FedKNN: Secure Federated k-Nearest Neighbor Search

XINYI ZHANG, Hong Kong Baptist University, China
QICHEN WANG, Hong Kong Baptist University, China
CHENG XU, Hong Kong Baptist University, China
YUN PENG, Guangzhou University, Guangdong Provincial Key Laboratory of Blockchain Security, China
JIANLIANG XU, Hong Kong Baptist University, China

Nearest neighbor search is a fundamental task in various domains, such as federated learning, data mining, information retrieval, and biomedicine. With the increasing need to utilize data from different organizations while respecting privacy regulations, private data federation has emerged as a promising solution. However, it is costly to directly apply existing approaches to federated k-nearest neighbor (kNN) search with difficult-to-compute distance functions, like graph or sequence similarity. To address this challenge, we propose FedKNN, a system that supports secure federated kNN search queries with a wide range of similarity measurements. Our system is equipped with a new Distribution-Aware kNN (DANN) algorithm to minimize unnecessary local computations while protecting data privacy. We further develop DANN*, a secure version of DANN that satisfies differential obliviousness. Extensive evaluations show that FedKNN outperforms state-of-the-art solutions, achieving up to 4.8× improvement on federated graph kNN search and up to 2.7× improvement on federated sequence kNN search. Additionally, our approach offers a trade-off between privacy and efficiency, providing strong privacy guarantees with minimal overhead.

CCS Concepts: • **Information systems → Combination, fusion and federated search**; • **Security and privacy → Management and querying of encrypted data**; *Privacy-preserving protocols*; *Hardware-based security protocols*.

Additional Key Words and Phrases: Federated analytics, kNN search, trusted execution environment (TEE), differential obliviousness

ACM Reference Format:
Xinyi Zhang, Qichen Wang, Cheng Xu, Yun Peng, and Jianliang Xu. 2024. FedKNN: Secure Federated k-Nearest Neighbor Search. *Proc. ACM Manag. Data* 2, 1 (SIGMOD), Article 11 (February 2024), 26 pages. https://doi.org/10.1145/3639266

1 INTRODUCTION

Nearest neighbor search is a fundamental problem in computer science that finds applications in various domains, including federated learning [15], data mining [30], information retrieval [49], and biomedicine [47]. With the rapid growth of data across organizations, utilizing data from multiple sources for nearest neighbor search has become increasingly prevalent. Here are two examples of such application scenarios:

Authors' addresses: Xinyi Zhang, Hong Kong Baptist University, Hong Kong, China, csxyzhang@comp.hkbu.edu.hk; Qichen Wang, Hong Kong Baptist University, Hong Kong, China, qcwang@comp.hkbu.edu.hk; Cheng Xu, Hong Kong Baptist University, Hong Kong, China, chengxu@comp.hkbu.edu.hk; Yun Peng, Guangzhou University, Guangdong Provincial Key Laboratory of Blockchain Security, Guangzhou, Guangdong, China, yunpeng@gzhu.edu.cn; Jianliang Xu, Hong Kong Baptist University, Hong Kong, China, xujl@comp.hkbu.edu.hk.

2836-6573/2024/2-ART11 $15.00
https://doi.org/10.1145/3639266

Table 1. Running Time Results (s) ($k = 128$, $m = 8$, Uniform)

Dataset	Baseline		HuFu-Ext [41]	
	Local Cost	Other Costs	Local Cost	Other Costs
AIDS	42.72	0.21	15.62	0.73
SYN	27.18	0.18	13.69	0.77
DBLP	4.29	0.20	5.61	0.67
GENOME	700.94	0.22	751.13	0.71

Example 1.1. Breast and ovarian cancers are genetically linked. To improve patient outcomes, hospitals can collaborate and use gene expression data to measure the DNA similarity of patients' cancer cells. Recognizing these commonalities can improve diagnosis accuracy and treatment outcomes. Using nearest neighbor search across different hospitals enables efficient identification of similar cancer cell gene expressions, allowing researchers and doctors to uncover significant genetic patterns and improve patient care.

Example 1.2. Pharmaceutical companies often seek to search for similar drugs in other companies' databases to improve the efficiency of drug development (e.g., AIDS). With vast drug candidate databases containing chemical structures and biological activities, companies can use nearest neighbor search to identify similar drug structures across databases. If they find a drug of interest, they can collaborate with the corresponding company to speed up drug development.

In the above application scenarios, a basic approach for nearest neighbor search is to share all relevant data with a third party (e.g., a cloud computing platform). This third party is responsible for conducting the computation and producing results. However, this approach faces several challenges. Sharing private data with a third party is often prohibited by privacy regulations (e.g., GDPR [17], CCPA [7], PIPL [1]), and can also raise concerns regarding liability and commercial competition. Moreover, the distribution and availability of data may vary across different organizations and are subject to changes, making it challenging to maintain a consistent and up-to-date global view of the data. These issues hinder the full utilization of combined data from all parties involved, leading to what is known as the data isolation problem [5].

Given the challenges associated with data silos and the limitations of sharing private data with third parties, the concept of private data federation has been proposed [3]. In a private data federation, participating parties agree on a shared schema and permissible queries. Each party runs these queries on their private data, sharing only the final results. Notably, individual parties' databases or intermediate results generated during query processing remain undisclosed. This approach enables secure collaboration while preserving the privacy and confidentiality of each party's data.

Several studies have focused on secure query processing over private data federation, such as SMCQL [3], Conclave [42], Senate [35], and SECRECY [32] for SQL queries, and Hu-Fu [41] for spatial queries. While these works have made initial progress towards federated nearest neighbor search using security tools like secure multi-party computation (SMC), they assume that the distance functions, such as Euclidean or Manhattan distance, can be easily computed, and the bottleneck lies in security tools.

For nearest neighbor search queries with difficult-to-compute distance functions, such as graph or sequence similarity, local computations at each participant can become the bottleneck, rendering existing solutions inefficient. In Table 1, following the system settings in Section 6, we provide some preliminary results (in terms of query time) of nearest neighbor search by applying the baseline and the state-of-the-art [41] algorithms directly to these types of queries. The results clearly show that

the overhead of local computations is the primary bottleneck of query efficiency. Moreover, existing solutions use SMC for secure interaction among participants, which can complicate the execution of operations such as sorting. This exacerbates the local computational burden for nearest neighbor queries, especially when there is a lack of prior knowledge about the data.

This paper aims to address the challenges of efficient and secure nearest neighbor search in private data federation, focusing on queries where the local computational overhead dominates. We follow the semi-honest threat model and adopt a hardware enclave-based approach to efficiently support secure queries. To this end, we propose FedKNN, an efficient system for supporting federated k-nearest neighbor (kNN) search queries. FedKNN is equipped with our newly proposed Distribution-Aware kNN (DANN) processing algorithm to reduce the local computations required for each participating party, which is a key factor in improving system efficiency. The core idea of DANN is to obtain the possible distribution of query results over the private data of each party using a fast preprocessing round. This allows each party to compute local kNNs with different k values separately, minimizing unnecessary local computations. To protect private information during query processing, we enhance DANN with differential privacy techniques and further develop DANN*, which maintains efficiency while meeting the differential obliviousness requirement [8].

