,000,000$, $d = 96$), Deep1B-10M ($n = 10,000,000$, $d = 96$) 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 at the approximate top-k) as those used in SANNS [11] to assess 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.

8.3 Microbenchmarks

In this section, we test some of our sub-protocols that could be of general interest. The results are obtained using a single thread unless specified.

8.3.1 PIR to secret shares. In Table 7 we demonstrate the performance of our PIR-to-share protocol Π_{PIR2A}, which extends the SealPIR family—originally limited to private queries—to support the broader MPC scenarios requiring private "query-then-compute".

We evaluate the protocol for processing a single query on a database containing 2048 items, where each item, approximately 50 Kb in size (resembling a cluster in ANNS), is encoded into the coefficients of a 4096-degree polynomial. The "naive" method is to add a random plaintext $\hat{r} \in_R \mathbb{A}_{N,p}$ to the PIR result, and then use noise flooding to eliminate the database information within the noise. Our Π_{PIR2A} is equipped with several optimizations described in Sections 5.2 and 7.2.

As demonstrated in Table 7, our PIR-to-shares approach significantly outperforms the "naive" one, achieving both faster execution and a smaller response size.

Impact of varying database size and item length. Our optimization does not change the database processing procedure, and focuses on using smaller HE parameters for the resulting ciphertexts. Therefore, the performance gains remain constant and independent of database size. For items of varying lengths, these gains scale linearly with the number of ciphertexts returned.

8.3.2 Distance Computation. We compare the performance of our distance computation method using H2A against that of SANNS using noise flooding in Table 8. We compute the distance of one point versus 100,000 points, each having a dimensionality of 128, quantized to 8 bits. For noise flooding, we use $N = 8192$, $p = 2^{24}$ and $q \approx 180$ bits. Since H2A enables us to use smaller HE parameters, our method significantly reduces communication costs by 9.3× and running time by $2.9 - 4.5\times$.

PIR to secret shares ($n = 2048$)	LAN	WAN	Comm.
Naive	278 ms	1011 ms	859 KB
Our Π_{PIR2A}	102 ms	468 ms	227 KB

Table 7: Performance comparison of PIR to secret shares.

Distance Computation ($n = 100,000, d = 128$)	LAN	WAN	Comm.
Noise Flooding [11]	763 ms	2115 ms	41.7 MB
Ours	265 ms	474 ms	4.50 MB

Table 8: Performance comparison of distance computation.

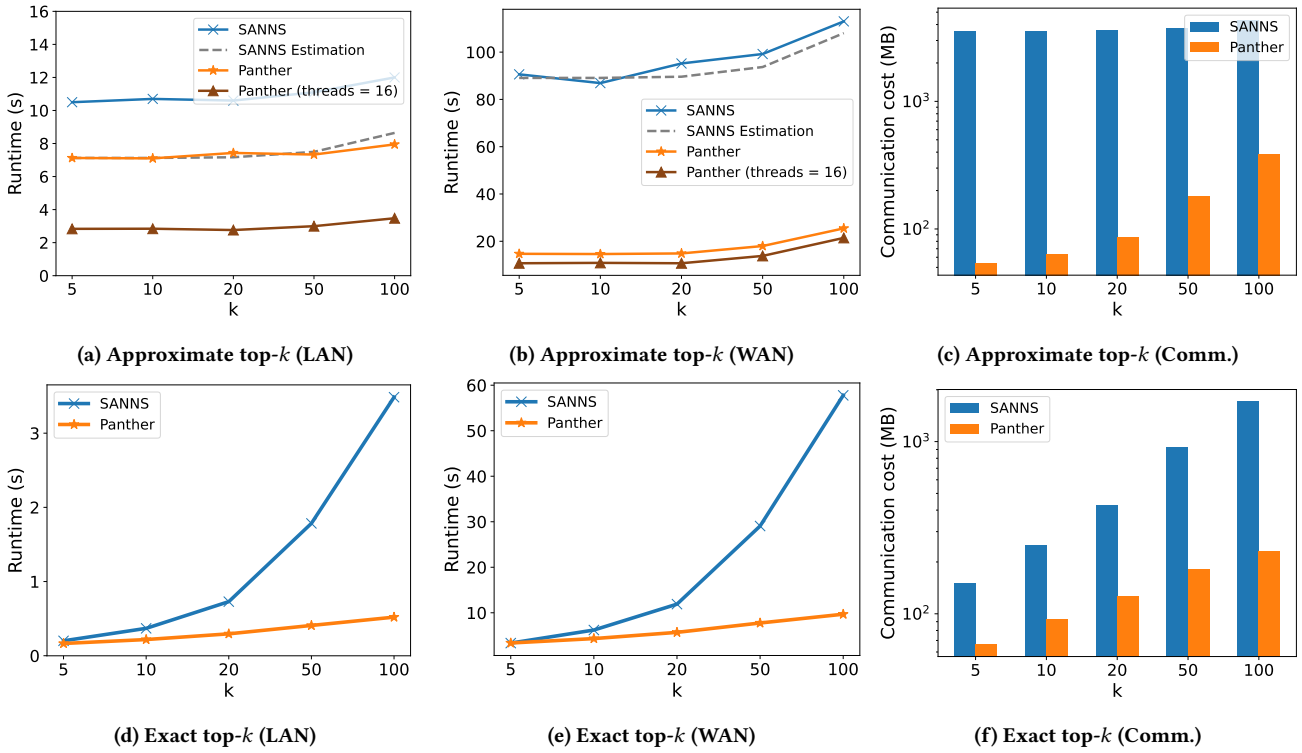
8.3.3 Top-k selection. We test our two improvements: The mix-primitive approximate top-k described in Section 6.1, and the better exact top-k described in Section 6.2. Figure 5 shows the evaluation results. The blue line is from the paper of SANNS, and the gray line is calculated by communication/bandwidth from the paper of SANNS. We test $n = 1,000,000$ 24-bit shared integers for mixed primitives approximate top-k. The approximate top-k algorithm can return an expected number of $(1-\delta)\cdot k$ accurate entries when the number of bins is set to $k' = k/\delta$. In our experiment, we use $\delta = 0.01$ and consider values of k from the set $\{5, 10, 20, 50, 100\}$. The numbers in Figure 5 (a),(b),(c) indicate that even in the single-threaded context, our protocol demonstrates a performance improvement of up to 5×. When using more threads, our protocol performance can be much better, while their GC-based algorithm cannot break through the bandwidth limit.

We test $n = 10,000$ 24-bit shared integers for exact top-k and assess $k \in \{5, 10, 20, 50, 100\}$. These parameters are closer to the actual usage in clustering-based ANNS. We reimplement the exact top-k algorithm in SANNS using emp-sh2pc to get data under these parameters. As shown in Figure 5 (d),(e),(f), we achieve an improvement of 2.3× to 7.5×, the performance gap becomes more significant when k increases.

8.4 End-to-End Evaluation

We present a comparison of end-to-end secure ANNS in Table 9. For SANNS, they just said their bandwidth ranges "from 500 MB/s to 7 GB/s" without indicating the actual bandwidth for each experiment, so we report their communication cost as outlined in their paper, and estimate their runtime using the communication cost divided by bandwidth. Note that this completely neglect their computational cost.