Our main contributions and results are summarized as follows:

- We present FedKNN, a pioneering system that can efficiently and securely compute a variety of kNN queries on a data federation, particularly for kNN queries where local computations are the performance bottleneck.
- We propose a new Distribution-Aware kNN (DANN) query algorithm that reduces the amount of unnecessary local computations while ensuring that the original data is not shared with third parties. We show that the expected complexity of DANN is close to the theoretical optimal and better than the baseline solution (Theorem 4.8).
- We develop DANN* using differential privacy and provide a detailed security analysis. Compared with the state-of-the-art solution [41], our approach satisfies differential obliviousness [8] and can offer provable privacy guarantees for each participant (Theorem 4.16).
- We implement a FedKNN prototype system and conduct extensive evaluations on various types of datasets. The results show that FedKNN outperforms current state-of-the-art solutions by up to 4.8× on federated graph kNN search and up to 2.7× on federated sequence kNN search. Our experiments also demonstrate that by adding differential privacy, our approach can provide strong privacy guarantees with only a small additional overhead.

In the rest of this paper, we present the kNN search and system model in Section 2 and describe the security basics in Section 3. We then propose our DANN and DANN* algorithms in Section 4 and prove that DANN* satisfies differential obliviousness. We present our system implementation in Section 5, followed by the evaluation results in Section 6. Finally, we review the related works in Section 7 and conclude the paper in Section 8.

2 BACKGROUND AND SYSTEM MODEL

2.1 k-Nearest Neighbor (kNN) Search

Consider a database $\mathcal{D}$, where each data object $o \in \mathcal{D}$ is represented as a tuple in the form of $o = \langle id,\ d \rangle$. Here, id is the unique identifier of the data object and d is the original data object (e.g., string, graph, or time series). The original data object d can be regarded as a point in the universe $\mathcal{U}$, where $\mathcal{U}$ is a metric space. The similarity of two points $x, y \in \mathcal{U}$ is defined by a similarity metric $\mathcal{M}(x, y)$.

Similarity Metrics. A variety of similarity metrics can be employed to measure the similarity or dissimilarity between data points in diverse applications. Examples include Euclidean distance and

Manhattan distance for geometric points, edit distance for strings and graphs, cosine similarity for vectors, and Jaccard distance for sets.

Some similarity metrics, such as Euclidean distance and Manhattan distance, can be computed in $O(l)$ time, where l represents the dimensionality of the space. When the dimensionality is relatively low, the cost of distance computation can be considered as a constant and omitted when analyzing the computational cost for KNN queries. This assumption is also prevalent in prior works [41]. However, certain similarity metrics, like edit distance, impose a higher computational burden.

Example 2.1. **Edit Distance.** The edit distance is a widely employed similarity metric for comparing two sequences [49, 50, 53] or graph data objects [9, 21, 28]. It is defined as the number of "edit operations" needed to transform one data object into another. Nevertheless, computing the edit distance for either sequences or graph data is NP-hard [14, 52].

A kNN search query Q is defined in the form of $\langle k,\ d_q \rangle$, where $d_q \in \mathcal{U}$ is the query point and k denotes the number of data objects $o \in \mathcal{D}$ that must be returned for the kNN query. Meanwhile, the query result set $Q(\mathcal{D})$ satisfies $|Q(\mathcal{D})| = k$ and $\forall o \in Q(\mathcal{D}), \forall o' \in \mathcal{D} - Q(\mathcal{D}), \mathcal{M}(d_q, o.d) \leq \mathcal{M}(d_q, o'.d)$.

An intuitive approach to processing kNNs queries involves computing the distance between each data object $o \in \mathcal{D}$ and the query point d_q, followed by sorting to retrieve the top-k results. However, this approach is inefficient as it computes distances between d_q and every $o \in \mathcal{D}$, even when $|\mathcal{D}|$ is significantly larger than k.

As a more effective alternative, we can construct an index $\mathcal{I}$ on the database $\mathcal{D}$, such as R-tree [11] for spatial data or search tree [9] for graph data. Utilizing preprocessing and pruning, the index is able to compute the kNN function in $O(k \times \text{cost}_d \times \text{cost}_u)$ time, with a hidden constant factor or logarithmic factor [9]:

- kNNCompute$(\mathcal{D}, k, d_q)$, for computing the kNNs of the query point d_q against all data objects in the database $\mathcal{D}$.

Here, cost_d represents the cost of computing the distance between any two points, and cost_u represents the unit cost required by kNNCompute$(\mathcal{D}, k, d_q)$, including the cost of accessing the index and the cost of retrieving data objects. For a given database instance $\mathcal{D}$ and similarity metric $\mathcal{M}$, the running time of kNNCompute$(\mathcal{D}, k, d_q)$ is generally proportional to k, which has also been verified by our experimental results (Figure 4 (a)) in Section 6.

Additionally, the index $\mathcal{I}$ may also assist in filtering the top-k lower-bound distances and support the following function as well:

- kLBCompute$(\mathcal{D}, k, d_q)$, for computing the top-k lower-bound distances of the query point d_q against all data objects in the database $\mathcal{D}$.

In general, kLBCompute$(\mathcal{D}, k, d_q)$ is much faster than kNNCompute$(\mathcal{D}, k, d_q)$. Therefore, we intend to leverage the lower-bound distances to estimate the distribution of top-k results across different parties, thereby reducing the need for costly distance computations.

It is worth noting that our study does not assume any particular similarity metric $\mathcal{M}$ and treat the kNN computation as a black-box process. Our system is flexible to work with any metrics and indices that provide the above two functions.

For the sake of simplicity, we normalize cost_d and cost_u to $O(1)$ in the remainder of this paper, assuming fixed $\mathcal{D}$ and $\mathcal{M}$. We also neglect other costs associated with local computations, as they are insignificant compared to $\text{cost}_d \times \text{cost}_u$. Our main goal is to minimize $\sum_i k_i$ for all kNNCompute$(\mathcal{D}, k, d_q)$ computations invoked within our system.

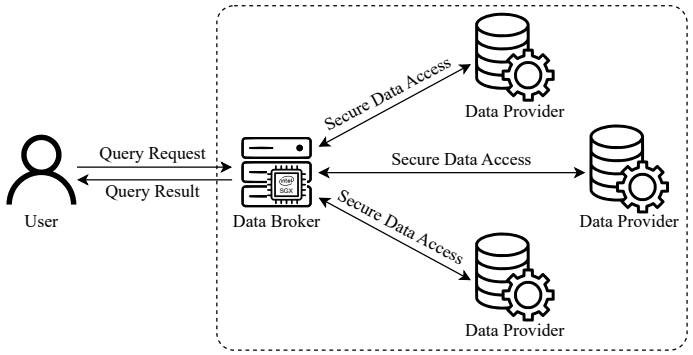


Fig. 1. System Model

2.2 Federated kNN

We consider the scenario of a private data federation, which is a collection of autonomous databases as depicted in Figure 1. This federation shares a unified query interface to offer data analytics services using the collective (sensitive) data of its members, without revealing unauthorized information to any party involved in the analytics process [3, 41, 42]. A private data federation F consists of three types of parties:

(1) *data providers* $\mathcal{P} = \{P_1, P_2, \cdots, P_m\}$, each data provider P_i holding a private database D_i, and $\mathcal{D} = D_1 \cup \cdots \cup D_m$;
(2) a *broker* responsible for planning and orchestrating the queries on $\mathcal{D}$; and
(3) a *user* who submits queries to the *broker*.