As shown, our approach greatly enhances the performance of each component. Our distance computation (the "Distances" column) appears to be slower than that of SANNS under LAN because we neglect their computational time. However, we are still more efficient under WAN due to smaller HE parameters. For the top-k



(a) Approximate top-k (LAN) (b) Approximate top-k (WAN) (c) Approximate top-k (Comm.)

(d) Exact top-k (LAN) (e) Exact top-k (WAN) (f) Exact top-k (Comm.)

Figure 5: Running time and communication costs of top-k selection subroutines.

Dataset		Distances		Top-k		Points Retrieval		Total		
		SANNS	Panther	SANNS	Panther	SANNS	Panther	SANNS	Panther	
S.1M	LAN (s)	0.11	0.27	1.29	0.31	2.17	0.90	3.62	1.49	2.4 ×
	WAN (s)	1.41	0.85	16.1	6.70	27.1	1.68	45.3	9.32	4.9 ×
	Comm.	56.7 MB	4.68 MB	645 MB	68.6 MB	1.06 GB	19.6 MB	1.77 GB	92.9 MB	↓ 95 %
D.1M	LAN (s)	0.09	0.20	1.24	0.33	1.84	0.96	3.23	1.50	2.2 ×
	WAN (s)	1.10	0.46	15.5	6.50	23.0	1.74	40.4	8.71	4.6 ×
	Comm.	44.1 MB	3.70 MB	620 MB	71.2 MB	920 MB	19.7 MB	1.58 GB	94.6 MB	↓ 94 %
D.10M	LAN (s)	0.12	0.67	4.81	0.77	6.39	2.45	11.3	3.89	2.9 ×
	WAN (s)	1.48	1.12	60.2	13.1	79.9	4.10	142	18.3	7.8 ×
	Comm.	59.4 MB	6.03 MB	2.35 GB	230 MB	3.12 GB	47.5 MB	5.53 GB	284 MB	↓ 95 %
Amzn	LAN (s)	0.05	0.10	1.06	0.33	1.23	0.82	2.29	1.25	1.8 ×
	WAN (s)	0.61	0.30	13.2	6.91	15.4	1.61	28.7	8.83	3.3 ×
	Comm.	24.4 MB	2.15 MB	528 MB	78.7 MB	617 MB	18.1 MB	1.12 GB	98.9 MB	↓ 91 %

Table 9: 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.**

component, we count in the full approximate top-k performance. It could be seen that our top-k 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 communication costs by up to 99%.

In summary, our scheme consistently outperforms SANNS in various datasets. We achieve a reduction in communication costs of up to 95%, while improving search times by up to 7.8×. If we take their computational cost into account, our improvements will be even more significant.

Comparison with a two-server work. Our performance is even comparable with [48] which works under a two-server setting. They answer a query on Deep1B-10M in 6.13 seconds, which is 3.0× faster than Panther under WAN but 1.6× slower than Panther under LAN. Note that their two servers reside in the same region of AWS regardless of WAN or LAN.

Dataset	Panther without H2A error	Panther
SIFT-1M	88.85%	88.85%
Deep1B-1M	89.16%	89.16%

Table 10: Accuracy comparison between Panther **without H2A errors (to simulate [11]) and** Panther**.**

8.5 Accuracy

Since Panther and SANNS have the same underlying plaintext routine, and share the same set of hyperparameters such as plaintext bitwidth, number of bins in approximate top-k, and shape of clusters, we expect that the accuracy of Panther will be the same as SANNS, which is determined by the plaintext clustering algorithm. SANNS achieves 10-NN accuracy of 0.9 (9 of 10 points are correct on average, Section 5.3 of [11]).

There are only two points that might lead to accuracy difference between Panther and SANNS. The first is the possibility of a one-bit error in the H2A step of distance computation (Section 7.3). Although the one-bit error occurs in the least significant bit and is unlikely to affect the accuracy, we want to experimentally show the results. However, SANNS did not provide enough details to reproduce the same experiment result in their paper; most importantly, we do not have the plaintext clustering algorithm they used. As a result, we choose the k-means clustering algorithm in [18] and use Panther with noise flooding instead of H2A (so that there are no one-bit errors) to simulate SANNS. Table 10 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 hyperparameters, either aligned or not aligned to the settings described in SANNS, the accuracy is still slightly below 0.9, which may because the plaintext clustering algorithm in [18] is not optimized for the ANNS task.

The second difference is that the batch queries in PIR naturally return additional items due to the cuckoo hash method used. Specifically, Panther returns 50% more points in the point retrieval step. These extra points will be fed into the top-k step so the size of the final output does not change. It is likely that feeding more candidate clusters to the top-k step could improve 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.

9 Conclusion

In this paper, we propose an efficient private ANNS framework Panther. It can significantly improve performance than the previous single-server ANNS framework. We design novel protocols to retrieve points with efficient batch PIR instead of DORAM. 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. After evaluating the end-to-end implementation on datasets of different sizes, Panther is superior to the previous work in both communication cost and computation cost.

Acknowledgment

Work of Min Zhang was supported by National Key R&D Program of China (2022YFB4501500, 2022YFB4501503) and Ant Group. Work of Jian Liu was supported by National Key R&D Program of China (2023YFB2704000).

References

[1] Ishtiyaque Ahmad, Yuntian Yang, Divyakant Agrawal, Amr El Abbadi, and Trinabh Gupta. 2021. Addra: Metadata-private voice communication over fully untrusted infrastructure. In *15th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2021, July 14-16, 2021*, Angela Demke Brown and Jay R. Lorch (Eds.). USENIX Association. https://www.usenix.org/conference/osdi21/presentation/ahmad

[2] Sebastian Angel, Hao Chen, Kim Laine, and Srinath T. V. Setty. 2018. 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*. IEEE Computer Society, 962–979. doi:10.1109/SP.2018.00062

[3] Toshinori Araki, Jun Furukawa, Kazuma Ohara, Benny Pinkas, Hanan Rosemarin, and Hikaru Tsuchida. 2021. Secure Graph Analysis at Scale. In *CCS '21: 2021 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, Republic of Korea, November 15 - 19, 2021*, Yongdae Kim, Jong Kim, Giovanni Vigna, and Elaine Shi (Eds.). ACM, 610–629. doi:10.1145/3460120.3484560

[4] Gilad Asharov, Shai Halevi, Yehuda Lindell, and Tal Rabin. 2018. Privacy-Preserving Search of Similar Patients in Genomic Data. *Proc. Priv. Enhancing Technol.* 2018, 4 (2018), 104–124. doi:10.1515/POPETS-2018-0034

[5] Hilal Asi, Fabian Boemer, Nicholas Genise, Muhammad Haris Mughees, Tabitha Ogilvie, Rehan Rishi, Guy N. Rothblum, Kunal Talwar, Karl Tarbe, Ruiyu Zhu, and Marco Zuliani. 2024. Scalable Private Search with Wally. arXiv:2406.06761 [cs.CR] https://arxiv.org/abs/2406.06761

[6] Martin Aumüller, Erik Bernhardsson, and Alexander Faithfull. 2020. ANN-Benchmarks: A benchmarking tool for approximate nearest neighbor algorithms. *Information Systems* 87 (2020), 101374. doi:10.1016/j.is.2019.02.006

[7] Arman Bhalla, Ivo Sluganovic, Klaudia Krawiecka, and Ivan Martinovic. 2021. MoveAR: Continuous biometric authentication for augmented reality headsets. In *Proceedings of the 7th acm on cyber-physical system security workshop*. 41–52.