This paper focuses on federated kNN queries. Any user can submit a query Q to the broker, which compiles Q into a query plan. Subsequently, the data providers collaboratively execute a secure federated data analytics protocol. Once the protocol execution is complete, the broker returns the final result set r to the user. Table 2 summarizes the frequently used notations in this paper.

2.3 Threat Model

In federated computing, preserving the privacy and security of each party's data is of the utmost importance. In this paper, we adopt a semi-honest adversary model similar to prior works [3, 41, 42]. Specifically, we assume that there is no collusion among parties, and all parties in the private data federation are honest but curious. They adhere to the protocol but may attempt to infer information about other parties' private inputs. Consequently, we consider two potential security threats in the system: (i) the adversary may obtain raw, sensitive data in P_i during data processing or communication; and (ii) the adversary may observe the data access pattern during data processing or communication to infer sensitive data in P_i (e.g., the adversary can observe memory access and network traffic to infer the order and source of the results). We will give the formal definition of data access pattern in Section 3.2.

To address the first threat, the protocol should ensure data confidentiality in two aspects:

- **Data Processing.** This involves only the broker, which processes data from $\mathcal{P}$. To ensure data confidentiality, the input data from $\mathcal{P}$ should be encrypted, and all operations should be performed on encrypted data.
- **Data Transmission.** All data transmitted from $\mathcal{P}$ to the broker should be sent via a secure channel to prevent eavesdropping.

To counter the second threat, the protocol should protect the data access pattern in two aspects:

Table 2. Frequently Used Notations

Notation	Description
F	the private data federation
$\mathcal{P}$	the set of data providers
$\mathcal{D}$	the set of the private databases held at every P_i
m	the number of data providers in $\mathcal{P}$
ε	the privacy budget for the differential obliviousness
λ	the potential max error probability for the query
k_i	the number of local kNN results computed by P_i in one round
d	the data object in the database
d_q	the query point
$dist$	the distance between two data objects

- **Data Processing.** This involves the curious data provider and the broker. For the broker, the memory access pattern during data processing must be protected, as the broker can potentially infer sensitive data from $\mathcal{P}$ through statistical analysis. For the curious data provider, it can potentially deduce the data distributions of other data providers by analyzing the local computations required for processing a kNN query.
- **Data Transmission.** The data transmitted from $\mathcal{P}$ to the broker should have the same size when k is fixed. This minimizes the potential for the adversary to infer additional information about $\mathcal{P}$ by analyzing the communication volume.

Our core task is to design an efficient and secure federated data analytics protocol that improves the performance of kNN queries over the data federation while ensuring the privacy of data providers as defined above. In the following sections, we first present the security basics that underlie our solution, and then describe our system, FedKNN, which fulfills these requirements.

3 SECURITY BASICS

3.1 Hardware Enclaves

Hardware enclaves have recently gained popularity due to their two security features: isolated execution and sealing. While the specific details may differ depending on the vendor (e.g., Intel SGX [12] and AMD SEV [27]), the general concepts remain the same. First, the isolated execution of an enclave process restricts access to a subset of memory so that only that particular enclave can access it. No other processes on the same processor, including the OS, hypervisor, or system management module, can access that memory. Second, sealing allows encrypting and authenticating the enclave's data so that only the same enclave can decrypt it. However, the current enclave architecture leaks memory access patterns when accessing the enclave's memory (EPC) and the rest of the main memory, which may be sensitive in practice [48]. In our FedKNN, we assume the data broker is equipped with a hardware enclave. We leverage the enclave to improve performance while also designing methods to protect access patterns.

3.2 Obliviousness

Obliviousness [20] is a commonly used security definition for many oblivious data structures, providing protection for access patterns. Before introducing the formal definition of obliviousness, we first introduce the leakage function of a protocol Π. Let $\mathcal{L}(\mathcal{D})$ be a leakage function that has to be revealed during the execution of Π on a database $\mathcal{D}$. We assume that the size of the private input ($|\mathcal{D}|$), the query Q, the size of the final result set $|\Pi(\mathcal{D})|$, and the number of data providers m

are not sensitive. Based on this assumption, we have $\mathcal{L}(\mathcal{D}) = \{|\mathcal{D}|, |\Pi(\mathcal{D})|, Q, m\}$.[1] We first define the access pattern during the protocol execution.

Definition 3.1 (Access Pattern). Let

$$\mathbf{Access}^{\Pi}(\mathcal{D}) := (q_1, q_2, \cdots, q_z) \tag{1}$$

denote the access pattern of length z, which defines the adversary's observation during the execution of Π on $\mathcal{D}$. Each access $q \in \mathbf{Access}^{\Pi}(\mathcal{D})$ can be represented as a 5-tuple $\langle op, src, dst, addr, data \rangle$, where $op \in \{read, write\}$, src is the party owning the data, dst is the party accessing the data, $addr$ is the logical address accessed by the protocol, and $data$ is what is to be written or to be read. In case $src = dst$, the access denotes a processor accessing its local memory data (referred to as *memory access pattern*); otherwise, it denotes a processor accessing the data from other parties (referred to as *network access pattern*).

Depending on the protection level of the access pattern, obliviousness can be divided into two types: *full obliviousness* [29] and *differential obliviousness* [8]. We first present the definition of full obliviousness.

Definition 3.2 (Full Obliviousness [29, 36]). We say a protocol Π is fully oblivious if, given an arbitrary database $\bar{\mathcal{D}}$ and a real database $\mathcal{D}$ where $|\bar{\mathcal{D}}| = |\mathcal{D}|$, a polynomial-time adversary $\mathcal{A}$ only has a negligible advantage to distinguish $\bar{\mathcal{D}}$ and $\mathcal{D}$:

$$Pr[Sim^{\Pi}(\mathcal{L}, \mathbf{Access}^{\Pi}(\mathcal{D}), \mathcal{D}) = 1] \leq Pr[Sim^{\Pi}(\mathcal{L}, \mathbf{Access}^{\Pi}(\bar{\mathcal{D}}), \mathcal{D}) = 1] + \delta \tag{2}$$

The simulator $Sim^{\Pi}(\mathcal{L}, \mathbf{Access}^{\Pi}(\bar{\mathcal{D}}), \mathcal{D})$ will output 1 if $\mathcal{A}$ determines that $\bar{\mathcal{D}}$ is the same as $\mathcal{D}$ based on $\mathcal{L}$ and $\mathbf{Access}^{\Pi}(\bar{\mathcal{D}})$. Intuitively, the probability that the adversary can gain additional information beyond the leakage function is negligible and controlled by the security parameter δ.