[8] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, Peter Rindal, and Peter Scholl. 2019. Efficient two-round OT extension and silent non-interactive secure computation. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. 291–308.

[9] Rui Cai, Chao Zhang, Lei Zhang, and Wei-Ying Ma. 2007. Scalable music recommendation by search. In *Proceedings of the 15th International Conference on Multimedia 2007, Augsburg, Germany, September 24-29, 2007*, Rainer Lienhart, Anand R. Prasad, Alan Hanjalic, Sunghyun Choi, Brian P. Bailey, and Nicu Sebe (Eds.). ACM, 1065–1074. doi:10.1145/1291233.1291466

[10] Ran Canetti. 2000. Security and composition of multiparty cryptographic protocols. *Journal of CRYPTOLOGY* 13 (2000), 143–202.

[11] Hao Chen, Ilaria Chillotti, Yihe Dong, Oxana Poburinnaya, Ilya P. Razenshteyn, and M. Sadegh Riazi. 2020. SANNS: Scaling Up Secure Approximate k-Nearest Neighbors Search. In *29th USENIX Security Symposium, USENIX Security 2020, August 12-14, 2020*, Srdjan Capkun and Franziska Roesner (Eds.). USENIX Association, 2111–2128. https://www.usenix.org/conference/usenixsecurity20/presentation/chen-hao

[12] Hao Chen, Wei Dai, Miran Kim, and Yongsoo Song. 2021. Efficient Homomorphic Conversion Between (Ring) LWE Ciphertexts. In *Applied Cryptography and Network Security - 19th International Conference, ACNS 2021, Kamakura, Japan, June 21-24, 2021, Proceedings, Part I (Lecture Notes in Computer Science, Vol. 12726)*, Kazue Sako and Nils Ole Tippenhauer (Eds.). Springer, 460–479. doi:10.1007/978-3-030-78372-3_18

[13] Ningning Cui, Xiaochun Yang, Bin Wang, Jianxin Li, and Guoren Wang. 2020. SVkNN: Efficient secure and verifiable k-nearest neighbor query on the cloud platform. In *2020 IEEE 36th International Conference on Data Engineering (ICDE)*. IEEE, 253–264.

[14] Morten Dahl, Daniel Demmler, Sarah El Kazdadi, Arthur Meyre, Jean-Baptiste Orfila, Dragos Rotaru, Nigel P Smart, Samuel Tap, and Michael Walter. 2023. Noah's Ark: Efficient Threshold-FHE Using Noise Flooding. In *Proceedings of the 11th Workshop on Encrypted Computing & Applied Homomorphic Cryptography*. 35–46.

[15] Ghada Dessouky, Farinaz Koushanfar, Ahmad-Reza Sadeghi, Thomas Schneider, Shaza Zeitouni, and Michael Zohner. 2017. 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. https://www.ndss-symposium.org/ndss2017/ndss-2017-programme/pushing-communication-barrier-secure-computation-using-lookup-tables/

[16] Naryttza N Diaz, Lutz Krause, Alexander Goesmann, Karsten Niehaus, and Tim W Nattkemper. 2009. TACOA–Taxonomic classification of environmental genomic

fragments using a kernelized nearest neighbor approach. *BMC bioinformatics* 10 (2009), 1–16.

[17] Jack Doerner and Abhi Shelat. 2017. Scaling ORAM for Secure Computation. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*, Bhavani Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu (Eds.). ACM, 523–535. doi:10.1145/3133956.3133967

[18] Matthijs Douze, Alexandr Guzhva, Chengqi Deng, Jeff Johnson, Gergely Szilvasy, Pierre-Emmanuel Mazaré, Maria Lomeli, Lucas Hosseini, and Hervé Jégou. 2024. The Faiss library. (2024). arXiv:2401.08281 [cs.LG]

[19] Léo Ducas and Damien Stehlé. 2016. 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*. Springer, 294–310.

[20] Daniel Escudero. 2022. An Introduction to Secret-Sharing-Based Secure Multiparty Computation. *IACR Cryptol. ePrint Arch.* (2022), 62. https://eprint.iacr.org/2022/062

[21] Junfeng Fan and Frederik Vercauteren. 2012. Somewhat practical fully homomorphic encryption. *Cryptology ePrint Archive* (2012).

[22] Cong Fu, Chao Xiang, Changxu Wang, and Deng Cai. 2019. Fast Approximate Nearest Neighbor Search With The Navigating Spreading-out Graph. *Proc. VLDB Endow.* 12, 5 (2019), 461–474. doi:10.14778/3303753.3303754

[23] Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, Qianyu Guo, Meng Wang, and Haofen Wang. 2023. Retrieval-Augmented Generation for Large Language Models: A Survey. *CoRR* abs/2312.10997 (2023). doi:10.48550/ARXIV.2312.10997 arXiv:2312.10997

[24] Changhee Hahn and Junbeom Hur. 2016. Efficient and privacy-preserving biometric identification in cloud. *ICT Express* 2, 3 (2016), 135–139. doi:10.1016/J.ICTE.2016.08.006

[25] Alexandra Henzinger, Emma Dauterman, Henry Corrigan-Gibbs, and Nickolai Zeldovich. 2023. Private Web Search with Tiptoe. In *Proceedings of the 29th Symposium on Operating Systems Principles, SOSP 2023, Koblenz, Germany, October 23-26, 2023*, Jason Flinn, Margo I. Seltzer, Peter Druschel, Antoine Kaufmann, and Jonathan Mace (Eds.). ACM, 396–416. doi:10.1145/3600006.3613134

[26] Alexandra Henzinger, Matthew M. Hong, Henry Corrigan-Gibbs, Sarah Meiklejohn, and Vinod Vaikuntanathan. 2023. One Server for the Price of Two: Simple and Fast Single-Server Private Information Retrieval. In *32nd USENIX Security Symposium, USENIX Security 2023, Anaheim, CA, USA, August 9-11, 2023*, Joseph A. Calandrino and Carmela Troncoso (Eds.). USENIX Association, 3889–3905. https://www.usenix.org/conference/usenixsecurity23/presentation/henzinger

[27] Zhicong Huang, Wen-jie Lu, Cheng Hong, and Jiansheng Ding. 2022. Cheetah: Lean and Fast Secure Two-Party Deep Neural Network Inference. *USENIX Security* (2022).

[28] Hervé Jégou, Matthijs Douze, and Cordelia Schmid. 2011. Product Quantization for Nearest Neighbor Search. *IEEE Trans. Pattern Anal. Mach. Intell.* 33, 1 (2011), 117–128. doi:10.1109/TPAMI.2010.57

[29] Evgenios M Kornaropoulos, Charalampos Papamanthou, and Roberto Tamassia. 2019. Data recovery on encrypted databases with k-nearest neighbor query leakage. In *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE, 1033–1050.

[30] Nishat Koti, Varsha Bhat Kukkala, Arpita Patra, and Bhavish Raj Gopal. 2024. Graphiti: Secure Graph Computation Made More Scalable. *IACR Cryptol. ePrint Arch.* (2024), 1756. https://eprint.iacr.org/2024/1756

[31] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 11 (1998), 2278–2324. doi:10.1109/5.726791

[32] 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. 2020. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin (Eds.). https://proceedings.neurips.cc/paper/2020/hash/6b493230205f780e1bc26945df7481e5-Abstract.html

[33] J. Liu, J. Li, D. Wu, and K. Ren. 2024. PIRANA: Faster Multi-query PIR via Constant-weight Codes. In *2024 IEEE Symposium on Security and Privacy (SP)*. IEEE Computer Society, Los Alamitos, CA, USA, 43–43. doi:10.1109/SP54263.2024.00039

[34] Manhua Liu, Xudong Jiang, and Alex Chichung Kot. 2007. Efficient fingerprint search based on database clustering. *Pattern Recognition* 40, 6 (2007), 1793–1803.

[35] Wanqi Liu, Hanchen Wang, Ying Zhang, Wei Wang, Lu Qin, and Xuemin Lin. 2021. EI-LSH: An early-termination driven I/O efficient incremental c-approximate nearest neighbor search. *VLDB J.* 30, 2 (2021), 215–235. doi:10.1007/S00778-020-00635-4

[36] Wen-jie Lu, Zhicong Huang, Zhen Gu, Jingyu Li, Jian Liu, Kui Ren, Cheng Hong, Tao Wei, and Wenguang Chen. 2023. BumbleBee: Secure Two-party Inference Framework for Large Transformers. *IACR Cryptol. ePrint Arch.* (2023), 1678. https://eprint.iacr.org/2023/1678

[37] Yury Malkov, Alexander Ponomarenko, Andrey Logvinov, and Vladimir Krylov. 2014. Approximate nearest neighbor algorithm based on navigable small world graphs. *Inf. Syst.* 45 (2014), 61–68. doi:10.1016/J.IS.2013.10.006

[38] Yury A. Malkov and Dmitry A. Yashunin. 2020. Efficient and Robust Approximate Nearest Neighbor Search Using Hierarchical Navigable Small World Graphs. *IEEE Trans. Pattern Anal. Mach. Intell.* 42, 4 (2020), 824–836. doi:10.1109/TPAMI.2018.2889473

[39] Yusuke Matsui, Yusuke Uchida, Hervé Jégou, and Shin'ichi Satoh. 2018. A survey of product quantization. *ITE Transactions on Media Technology and Applications* 6, 1 (2018), 2–10.

[40] Samir Jordan Menon and David J. Wu. 2022. 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*. IEEE, 930–947. doi:10.1109/SP46214.2022.9833700

[41] Muhammad Haris Mughees, Hao Chen, and Ling Ren. 2021. OnionPIR: Response Efficient Single-Server PIR. In *CCS '21: 2021 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, Republic of Korea, November 15 - 19, 2021*, Yongdae Kim, Jong Kim, Giovanni Vigna, and Elaine Shi (Eds.). ACM, 2292–2306. doi:10.1145/3460120.3485381

[42] Muhammad Haris Mughees and Ling Ren. 2023. Vectorized Batch Private Information Retrieval. In *44th IEEE Symposium on Security and Privacy, SP 2023, San Francisco, CA, USA, May 21-25, 2023*. IEEE, 437–452. doi:10.1109/SP46215.2023.10179329

[43] Moni Naor and Benny Pinkas. 1999. Oblivious transfer and polynomial evaluation. In *Proceedings of the thirty-first annual ACM symposium on Theory of computing*. 245–254.

[44] Benjamin Ostrovsky. 2024. Privacy-Preserving Dijkstra. In *Advances in Cryptology - CRYPTO 2024 - 44th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2024, Proceedings, Part IX (Lecture Notes in Computer Science, Vol. 14928)*, Leonid Reyzin and Douglas Stebila (Eds.). Springer, 74–110. doi:10.1007/978-3-031-68400-5_3

[45] Yun Peng, Byron Choi, Tsz Nam Chan, Jianye Yang, and Jianliang Xu. 2023. Efficient Approximate Nearest Neighbor Search in Multi-dimensional Databases. *Proc. ACM Manag. Data* 1, 1 (2023), 54:1–54:27. doi:10.1145/3588908

[46] Deevashwer Rathee, Mayank Rathee, Rahul Kranti Kiran Goli, Divya Gupta, Rahul Sharma, Nishanth Chandran, and Aseem Rastogi. 2021. 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*. IEEE, 1003–1020. doi:10.1109/SP40001.2021.00086

[47] Deevashwer Rathee, Mayank Rathee, Nishant Kumar, Nishanth Chandran, Divya Gupta, Aseem Rastogi, and Rahul Sharma. 2020. Cryptflow2: Practical 2-party secure inference. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*. 325–342.

[48] Sacha Servan-Schreiber, Simon Langowski, and Srinivas Devadas. 2022. 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*. IEEE, 911–929. doi:10.1109/SP46214.2022.9833702

[49] Anil Shanbhag, Holger Pirk, and Samuel Madden. 2018. Efficient Top-K query processing on massively parallel hardware. In *Proceedings of the 2018 International Conference on Management of Data*. ACM, 1557–1570.

[50] Mark Shaneck, Yongdae Kim, and Vipin Kumar. 2009. Privacy preserving nearest neighbor search. In *Machine Learning in Cyber Trust: Security, Privacy, and Reliability*. Springer, 247–276.

[51] Hayim Shaul, Dan Feldman, and Daniela Rus. 2020. Secure k-ish Nearest Neighbors Classifier. *Proc. Priv. Enhancing Technol.* 2020, 3 (2020), 42–61. doi:10.2478/POPETS-2020-0045

[52] Shang Song, Lin Liu, Rongmao Chen, Wei Peng, and Yi Wang. 2023. Secure Approximate Nearest Neighbor Search with Locality-Sensitive Hashing. In *Computer Security - ESORICS 2023 - 28th European Symposium on Research in Computer Security, The Hague, The Netherlands, September 25-29, 2023, Proceedings, Part III (Lecture Notes in Computer Science, Vol. 14346)*, Gene Tsudik, Mauro Conti, Kaitai Liang, and Georgios Smaragdakis (Eds.). Springer, 411–430. doi:10.1007/978-3-031-51479-1_21

[53] Ebrahim M Songhori, Siam U Hussain, Ahmad-Reza Sadeghi, and Farinaz Koushanfar. 2015. Compacting privacy-preserving k-nearest neighbor search using logic synthesis. In *Proceedings of the 52nd Annual Design Automation Conference*. 1–6.

[54] Yifang Sun, Wei Wang, Jianbin Qin, Ying Zhang, and Xuemin Lin. 2014. SRS: Solving c-Approximate Nearest Neighbor Queries in High Dimensional Euclidean Space with a Tiny Index. *Proc. VLDB Endow.* 8, 1 (2014), 1–12. doi:10.14778/2735461.2735462

[55] Adithya Vadapalli, Ryan Henry, and Ian Goldberg. 2023. Duoram: A {Bandwidth-Efficient} distributed {ORAM} for 2-and 3-party computation. In *32nd USENIX Security Symposium (USENIX Security 23)*. 3907–3924.

[56] Avery Wang. 2003. An Industrial Strength Audio Search Algorithm. In *ISMIR 2003, 4th International Conference on Music Information Retrieval, Baltimore, Maryland, USA, October 27-30, 2003, Proceedings*.