However, achieving full obliviousness is expensive and will significantly degrade the query performance. To improve performance, a relaxed security definition, called (ε, δ)-differential obliviousness, has been proposed [8, 36, 46]. Rather than making the access pattern of the protocol indistinguishable for all inputs, differential obliviousness only necessitates that the access pattern satisfies differential privacy. Yet, unlike differential privacy, which often adds noise to the final results or raw data, differential obliviousness solely incorporates dummy operations during execution to conceal the access pattern. Therefore, differential obliviousness does not impact the query results. The formal definition of (ε, δ)-differential obliviousness is:

Definition 3.3 ((ε, δ)-Differential Obliviousness [8]). We say a protocol Π is (ε, δ)-oblivious if, for any two neighboring databases $\mathcal{D}_1$ and $\mathcal{D}_2$ where $|\mathcal{D}_1| = |\mathcal{D}_2|$ and $|\mathcal{D}_1 - \mathcal{D}_2| = 1$, the following condition is satisfied:

$$Pr[Sim^{\Pi}(\mathcal{L}, \mathbf{Access}^{\Pi}(\mathcal{D}_1), \mathcal{D}_1) = 1] \leq e^{\varepsilon} Pr[Sim^{\Pi}(\mathcal{L}, \mathbf{Access}^{\Pi}(\mathcal{D}_2), \mathcal{D}_1) = 1] + \delta \tag{3}$$

where ε represents the privacy budget and δ is a negligible probability for which ε does not hold.

3.3 Oblivious Data Processing

Oblivious Sort. Oblivious sorting [2] is a family of sorting algorithms whose memory access pattern is solely determined by the input data's length, regardless of the distribution of the data being sorted. It can be used as a building block to design many oblivious algorithms. One well-known approach to implementing an oblivious sort is the Bitonic sort [26], which has a time complexity of $O(N \log^2 N)$ to sort N elements.

[1] We use $\mathcal{L}$ and $\mathcal{L}(\mathcal{D})$ interchangeably when the context is clear.

Oblivious Priority Queue. The oblivious priority queue closely resembles a standard priority queue, with the crucial distinction that it prevents the leakage of internal state information, such as the queue's size, through memory access pattern during algorithm execution. There are several approaches to implementing an oblivious priority queue. One notable approach is to leverage oblivious RAM [38], which is particularly well-suited for managing large-capacity queues. In our paper, however, we use the oblivious sorting instead, as it has been shown to have better performance for small-capacity queues. The oblivious priority queue supports the following operations:

- $ObliPriorQueue \leftarrow \texttt{init}(N)$: initialize the oblivious priority queue with capacity N.
- $\texttt{Enqueue}(e)$: push one element e to the queue. If e is a tuple, the queue will sort based on its first attribute. It is important to note that the enqueue operation does not increase the queue's capacity. If the queue is full and the new element is smaller than the largest element in the queue, the largest element will be removed before adding the new element. Otherwise, the new element will not be added to the queue.
- $e \leftarrow \texttt{Dequeue}()$: retrieve the smallest non-dummy element e from the queue. If no non-dummy elements are in the queue, it returns a dummy element.
- $i \leftarrow \texttt{GetIndex}(e)$: use the linear scan to obliviously find the index of the element e in the queue. If e does not exist in the queue, it will return -1.

4 FedKNN ALGORITHMS

In this section, we present three kNN algorithms used in our FedKNN system. First, we present the baseline algorithm in Section 4.1, which minimizes the communication cost. To reduce the local computation overhead of the baseline algorithm, we propose a Distribution-Aware kNN (DANN) algorithm in Section 4.2. To provide provable security guarantees for DANN, we introduce DANN* in Section 4.3. We prove in Section 4.4 that DANN* satisfies $(\varepsilon, \delta)-$differential obliviousness.

4.1 Baseline Solution

Since each data provider P_i can compute kNNs locally, an intuitive approach to support federated kNN queries while maintaining obliviousness is as follows: each data provider computes and sends its local top-k results to the broker; the broker then uses oblivious sort [26] to sort all received results by distance and selects the top-k results to produce the final federated kNN results. To ensure execution confidentiality, all computations of the broker are performed within its enclave. Moreover, a secure channel is established between the broker and each data provider through the enclave for all communications. The baseline algorithm minimizes the communication cost as it involves only one round of communication, with a total volume of $O(m \cdot k)$, where m represents the number of data providers.

Example 4.1. We use Figure 2 as an example to illustrate the query procedure of the baseline solution. Suppose we have three data providers $\mathcal{P} = \{P_1, P_2, P_3\}$ in the data federation, and each P_i has three data objects in its private database D_i. A user wants to retrieve the three most similar objects to the query point d_q by sending a query $Q = \langle 3, d_q \rangle$ to the broker. The broker will send an identical sub-query to each P_i to request each P_i to return its local top-3 similar objects to d_q. Once P_i finishes its local kNN computations, it will send the result set to the broker, i.e., $r_1 = \{\langle 8, d_1\rangle, \langle 9, d_2\rangle, \langle 11, d_3\rangle\}$, $r_2 = \{\langle 6, d_4\rangle, \langle 10, d_5\rangle, \langle 12, d_6\rangle\}$, and $r_3 = \{\langle 12, d_7\rangle, \langle 13, d_8\rangle, \langle 14, d_9\rangle\}$. Finally, the broker will use an oblivious priority queue to filter the final top three results, i.e., $r = \{\langle 6, d_4\rangle, \langle 8, d_1\rangle, \langle 9, d_2\rangle\}$.

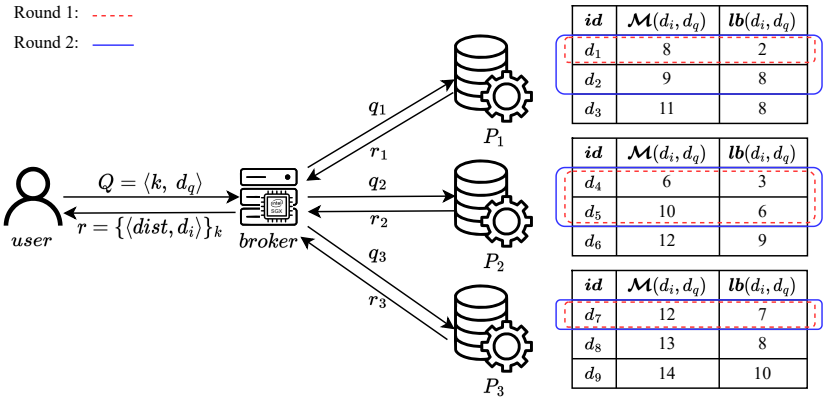


Fig. 2. An Example of FedKNN

Theorem 4.2. (Baseline cost). *For any arbitrary federated database $\mathcal{D}$, the baseline algorithm has a time cost of $O(m \cdot k)$.*[2]

Theorem 4.3. (Baseline security). *The baseline algorithm is fully oblivious for any two databases $\mathcal{D}_1$ and $\mathcal{D}_2$ with $|\mathcal{D}_1| = |\mathcal{D}_2|$.*

4.2 Distribution-Aware kNN Query (DANN)

From Theorem 4.2, we can find that if the computation cost of similarity metrics is high, the overhead of local computations can become the bottleneck, which will significantly affect the scalability of the system. In this subsection, we propose a novel federated kNN query algorithm, Distribution-Aware kNN (DANN), to address this bottleneck. DANN enhances the performance of federated kNN queries by reducing the unnecessary local kNN computations performed by each data provider P_i.

The core idea of DANN is to obtain the possible distribution of query results over the private dataset of each data provider using a fast preprocessing round. With the information on possible result distribution, the broker can prompt each P_i to compute their local kNNs with different k_i values separately, thus reducing local computations to enhance query performance. Algorithm 1 gives a detailed description of the query processing procedure. The broker starts by initializing two priority queues and a map (Lines 2-4). The priority queues track the query results and lower-bound distance estimations, whereas the map stores the NN counts for each data provider. The main procedure consists of three phases: (1) lower bound estimation, (2) first-round kNN computation, and (3) second-round kNN computation.

(1) Lower bound estimation. When the broker receives a new query $\langle k, d_q \rangle$, it sends the query to each P_i to quickly estimate a list of lower-bound distances. For each P_i, upon receiving $\langle k, d_q \rangle$ from the broker, it computes the top-k lower-bound distances of the query point d_q. The top-k lower-bound list $\{lb\}_i$ is then returned to the broker. After receiving $\{lb\}_i$ from each data provider (Lines 5-6), the broker uses a priority queue to filter the global top-k $\langle lb, i \rangle$ list (Line 7). Next, the broker will count the number of each P_i in the global top-k list as k_i (Lines 8-10), which is used to estimate the distribution of kNN results in $\mathcal{P}$. It should be noted that if P_i has no lower bound in the global top-k list, its k_i will be set to 1 by default. This is because P_i can still potentially have some

[2]Due to space limitations, the proofs of Theorem 4.2 and Theorem 4.3 are provided in the technical report [54].

Algorithm 1: Distribution-Awared kNN Query

```
1  Function DANN(k, d_q)
       Input: User query parameter k, query point d_q
       Output: Query result set
2      r ← PriorityQueue.Init(k);
3      lbQueue ← PriorityQueue.Init(k);
4      kMap ← Map.Init();
       // Phase 1: Lower bound estimation
5      for i ∈ {1, ··· , m} do
6          {lb}_i ← P_i.LocalKLB(k, d_q);
7          for lb ∈ {lb}_i do lbQueue.Enqueue(⟨lb, i⟩) ;
8      for i ∈ {1, ··· , m} do
9          k_i ← Max(|{⟨lb, j⟩ ∈ lbQueue ∧ i = j}|, 1);
10         kMap[i] ← k_i;
       // Phase 2: First-round kNN computation
11     for i ∈ {1, ··· , m} do
12         r_i ← P_i.LocalKNN(kMap[i], d_q);
13     for ⟨dist, d⟩ ∈ {r_i} do
14         r.Enqueue(⟨dist, d⟩);
15     for i ∈ {1, ··· , m} do
16         if r_i.last() ∈ r then
17             q_i ← r.GetIndex(r_i.last());
18             k'_i ← kMap[i] + k − q_i;
19             kMap[i] ← k'_i;
       // Phase 3: Second-round kNN computation
20     Repeat Lines 11-14;
21     return r;
```

query results, and setting k_i to 1 ensures that P_i will always perform some local kNN computations in the next phase.

Example 4.4. We still use Figure 2 as an example to illustrate the query procedure of DANN, where $Q = \langle 3, d_q \rangle$. For the lower bound estimation phase, the broker will send $\langle 3, d_q \rangle$ to the three data providers P_1, P_2, and P_3 to obtain their local top-3 lower-bound distances. After each P_i finishes its local lower-bound computation, it will return the low-bound list to the broker, i.e., $\{lb\}_1 = \{2, 8, 8\}$, $\{lb\}_2 = \{3, 6, 9\}$, and $\{lb\}_3 = \{7, 8, 10\}$. Then, the broker will use the priority queue to filter the global top-3 lower bounds, $lbQueue = \{\langle 2, 1\rangle, \langle 3, 2\rangle, \langle 6, 2\rangle\}$, and use them to estimate the first-round k_i for each P_i. Hence, we have $k_1 = 1$ and $k_2 = 2$. For P_3, there is no lower-bound value in $lbQueue$ that comes from $\{lb\}_3$, so $k_3 = 1$ by default.

(2) First-round kNN computation. After finishing the distribution estimation, the broker requests each P_i to return their top-k_i ($k_i \geq 1$) results (Lines 11-12). Once the first-round local kNN results are received from all data providers, the broker selects the top-k objects as candidate results (Lines 13-14). Since each data provider will provide at least one result and $k \leq \sum_i k_i < k + m$, the following lemma stands true.

Lemma 4.5. *For any arbitrary federated database $\mathcal{D}$, the first-round computation has a time cost of $O(k + m)$.*

The broker then iteratively checks the candidate results to obtain a new k_i' for each P_i, which will be used in the second-round local kNN computation. Specifically, the broker first checks whether the last data object in r_i returned by P_i is in the top-k candidate result set r (Line 16). If not, this means that P_i has returned enough local results to construct the final federated kNN results. Otherwise, P_i may have more results that need to be returned. In this case, the broker determines the position q_i of P_i's last data object in r. The maximum number of extra possible results that may contribute to the final query results is $k - q_i$. Thus, the second-round NN count k_i' of P_i will be updated to $k_i' = k_i + k - q_i$ (Lines 17-19).

Example 4.6. Continuing from Example 4.4, after obtaining the possible results' distribution from the lower bound estimation, the broker will next send a different sub-query q_i to each P_i based on k_i. Therefore, in the first-round local kNN computation, we get $r_1 = \{\langle 8, d_1\rangle\}$, $r_2 = \{\langle 6, d_4\rangle, \langle 10, d_5\rangle\}$, and $r_3 = \{\langle 12, d_7\rangle\}$. The broker then filters the set of candidate results $r = \{\langle 6, d_4\rangle, \langle 8, d_1\rangle, \langle 10, d_5\rangle\}$ for checking. In the checking step, for P_1, the last data object $\langle 8, d_1\rangle \in r$ and $q_1 = 2$, which implies that P_1 may still have at most one result, so its second-round NN count is updated to $k_1' = 2$. Similarly, for P_2, we have $\langle 10, d_5\rangle \in r$ and $q_2 = 3$. As $q_2 = 3$ is the last index of r, no extra results are needed from P_2, so k_2' remains as $k_2' = 2$. For P_3, as $\langle 12, d_7\rangle \notin r$, it means that P_3 has returned enough local results for constructing the final result set, so its second-round NN count keeps the same as the first-round NN count.

(3) Second-round kNN computation. Due to the imperfect lower-bound estimation, the first-round kNN computation may miss some results. Therefore, to ensure the correctness of kNN search results, the second-round kNN computation is needed to retrieve all potential candidates and obtain the final results. After checking the candidate results in the first round, the broker will send k_i' to each P_i for the second-round local kNN computation.[3] Upon receiving the second-round results from all data providers, the broker will select the top-k results as the final results of the federated kNN query (Line 20).