[57] Jingdong Wang, Naiyan Wang, You Jia, Jian Li, Gang Zeng, Hongbin Zha, and Xian-Sheng Hua. 2014. Trinary-Projection Trees for Approximate Nearest Neighbor Search. *IEEE Trans. Pattern Anal. Mach. Intell.* 36, 2 (2014), 388–403. doi:10.1109/TPAMI.2013.125

[58] Xiao Wang, Alex J. Malozemoff, and Jonathan Katz. 2016. EMP-toolkit: Efficient MultiParty computation toolkit. https://github.com/emp-toolkit.

[59] Xiang Wu, Ruiqi Guo, Ananda Theertha Suresh, Sanjiv Kumar, Daniel N. Holtmann-Rice, David Simcha, and Felix X. Yu. 2017. Multiscale Quantization for Fast Similarity Search. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett (Eds.). 5745–5755. https://proceedings.neurips.cc/paper/2017/hash/b6617980ce90f637e68c3ebe8b9be745-Abstract.html

[60] Kun Yang, Chengliang Tian, Hequn Xian, Weizhong Tian, and Yan Zhang. 2023. Query on the cloud: improved privacy-preserving k-nearest neighbor classification over the outsourced database. *World Wide Web* 26, 4 (2023), 1747–1774.

[61] Andrew Chi-Chih Yao. 1982. Protocols for Secure Computations (Extended Abstract). In *23rd Annual Symposium on Foundations of Computer Science, Chicago, Illinois, USA, 3-5 November 1982*. IEEE Computer Society, 160–164. doi:10.1109/SFCS.1982.38

[62] Andrew Chi-Chih Yao. 1986. How to Generate and Exchange Secrets (Extended Abstract). In *27th Annual Symposium on Foundations of Computer Science, Toronto, Canada, 27-29 October 1986*. IEEE Computer Society, 162–167. doi:10.1109/SFCS.1986.25

[63] Yuxin Zheng, Qi Guo, Anthony K. H. Tung, and Sai Wu. 2016. LazyLSH: Approximate Nearest Neighbor Search for Multiple Distance Functions with a Single Index. In *Proceedings of the 2016 International Conference on Management of Data, SIGMOD Conference 2016, San Francisco, CA, USA, June 26 - July 01, 2016*, Fatma Özcan, Georgia Koutrika, and Sam Madden (Eds.). ACM, 2023–2037. doi:10.1145/2882903.2882930

[64] Mingxun Zhou, Andrew Park, Wenting Zheng, and Elaine Shi. 2023. PIANO: Extremely Simple, Single-Server PIR with Sublinear Server Computation. In *2024 IEEE Symposium on Security and Privacy (SP)*. IEEE Computer Society, 55–55.

[65] Mingxun Zhou, Elaine Shi, and Giulia Fanti. 2024. Pacmann: Efficient Private Approximate Nearest Neighbor Search. Cryptology ePrint Archive, Paper 2024/1600. https://eprint.iacr.org/2024/1600

[66] Jinhao Zhu, Liana Patel, Matei Zaharia, and Raluca Ada Popa. 2025. Compass: encrypted semantic search with high accuracy. In *19th USENIX Symposium on Operating Systems Design and Implementation (OSDI 25)*. 915–938.

[67] Martin Zuber and Renaud Sirdey. 2021. Efficient homomorphic evaluation of k-NN classifiers. *Proc. Priv. Enhancing Technol.* 2021, 2 (2021), 111–129. doi:10.2478/POPETS-2021-0020

A Naive Expand

Algorithm 3 presents the "naive" expand algorithm in [2]. We remove the inverse process in it to make it suitable for our Algorithm 2.

Algorithm 3 Naive expand in [2] (Figure 3. line 6 - 12)

Input: $query, n$
Output: a vector of n ciphertexts $\langle \mathrm{ct}_1, \mathrm{ct}_2, ..., \mathrm{ct}_n \rangle$

function NaiveExpand($query, n$)
  Find smallest $m = 2^l$ such that $m \geq n$
  $ciphertexts \leftarrow [query]$
  for $j \leftarrow 0, ..., l-1$ **do**
    for $k \leftarrow 0, ..., 2^j - 1$ **do**
      $c_0 \leftarrow ciphertexts[k]$
      $c_1 \leftarrow c_0 \cdot x^{-2j}$
      $c'_k \leftarrow c_0 + \mathrm{Sub}(c_0, N/2^j + 1)$
      $c'_{k+2j} \leftarrow c_1 + \mathrm{Sub}(c_1, N/2^j + 1)$
    end for
    $ciphertexts \leftarrow [c'_0, ..., c'_{2^{j+1}-1}]$
  end for
  return $ciphertexts$
end function

B Ideal functionalities for Subroutines

$\mathcal{F}_{\text{ApproxTopK}}$

Public Parameters: input size t, input bit-length ℓ_d, output size k, bin size l, truncate bit size r.

- On input $\langle [\![d_1]\!]_c, [\![d_2]\!]_c, ..., [\![d_t]\!]_c \rangle$ from C.
- On input $\langle [\![d_1]\!]_s, [\![d_2]\!]_s, ..., [\![d_t]\!]_s \rangle$ from $\mathcal{S}$.
- Compute $d_i = [\![d_i]\!]_c + [\![d_i]\!]_s \pmod{2^{\ell_d}}$ and set $d'_i = \lfloor d_i / 2^r \rfloor$. Generate $idlist = \langle 1, 2, ..., t \rangle$ and sample a random permutation δ. Apply it to $idlist$ $\langle \delta(1), \delta(2), .., \delta(t) \rangle$. Let $\langle (v_1, ind_1), ..., (v_k, ind_k) \rangle \leftarrow$ ApproxTopk($\langle (d'_1, \delta(1)), ..., (d'_t, \delta(t)) \rangle, k, l$). Output δ to $\mathcal{S}$ and $\langle ind_1, ..., ind_k \rangle$ to C.

Figure 6: Ideal functionality $\mathcal{F}_{\text{ApproxTopK}}$

$\mathcal{F}_{\text{BatchPIR}}$

Public Parameters: Size of db (number of items t, each item size ℓ), number of queries L, number of buckets B, each bucket size n_b, hash function $\mathcal{H}$ in cuckoo hashing.

- On input $\langle id_1, id_2, ..., id_L \rangle$ from C.
- On input db and a permutation δ from $\mathcal{S}$.
- Apply the permutation to db to get db′. Apply the cuckoo hashing algorithm to encode the db′ into B buckets, let x_i^j represent i^{th} item in j^{th} bucket. Encode L queries into $\langle id'_1, id'_2, ..., id'_B \rangle$. Retrieve the information $\vec{x} = \langle x^1_{id'_1}, x^2_{id'_2}, ..., x^B_{id'_B} \rangle$. Sample a random array $\vec{r} = \langle r_1, ..., r_B \rangle$.
- Output $\vec{r}$ to C and $\vec{x} - \vec{r}$ to $\mathcal{S}$.

Figure 7: Ideal functionality $\mathcal{F}_{\text{BatchPIR}}$

$\mathcal{F}_{\text{ExactTopK}}$

Public Parameters: input size n, input bit-length ℓ_d, bit-length of id ℓ_{bid}, output size k, truncate bit size r.