Lemma 4.7. *For any arbitrary federated database $\mathcal{D}$, assuming $k_i \geq \frac{k}{m}$ and the data objects are randomly distributed among all data providers, the second-round kNN computation has an expected time cost of $O(m^2)$.*

Proof. The additional cost equals the sum of $k - q_i$ for all data providers. To compute the expected value of q_i, we can model the problem: Given k integers from 1 to k and randomly selecting k_i integers from the list without duplication, what is the expected maximum integer selected?

Let j be the maximum integer, then there exist $C_{j-1}^{k_i-1}$ different combinations, and $Pr[max = j] = C_{j-1}^{k_i-1} \Big/ C_k^{k_i}$. Hence,

$$E[q_i] = \sum_{j=k_i}^{k} j \cdot Pr[max = j] = \sum_{j=k_i}^{k} j \cdot \frac{C_{j-1}^{k_i-1}}{C_k^{k_i}} = \frac{k_i}{k_i + 1} \cdot (k + 1)$$

and

$$E(\sum_i k_i') = \sum_i E(k - q_i) = \sum_i (\frac{k - k_i}{k_i + 1})$$

It is clear that if k_i becomes larger, $E(\sum_i k_i')$ becomes smaller. If $k_i \geq \frac{k}{m}$ for all i, then

$$E(\sum_i k_i') \leq \sum_i \frac{km - k}{k + m} < m \cdot (m - 1) = O(m^2)$$

[3]Note that each data provider can continue from the first round and perform the incremental computation to compute the top-k_i' results in the second round.

Then, the second-round computation is expected to have $O(m^2)$ additional cost. □

Although it is possible that $k_i < \frac{k}{m}$, we can slightly modify the first-round computation to return the top-$max(\frac{k}{m}, k_i)$ results. It adds only an additional constant factor to the first-round computation. In practice, the second-round computation cost would be much less than $O(m^2)$, as will be shown in the experiments (Figure 4(b)).

Combining Lemmas 4.5 and 4.7, we can obtain the theorem:

Theorem 4.8. (DANN cost). *For any arbitrary federated database $\mathcal{D}$, the DANN algorithm has an expected time cost of $O(k + m^2)$.*

Note that when $k \geq m^2$, our method is expected to have a time cost of $O(k)$, which is optimal.

Example 4.9. Continuing from Example 4.6, we continue the local kNN computation with k'_i for each P_i. The results are $r_1 = \{\langle 8, d_1\rangle, \langle 9, d_2\rangle\}$, $r_2 = \{\langle 6, d_4\rangle, \langle 10, d_5\rangle\}$, and $r_3 = \{\langle 12, d_7\rangle\}$. The broker then filters the final top-3 result set $r = \{\langle 6, d_4\rangle, \langle 8, d_1\rangle, \langle 9, d_2\rangle\}$.

Potential Information Leakage of DANN. Although DANN has expected optimal time complexity, it does not consider data privacy, which could potentially lead to information leakage. As we mentioned in Section 2.3, the adversaries in the protocol can be divided into two types: *curious data provider* (denoted as $\mathcal{A}_p$) and *curious broker* (denoted as $\mathcal{A}_b$). For $\mathcal{A}_p$, `LocalKLB` and `LocalKNN` are executed by the data providers in plaintext. For the lower-bound estimation, as the broker requests each P_i to return k lower-bound distances of the query point, $\mathcal{A}_p$ cannot acquire additional information through communication other than $\mathcal{L}$. For the two rounds of local kNN search, $\mathcal{A}_p$ can obtain the local NN counts k_i and k'_i, in addition to $\mathcal{L}$. This extra information gives $\mathcal{A}_p$ with an advantage in distinguishing between two distinct databases.

To $\mathcal{A}_b$, Algorithm 1 is executed by the broker in the enclave. $\mathcal{A}_b$ cannot obtain plaintext sensitive data during data processing or communication, which is guaranteed by the enclave. However, as we mentioned in Section 2.3, $\mathcal{A}_b$ can still observe the data access pattern during the processing of local top-k lower-bound distances and local kNN results.

Specifically, in the data processing stage, Algorithm 1 utilizes two priority queues, r and $lbQueue$, to store candidate results and lower-bound distances. Their access patterns can reveal the relative order and number of results from different data providers, thereby providing an additional advantage for $\mathcal{A}_b$ to distinguish between two distinct databases. For the update and access of $kMap$, in the lower-bound estimation phase, we traverse $i \in \{0...m\}$ for the update of k_i, so the access pattern of $kMap$ is solely determined by the number of data providers and is unrelated to specific data and query being processed. Therefore, the access and update of $kMap$ in the lower-bound estimation phase is oblivious. However, in the first-round kNN computation, as it is necessary to check whether P_i's last data object is in the candidate result set r to determine whether to update k_i, the memory access pattern of $kMap$ will be leaked. By observing which k_i in $kMap$ is updated, $\mathcal{A}_b$ can potentially infer the distribution of the final query results among the data providers.

During data transmission, the size of r_i sent back to the broker varies due to the differences in the local NN counts for each data provider. By observing the communication volume, $\mathcal{A}_b$ can infer the local NN counts k_i and k'_i. Consequently, this provides $\mathcal{A}_b$ with an additional advantage beyond $\mathcal{L}$.

In summary, during the execution of DANN, $\mathcal{A}_b$ can potentially obtain the order of results and the number of each data provider's results in the final result set r by observing the memory access pattern. Moreover, $\mathcal{A}_b$ can also learn each data provider's local NN counts k_i and k'_i from the network access pattern.

Algorithm 2: Secure Distribution-Awared kNN Query

1 **Function** DANN*$(k, d_q, \varepsilon, \lambda)$
  Input: User query parameter k, query point d_q, privacy budget ε, and max error probability λ
  Output: Query results r
2 $r \leftarrow$ ObliviousPriorityQueue.Init(k);
3 $lbQueue \leftarrow$ ObliviousPriorityQueue.Init(k);
4 $kMap \leftarrow$ Map.Init();
  // Phase 1: Lower bound estimation
5 **for** $i \in \{1, \cdots, m\}$ **do**
6 $\{lb\}_i \leftarrow P_i$.LocalKLB(k, d_q);
7 **for** $lb \in \{lb\}_i$ **do** $lbQueue$.Enqueue$(\langle lb, i\rangle)$;
8 **for** $i \in \{1, \cdots, m\}$ **do**
9 $k_i \leftarrow \mathsf{Max}(|\{\langle lb, j\rangle \in lbQueue \wedge i = j\}|, 1)$;
10 $kMap[i] \leftarrow k_i$;
  // Phase 2: First-round kNN computation
11 **for** $i \in \{1, \cdots, m\}$ **do**
12 $k_i \leftarrow kMap[i] - \frac{2}{\varepsilon}\ln(2 \cdot \lambda) + \mathsf{Lap}(\frac{2}{\varepsilon})$;
13 $k_i \leftarrow \mathsf{Min}(\mathsf{Max}(k_i, 1), k))$; // Make $1 \le k_i \le k$
14 $r_i \leftarrow P_i$.LocalKNNwithPadding(k_i, k, d_q);
15 **for** $\langle dist, d\rangle \in \{r_i\}$ **do**
16 r.Enqueue$(\langle dist, d\rangle)$;
17 **for** $i \in \{1, \cdots, m\}$ **do**
18 $isIn \leftarrow r_i$.last() $\in r$;
19 $q_i \leftarrow isIn \cdot r$.GetIndex($r_i$.last()) $+ \neg isIn \cdot k$;
20 $k'_i \leftarrow kMap[i] + k - q_i$;
21 $kMap[i] \leftarrow k'_i$;
  // Phase 3: Second-round kNN computation
22 Repeat Lines 11-16;
23 **return** r;

4.3 Secure Version of DANN (DANN*)

To mitigate the potential information leakage discussed above, we propose a secure version of DANN, called DANN*. In DANN*, we adopt (ε, δ)-differential obliviousness, which, as mentioned in Section 3.2, can maintain the algorithm's efficiency while providing a measurable level of privacy protection. This approach allows us to set a formal limit on the information leakage of the data providers.

DANN* requires some modifications to the DANN algorithm to meet the differential obliviousness requirement. We show DANN* in Algorithm 2 and highlight the changes in blue. For the input, DANN* requires two additional parameters, ε and λ, from the user. For these two parameters, ε means how many advantages the adversary may gain from the access pattern, and λ is the maximum probability that the final kNN results contain false objects due to the noise.

We make the following modifications to prevent the broker from inferring sensitive information about the data providers from the memory access pattern. First, we replace the priority queues r and $lbQueue$ with an oblivious version introduced in Section 3.3 (Lines 2-3). Second, to address the issue of memory access pattern leakage resulting from candidate result checking during the first-round kNN computation, we make the process oblivious (Lines 17-21). Specifically, we use boolean operations to replace conditional branches (Lines 18-19), to calculate the second-round NN

count and update all values in $kMap$. This only increases the multiplication operation cost at most $2 \cdot m$ times and the $kMap$ update overhead $m - 1$ times. As a result, our control flow of updating $kMap$ during the checking step relates only to the number of data providers m.

For the curious data provider, it may deduce the data distributions of other data providers by analyzing the local NN counts k_i and k_i' during query processing. To counter this threat, it is necessary to ensure that the local NN counts k_i and k_i' satisfy differential privacy for each P_i, in order to achieve differential obliviousness. Meanwhile, as k_i and k_i' are positive integers, a Laplace noise with $\frac{\varepsilon}{2}$ privacy budget and an offset $-\frac{2}{\varepsilon}\ln(2 \cdot \lambda)$ are added to k_i and k_i' (Line 12). The offset guarantees that the added noise is positive with probability $1 - \lambda$.

Definition 4.10 (Laplace mechanism $\mathcal{F}(\mathcal{D}, f, \varepsilon)$). Given a function $f : \mathcal{D} \mapsto \mathbb{R}^*$, we add a non-negative noise ξ drawn from the Laplace distribution $Lap(\frac{\Delta f}{\varepsilon})$ to the output $f(\mathcal{D})$. Formally, the Laplace mechanism outputs the following:

$$\mathcal{F}(\mathcal{D}, f, \varepsilon) = f(\mathcal{D}) + \xi \tag{4}$$

where $\xi \sim Lap(\frac{\Delta f}{\varepsilon})$ and Δf is the sensitivity of function f.

To prevent adversaries from inferring the number of results returned by each P_i by observing the network access pattern, we add dummy results to pad the result set from each P_i to a uniform length of k (Line 14) before transmitting to the broker. This ensures that the network access pattern remains invariant, regardless of the underlying databases.

With the above modifications, DANN* can achieve differential obliviousness for all participants in F, which we will prove in Section 4.4. Meanwhile, DANN* still keeps the efficiency of DANN for handling federated kNN queries. We will show it in Section 6.

Theorem 4.11. (DANN* cost). *For any arbitrary federated database $\mathcal{D}$, the DANN* algorithm has an expected time cost $O(k + m^2 - \frac{4}{\varepsilon} \cdot \ln(2 \cdot \lambda))$.*

Proof. As DANN* uses Laplace noise to maintain the differential privacy of each round's local NN count, the expectation of the Laplace noise is 0. To bound the error, each NN count is offset by $-\frac{2}{\varepsilon} \cdot \ln(2 \cdot \lambda)$. Therefore, given Theorem 4.8, the total number of local kNN computations required for the two rounds is bounded by $O(k + m^2 - \frac{4}{\varepsilon} \cdot \ln(2 \cdot \lambda))$. □

4.4 Security Analysis of DANN*

We now prove that DANN* satisfies (ε, δ)-differential obliviousness.

Definition 4.12 (Sensitivity of k_i). The sensitivity of the function $f : \mathcal{D} \mapsto \mathbb{R}^*$ is the maximum difference of the adversary's view in k_i of neighboring databases $\mathcal{D}_1$ and $\mathcal{D}_2$. Formally, it is defined as:

$$\Delta f = \max_{\|\mathcal{D}_1 - \mathcal{D}_2\| = 1} \|f(\mathcal{D}_1) - f(\mathcal{D}_2)\|_1 \tag{5}$$

As k_i is the local NN count of P_i, it is not hard to have $\Delta f = 1$ in our algorithm of two neighboring databases.

Lemma 4.13. *For any neighboring databases $\mathcal{D}_1$ and $\mathcal{D}_2$, the Laplace mechanism satisfies $(\varepsilon, 0)$-differential privacy.*

Proof. Let $p_{\mathcal{D}_1}$ and $p_{\mathcal{D}_2}$ denote the probability density functions of $\mathcal{F}(\mathcal{D}_1, f, \varepsilon)$ and $\mathcal{F}(\mathcal{D}_2, f, \varepsilon)$, respectively. Thus, $\mathcal{F}(\mathcal{D}_1, f, \varepsilon) = f(\mathcal{D}_1) + \xi_1$ and $\mathcal{F}(\mathcal{D}_2, f, \varepsilon) = f(\mathcal{D}_2) + \xi_2$. Then, for any output $z \in \mathbb{R}^*$, we have:

$$\begin{aligned}
\frac{p_{\mathcal{D}_1}(z)}{p_{\mathcal{D}_2}(z)} &= \frac{Pr[f(\mathcal{D}_1)+\xi_1=z]}{Pr[f(\mathcal{D}_2)+\xi_2=z]} = \frac{Pr[\xi_1=z-f(\mathcal{D}_1)]}{Pr[\xi_2=z-f(\mathcal{D}_2)]} \\
&= \frac{\frac{\varepsilon}{2\Delta f}\cdot e^{-\frac{\varepsilon}{\Delta f}(|z-f(\mathcal{D}_1)|)}}{\frac{\varepsilon}{2\Delta f}\cdot e^{-\frac{\varepsilon}{\Delta f}(|z-f(\mathcal{D}_2)|)}} \\
&= e^{\frac{\varepsilon}{\Delta f}(|z-f(\mathcal{D}_2)|-|z-f(\mathcal{D}_1)|)} \\
&\leq e^{\frac{\varepsilon}{\Delta f}(|f(\mathcal{D}_1)-f(\mathcal{D}_2)|)} \\
&\leq e^{\frac{\varepsilon}{\Delta f}\cdot\Delta f} = e^{\varepsilon}
\end{aligned}$$

where the inequality follows from the triangle inequality and the last simplification of the equation follows from Definition 4.12. Therefore, the mechanism satisfies $(\varepsilon, 0)$-differential privacy. □