- Input $\langle [\![d_1]\!]_c, ..., [\![d_n]\!]_c \rangle$, $\langle [\![id_1]\!]_c, ..., [\![id_n]\!]_c \rangle$ from C.
- Input $\langle [\![d_1]\!]_s, ..., [\![d_n]\!]_s \rangle$, $\langle [\![id_1]\!]_s, ..., [\![id_n]\!]_s \rangle$ from $\mathcal{S}$.
- Compute $d_i = [\![d_i]\!]_c + [\![d_i]\!]_s \pmod{2^{l_d}}$ and set $d'_i = \lfloor d_i / 2^r \rfloor$, compute $id_i = [\![id_i]\!]_c + [\![id_i]\!]_s$. Follow Algorithm 4, let $\langle (v_1, ind_1), ..., (v_k, ind_k) \rangle \leftarrow$ NetworkTopk($\langle (d'_1, id_1), ..., (d'_n, id_n) \rangle, k$). Sample a random array $\langle s_1, ..., s_k \rangle$, output it to $\mathcal{S}$ and output$\langle ind_1 - s_1, ..., ind_k - s_k \rangle$ to C.

Figure 8: Ideal functionality $\mathcal{F}_{\text{ExactTopK}}$

Here we define the ideal functionalities of key building block in our protocol. We securely implement the $\mathcal{F}_{\text{ExactTopK}}$ using GC

Algorithm 4 Exact Top-k selection algorithm

Input: $\vec{v} = \langle (key_1, id_1), ..., (key_n, id_n) \rangle$, k
Output: $\langle (o_1, ID_1), ..., (o_k, ID_k) \rangle$

function ExactTopK($\vec{v}, n, k$)
  $\vec{v} \leftarrow$ PartialSort($\vec{v}, n, k$)
  $l \leftarrow n/k$ ▷ For simplicity, we assume n divides k
  while $l > 1$ **do**
    for $i \leftarrow 1, ..., \lfloor l/2 \rfloor$ **do**
      $l, r \leftarrow (i-1)k + 1, ik$
      $l', r' \leftarrow (i-1)k + \lceil l/2 \rceil + 1, ik + \lceil l/2 \rceil$
      $\vec{v}[l:r] \leftarrow$ KMerge($\vec{v}[l:r], \vec{v}[l':r'], k$)
      $\vec{v}[l:r] \leftarrow$ BitonicMerge($\vec{v}[l:r], k$)
    end for
    $l \leftarrow \lceil l/2 \rceil$
    $\vec{v} \leftarrow \vec{v}[1:lk]$
  end while
  return $\vec{v}$
end function

function PartialSort($\vec{v}, n, k$)
  $l \leftarrow n/k$ ▷ For simplicity, we assume n divides k
  for $i \leftarrow 1, ..., l$ **do**
    $\vec{v}^i \leftarrow$ Sort($\vec{v}[(i-1)k + 1{:}ik], k$)
      ▷ where $\vec{v}^i$ is the input sorted ascending by $v.key$
  end for
  return $\{\vec{v}^1 || \vec{v}^2 || ... || \vec{v}^l\}$
end function

function KMerge($\vec{v}_1, \vec{v}_2, k$) ▷ Top-kMerge
  for $i \leftarrow 1, ..., k$ **do**
    $j \leftarrow k - i + 1$
    $\vec{v}_1[i], \vec{v}_2[j] \leftarrow$ CompSwap($\vec{v}_1[i], \vec{v}_2[j]$)
  end for
  return $\vec{v}_1$
end function

function BitonicMerge($\vec{v}, k$)
  if $k <= 1$ **then**
    return $\vec{v}$
  end if
  $m \leftarrow 2^{\lfloor \log k \rfloor}$
  for $i \leftarrow 1, ..., k - m$ **do**
    $\vec{v}[i], \vec{v}[i+m] \leftarrow$ CompSwap($\vec{v}[i], \vec{v}[i+m]$)
  end for
  $\vec{v}[1{:}m] \leftarrow$ BitonicMerge($\vec{v}[1{:}m], m$)
  $\vec{v}[m+1{:}k], \leftarrow$ BitonicMerge($\vec{v}[m+1:k], k-m$)
  return $\vec{v}$
end function

function CompSwap(v_1, v_2)
  $b \leftarrow$ Compare($v_1.key, v_2.key$)
  $v_{min}, v_{max} \leftarrow$ Mux(v_1, v_2, b)
  return (v_{min}, v_{max})
end function

by optimizing the computation circuit complexity. The security can easily follow the GC protocol. We implement $\mathcal{F}_{\text{ApproxTopK}}$ using OT [27] and GC protocol. The security proof of it is easy to see in the $(\mathcal{F}_{\text{compare}}, \mathcal{F}_{\text{mux}})$-hybrid model since only OT message and random value has been exchanged. Note that $\mathcal{F}_{\text{ApproxTopK}}$ outputs the $\langle ind_1, ..., ind_k \rangle$ to client. The output can be easily simulated by uniform random samples values (Lemma F.2). Cuckoo hashing can be computed locally by each party without compromising privacy [42]. The security proof is easy to see in $(\mathcal{F}_{\text{PIR2A}}, \mathcal{F}_{\text{trunc}})$-hybrid model. The security of our shuffle-and-reveal protocol is easy to see in $(\mathcal{F}_{\text{ApproxTopK}}, \mathcal{F}_{\text{BatchPIR}})$-hybrid model.

C Top-k Selection Network

Algorithm 4 presents our optimized top-k selection network.

D Full Protocol

In this section, we present the detailed version of our full protocol $\Pi_{\textbf{Panther}}$ in Figure 9. We detail the protocols used to achieve the ideal functionality for each subroutine. We briefly describe several 2-party functionalities used in the detailed protocol: $\mathcal{F}_{\text{cmp}}$ takes inputs $(\llbracket x \rrbracket, \llbracket y \rrbracket)$ and outputs $\llbracket \mathbf{1}\{x < y\} \rrbracket^B$, which is the boolean share indicating whether $x < y$. $\mathcal{F}_{\text{mux}}$ takes inputs $(\llbracket x \rrbracket, \llbracket y \rrbracket, \llbracket c \rrbracket^B)$ and outputs $\llbracket c?x : y \rrbracket$, representing the conditional selection of x or y based on the boolean value c. $\mathcal{F}_{\text{Trunc}}$ takes inputs $\llbracket x \rrbracket$ and a public parameter f (number of bits to truncate) and outputs $\llbracket \lfloor x/2^f \rfloor \rrbracket$, the truncated value of x. $\mathcal{F}_{\text{MSB}}$ takes input $\llbracket x \rrbracket$ and outputs $\llbracket w \rrbracket^B$, the boolean share of the most significant bit (MSB) of x. Finally, $\mathcal{F}_{\text{extend}}$ lift input $\llbracket x \rrbracket$ from a smaller ring to a larger ring.

E Inefficiency of DUORAM's subroutines

In Section 5 of the DUORAM paper [55], they invoke the generic OT-based PIR-to-SPIR transform by Naor and Pinkas [43]. We rewrite the transform (Protocol 2.1 of [43]) as follows:

We can see that the receiver needs to generate fresh keys and re-encrypt the database for each query. To see why, assume $N = 3$, and sender allows receiver to fetch X_0 and X_3 without re-encrypting the database. The receiver receives $K_{1,0}$ and $K_{2,0}$ to decrypt Y_0 and obtain X_0, and $K_{1,1}$ and $K_{2,1}$ to decrypt Y_3 and obtain X_3. However, with these keys—$K_{1,0}$, $K_{2,0}$, $K_{1,1}$, and $K_{2,1}$—the receiver can also decrypt Y_1, which is encrypted using $K_{1,0}$ and $K_{2,1}$, thereby accessing X_1, an item she was not authorized to retrieve. This is insecure because, in Line 5 of the protocol, the receiver may obtain more Y_I values than required (this is allowed in PIR), potentially including Y_1, and use the accumulated keys to decrypt unauthorized items.

F Security Proof

F.1 PIR with Level-2 H2A Protocol

We propose PIR with level-2 H2A protocol Π_{PIR2A} in Figure 3, which allows $\mathcal{C}$ to search an item $\text{db}[id]$ from $\mathcal{S}$, with the response secret-shared by two parties and potentially contain at most a one-bit error. $\mathcal{C}$ learns nothing about db and $\mathcal{S}$ learns nothing about query index id.

Proof. We proof the privacy of Π_{PIR2A} here.
(Corrupted Server.) We construct a simulator for the server. First, the simulator generates a key sk for the AHE scheme. In the real protocol, the first message from the client to the server consists of $\text{RLWE}_{\text{sk}}^{N,q,p}(x^{id_1})$ and $\text{RLWE}_{\text{sk}'}^{N',q',p'}(x^{id_2})$. The simulator just sends $\text{RLWE}_{\text{sk}}^{N,q,p}(0)$ and $\text{RLWE}_{\text{sk}'}^{N',q',p'}(0)$ to the server. Based on the

Public Parameters: number of elements n, number of final search results k, number of clusters t, number of retrieved clusters u, number of returned points B by batch PIR, and other hyperparameters in ANNS and batch PIR. The approximate top-k parameters: number of bins k'. Point dimension d and bitwidth ℓ_0. Parameters in leveled HE.
Input: C inputs a query $\vec{q}$. $\mathcal{S}$ inputs the database db = $\langle(\vec{p}_1, \mathrm{ID}(\vec{p}_1)), ..., (\vec{p}_n, \mathrm{ID}(\vec{p}_n))\rangle$, where $\vec{p}_i$ is a point and $\mathrm{ID}(\vec{p}_i)$ is its identifier. Both the query and the points have been quantized to the same bit length.
Output: C receives the identifiers of the top-k nearest points to $\vec{q}$.

Setup: $\mathcal{S}$ locally partitions db into t disjoint clusters $\langle C_1, ..., C_t\rangle$, with each cluster containing up to m points. The cluster centroids $\langle\vec{c}_1, ..., \vec{c}_t\rangle$ are chosen to approximately minimize the sum of distances between each centroid and all points in its cluster (e.g., via k-means). C calls **KeyGen** with different parameters to generate RLWE secret keys: sk_d for distance computation, and sk_1 and sk_2 for batch PIR. ▷ One-time preprocessing

---

1: $\mathcal{S}$ randomly shuffles the clusters and their centroids to generate: db'=$\langle(idx_1, C_1), ..., (idx_t, C_t)\rangle$ and $\mathrm{db}'_{ctr} = \langle(idx_1, \vec{c}_1), ..., (idx_t, \vec{c}_t)\rangle$, where idx_i represents the index of the cluster C_i after the shuffle. ▷ Shuffle

[Distance Computation (case 1)]:

2: C sends encrypted query point $\mathsf{ct}_i = \mathsf{RLWE}^{N,p,q}_{\mathsf{sk}_d}(\vec{q}[i])$ for $1 \le i \le d$ to $\mathcal{S}$.

3: $\mathcal{S}$ constructs $\hat{c}_{ik} = \vec{c}_{kN_d+1}[i] + \vec{c}_{kN_d+2}[i]x + ... + \vec{c}_{(k+1)N_d}[i]x^{N-1}$, and computes for $1 \le k \le \lceil t/N_d\rceil$: $\mathsf{d_k} = \boxplus_{i=1}^{d} \hat{\mathsf{c}}_{\mathsf{ik}} \cdot \mathsf{ct_i}$

4: Invoke H2A on each ciphertext result $\mathsf{d_k}$, and then C and $\mathcal{S}$ output $\langle[\![d_1]\!], ..., [\![d_t]\!]\rangle$, where $d_i = \langle\vec{q}, \vec{c}_i\rangle$.

5: C and $\mathcal{S}$ locally compute the shared distance result by $[\![dst_i]\!] = ||\vec{q}||_2^2 + ||\vec{c}_i||_2^2 - 2[\![d_i]\!]$.

[Approximate Tok-k]:

6: $\mathcal{S}$ and C input shared distances into $\mathcal{F}_{\mathrm{Trunc}}$ to truncate the lower bits. ▷ Protocol Π_{Trunc}[27]

7: $\mathcal{S}$ and C split the inputs into k' bins. For each bin, they invoke $\mathcal{F}_{\mathrm{cmp}}$ and $\mathcal{F}_{\mathrm{mux}}$ to compute the min element and its index of each bin $\langle([\![id_i^{min}]\!], [\![v_i^{min}]\!])\rangle_{i=1}^{k'}$ ▷ Protocol Π_{cmp} and Π_{mux}[27]

8: $\mathcal{S}$ and C use Garbled Circuit to compute the top-k selection network algorithm for the min results of each bin, producing u indices: $\langle[\![idx'_1]\!], [\![idx'_2]\!], ..., [\![idx'_u]\!]\rangle$.

[Batch PIR]:

9: Reveal the u shared indices $\langle idx'_1, ..., idx'_u\rangle$ to C. ▷ Reveal shuffled indices

10: C uses cuckoo hashing to encode u indices into B queries. ▷ If use 3-way cuckoo hashing, $B \approx 1.5u$.

11: C input queries and $\mathcal{S}$ input encoded db into Π_{PIR2A} (Fig. 3) to share the points in the candidate clusters and use $\mathcal{F}_{\mathrm{Trunc}}$ to fix 1-bit error. Use $\mathcal{F}_{\mathrm{extend}}$ to compose data. Then they hold $\langle([\![\vec{p}_i]\!], [\![ID(\vec{p}_i)]\!], [\![||\vec{p}_i||_2^2]\!])\rangle_{i=1}^{B\cdot N}$. ▷ B clusters contain up to $B \cdot m$ points.

▷ All PIR-retrieved data is shared in $\mathbb{Z}_{2^{\ell_1}}$, where $\ell_1 = k - 2$ and 2^k is the plaintext modulus in the first PIR dimension. If $||\vec{p}_i||_2^2$ exceeds 2^{ℓ_1}, $\mathcal{S}$ splits it into multiple parts. $\mathcal{S}$ and C extend these parts to $\ell_2 = 2\ell_0 + \log d$ bits and recombine them to recover $[\![||\vec{p}_i||_2^2]\!]$. The same operation is applied to $ID(\vec{p}_i)$. Note that $ID(\vec{p}_i)$ and $||\vec{p}_i||_2^2$ are shared in $\mathbb{Z}_{2^{\ell_2}}$, while $\vec{p}_i$ is shared in $\mathbb{Z}_{2^{\ell_1}}$.

[Distance Computation (case 2)]:

12: $\mathcal{S}$ and C use line 3-4 to compute the $[\![s_i]\!] = \langle\vec{q}, [\![\vec{p}_i]\!]_s\rangle$,for $1 \le i \le B \cdot m$.

13: $\mathcal{S}$ and C input $[\![\vec{p}_i]\!]$ into $\mathcal{F}_{\mathrm{MSB}}$ to get the wrap bit $[\![\vec{w}_i]\!]^B$. Then use OT to compute shared $\vec{w}_i \odot \vec{q}$ for result correction.

14: $\mathcal{S}$ and C compute the distance: $[\![pdst_i]\!] = ||\vec{q}||_2^2 + [\![||\vec{p}_i||_2^2]\!] - 2\langle\vec{q}, [\![\vec{p}_i]\!]_s\rangle - 2\langle\vec{q}, [\![\vec{p}_j]\!]_c\rangle + 2 \cdot 2^{\ell_1}[\![\langle\vec{w}_i, \vec{q}\rangle]\!] \mod 2^{\ell_2}$.

[Exact Top-k]:

15: $\mathcal{S}$ and C input shared distances into $\mathcal{F}_{\mathrm{Trunc}}$ to truncate the lower bits.

16: $\mathcal{S}$ and C use Garbled Circuit to compute a top-k selection network algorithm over the distances and identifiers, and output shares of the identifiers corresponding to the top-k smallest distances.

17: $\mathcal{S}$ reveals its share of above k identifiers to C.

18: C outputs the identifiers of the top-k nearest points.

Figure 9: Protocol Π_{Panther} for private ANNS.

RLWE assumption, these views are indistinguishable from the real protocol.

(Corrupted Client.) We construct a simulator for the client. The simulator simulates the message form server. In the real protocol, the server send the first ciphertext w'_0 output by H2A protocol. This is indistinguishable form $\mathrm{ct} \leftarrow \mathsf{RLWE}^{N',q',p'}_{\mathsf{pk}'}(0) \boxplus (\hat{r}, 0)$. Appendix C in [36] provides more detail about the security proof of H2A protocol. In the real protocol, the server sends another message w_1 from server. The simulator samples $\hat{r}_i \in_R \mathbb{A}_{N,q'}$. Based on the RLWE-assumption, it is impossible to distinguish between $\hat{r}_i$ and $\hat{a}_i$ in step 7, where $\hat{a}_i$ is re-randomized by adding a fresh ciphertext $\mathsf{RLWE}^{N,q,p}_{\mathsf{pk}}(0)$. Then the simulator computes $\mathrm{w}_s \leftarrow \sum_{i=1}^{n_2}(\mathrm{ct}_i^2 \cdot \hat{r}_i)$, where ct_i^2 can be locally computed from the input of client. This is indistinguishable from the real protocol message w_1. Note that we

Input: Sender inputs $N+1$ strings $X_0, X_1, \ldots, X_N \in \{0,1\}^\ell$ and receiver inputs an index $I \in \{0,1,\ldots,N\}$.

▷ where the sender can be seen as $\mathcal{S}$ and the receiver as C.

Output: Receiver gets X_I.

1: Let $k = \log_2(N+1)$ (assuming $N = 2^k - 1$ for simplicity)
2: For each $j = 1$ to k and each $b \in \{0,1\}$, the sender chooses a random key $K_{j,b}$ for a pseudorandom function F
3: For each $m = 0$ to N, let m's binary representation be $m_1 m_2 \ldots m_k \in \{0,1\}^k$
4: The sender computes $Y_m = X_m \oplus \bigoplus_{j=1}^{k} F_{K_{j,m_j}}(j)$ for each $m = 0$ to N
5: The sender sends Y_I to the receiver via PIR
6: Let the binary representation of I be $i_1 i_2 \ldots i_k \in \{0,1\}^k$
7: **for** $j = 1$ to k **do**
8: The sender and receiver perform a 1-out-of-2 OT protocol where the sender's inputs are $K_{j,0}$ and $K_{j,1}$, and the receiver chooses to receive K_{j,i_j} based on the bit i_j
9: **end for**
10: After all OTs, the receiver has $K_{1,i_1}, K_{2,i_2}, \ldots, K_{k,i_k}$
11: The receiver computes $X_I = Y_I \oplus \bigoplus_{j=1}^{k} F_{K_{j,i_j}}(j)$

Figure 10: DUORAM: OT-based PIR-to-SPIR

don't need to mask the noise in w_1, because it only contains the information about random polynomials $\hat{a}_i$.

□

F.2 Shuffle-and-reveal

In this section, we aim to demonstrate that revealing the shuffled index to the client does not result in any additional information leakage.

Lemma 2. *Consider the following two processes:*

(1) Fix k distinct output index $\langle o_1, o_2, .., o_k\rangle$. Apply a random permutation δ to the $ID = \delta(\langle 1, 2, .., t\rangle) = \langle id_1, ..., id_t\rangle$. Define output $O = \langle id_{o_1}, ..., id_{o_k}\rangle$.

(2) Given the set $ID = \langle 1, 2, .., t\rangle$. Randomly samples k positions without replacement and output P.

The O and P are equally distributed, which are perfectly distinguishable.

The proof is easy to see by using combinatorial theory to calculate probability. In other words, the output O can be easily simulated by randomly samples P, which is independent of $\langle o_1, o_2, .., o_k\rangle$, thus O could safely be public to C as PIR queries.

F.3 The full protocol Π_{Panther}

Here we provide the security proof of the full protocol Π_{Panther}. For simplicity, we have omitted certain details of the clustering-based ANNS algorithms, such as using a stash to save some specific points for a linear scan and dividing clusters into multiple groups and searching nearest clusters in each group. These steps can be simply implemented by our sub-protocol.

Theorem F.1. *Assuming the hardness of decision-RLWE problem, the protocol Π_{Panther} securely computes ANNS in the ($\mathcal{F}_{\text{BatchPIR}}$, $\mathcal{F}_{\text{ApproxTopK}}$,$\mathcal{F}_{\text{ExactTopK}}$)-hybrid model in the presence of a semi-honest adversary $\mathcal{A}$.*

Proof. Again the correctness is easy to verify. We only briefly proof the protocol privacy here.

(Corrupted Client.) The simulator generates a secret key sk. The first message of the real protocol. It samples random $\mathsf{RLWE}_{\mathsf{pk}}^{N,q,p}(0) \boxplus (\hat{r}^i, 0)$ and sends it to the server. Then it feeds it to the sub-protocol and forwards the output to the client. The simulator then forwards the r_{ij} to $\mathcal{F}_{\text{ApproxTopK}}$ and obtains the indices $\langle rid_1, ..., rid_k\rangle$, which is distinguishable from uniformly random samples sequence (Lemma F.2). Then feed the indices to $\mathcal{F}_{\text{BatchPIR}}$. The return message is distinguishable from uniform based on the RLWE assumption and uniformly random mask. Then it samples random distance result rd_i and sends $\mathsf{RLWE}_{\mathsf{pk}}^{N,q,p}(rd_i) \boxplus \mathsf{RLWE}_{\mathsf{pk}}^{N,q,p}(0) \boxplus (\hat{r}'^i, 0)$ to the client. Then it feeds the value of decryption into $\mathcal{F}_{\text{ExactTopK}}$. At last it forwards output to the client.

(Corrupted Server.) The simulator for the server works in almost the same fashion, the simulator generates a secret key sk. The real client sends the encryption of $\vec{q}[i]$ for $1 \le i \le d$. The simulator sends d encryption of zeros to the server. This is indistinguishable from the real protocol based on the RLWE assumption.

Since the intermediate values revealed to the client and server all independent uniformly random values, the view generated from simulator is indistinguishable from the real protocol. □