Theorem 4.14. *For any neighboring databases $\mathcal{D}_1$ and $\mathcal{D}_2$, DANN* achieves full obliviousness with respect to $\mathcal{A}_b$.*

Proof. This claim directly stems from the obliviousness claim of the oblivious priority queue. By employing the oblivious priority queue, altering the program flow to depend solely on the number of data providers, and padding the data transmission in DANN*, $\mathcal{A}_b$ cannot derive additional significant advantages via the access pattern. Hence, we have:

$$\Pr[\mathbf{Access}^{\Pi}(\mathcal{D}_1) \in S] \leq Pr[\mathbf{Access}^{\Pi}(\mathcal{D}_2) \in S] + \delta$$

Here, δ accounts for a negligible probability that stems from the fully oblivious operations and S represents any subset of possible memory access patterns. Therefore, for any federated databases $\mathcal{D}_1$ and $\mathcal{D}_2$ of the same size, we have:

$$Pr[Sim^{\Pi}(\mathcal{L}, \mathbf{Access}^{\Pi}(\mathcal{D}_1), \mathcal{D}_1) = 1] \leq Pr[Sim^{\Pi}(\mathcal{L}, \mathbf{Access}^{\Pi}(\mathcal{D}_2), \mathcal{D}_1) = 1] + \delta$$

which meets the requirement of Definition 3.2. As such, DANN* achieves full obliviousness with respect to $\mathcal{A}_b$. □

Theorem 4.15. *For any neighboring databases $\mathcal{D}_1$ and $\mathcal{D}_2$, DANN* achieves $(\varepsilon, 0)$-differential obliviousness with respect to $\mathcal{A}_p$.*

Proof. As mentioned above, $\mathcal{A}_p$ can know its k_i and k_i' in both rounds. Here k_i and k_i' can be considered a counting query, and the sensitivity of a counting query is 1 (a single element addition or deletion can affect at most one count). Lemma 4.13 suggests a $(\frac{\varepsilon}{2}, 0)$-differential privacy counting query can be achieved by adding noise with the parameter of $\frac{2}{\varepsilon}$. Two rounds of k_i and k_i' can be considered a combination of two counting queries. With Lemma 4.13, we have:

$$\frac{p_{\mathcal{D}_1}(k_i)}{p_{\mathcal{D}_2}(k_i)} \cdot \frac{p_{\mathcal{D}_1}(k_i')}{p_{\mathcal{D}_2}(k_i')} \leq e^{\frac{\varepsilon}{2}} \cdot e^{\frac{\varepsilon}{2}} = e^{\varepsilon}$$

So the access pattern revealed to $\mathcal{A}_p$ has:

$$Pr[\mathbf{Access}^{\Pi}(\mathcal{D}_1) \in S] \leq e^{\varepsilon} Pr[\mathbf{Access}^{\Pi}(\mathcal{D}_2) \in S]$$

Therefore, we have:

$$Pr[Sim^{\Pi}(\mathcal{L}, \mathbf{Access}^{\Pi}(\mathcal{D}_1), \mathcal{D}_1) = 1] \leq e^{\varepsilon} Pr[Sim^{\Pi}(\mathcal{L}, \mathbf{Access}^{\Pi}(\mathcal{D}_2), \mathcal{D}_1) = 1]$$

which meets the requirement of Definition 3.3. Hence, DANN* achieves $(\varepsilon, 0)$-differential obliviousness with respect to $\mathcal{A}_p$. □

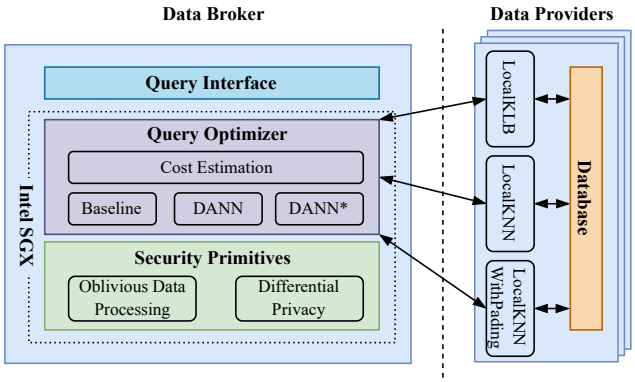


Fig. 3. Architecture of FedKNN

Theorem 4.16. *(DANN* security) DANN* is (ε, δ)-differentially oblivious for any neighboring databases $\mathcal{D}_1$ and $\mathcal{D}_2$.*

Proof. With Theorem 4.14, DANN* achieves full obliviousness with respect to $\mathcal{A}_b$; and with Theorem 4.15, DANN* achieves $(\varepsilon, 0)$-differential obliviousness with respect to $\mathcal{A}_p$. Therefore, for any adversary, we have:

$$Pr[Sim^{\Pi}(\mathcal{L}, \textbf{Access}^{\Pi}(\mathcal{D}_1), \mathcal{D}_1) = 1] \leq e^{\varepsilon} Pr[Sim^{\Pi}(\mathcal{L}, \textbf{Access}^{\Pi}(\mathcal{D}_2), \mathcal{D}_1) = 1] + \delta$$

so DANN* is (ε, δ)-differentially oblivious. □

5 IMPLEMENTATION

We have implemented a prototype system for FedKNN that incorporates our proposed algorithms, as illustrated in Figure 3. The system was built using the Rust programming language on the Linux platform. FedKNN consists of a broker and multiple data providers.

Broker Implementation. The broker side of FedKNN comprises three main components: a query interface, a query optimizer, and security primitives. The query interface serves as the interaction point between FedKNN and users. It is responsible for processing and forwarding user query requests, as well as returning query results to users upon completion. The query optimizer encompasses cost estimation and the three kNN algorithms proposed in Section 4. Cost estimation consists of local kNN estimation and network estimation. The former measures each data provider's local kNN computation time using sample queries. In contrast, the latter measures the network latency and transmission speed between the broker and data providers. Based on the cost estimation, FedKNN automatically selects the baseline algorithm when the local kNN computation cost is lower than the communication cost (e.g., when using Euclidean distance for spatial data), since the baseline algorithm minimizes communication costs. Otherwise, if the local computation cost is higher than the communication cost (e.g., when using the graph edit distance for graph data), FedKNN selects either the DANN or DANN* algorithm to process queries, based on whether the user desires to protect data access patterns. DANN addresses the inefficiency of the baseline and offers a secure solution against the first threat mentioned in Section 2.3, except for access pattern attacks. On the other hand, DANN* is specifically designed to counter access pattern attacks. In scenarios where access pattern threats are not a concern, DANN is preferable due to its superior query performance compared to DANN*. Finally, the security primitives provide oblivious data processing and differential privacy, which are used during secure computation in FedKNN. Both the

Table 3. Statistics of Datasets

Dataset	$\|D\|$	Data Type	$\mathcal{M}(x, y)$
AIDS	42,689	Graph	Graph Edit Distance
SYN	1,000,000	Graph	Graph Edit Distance
DBLP	6,553,812	Sequence	Edit Distance
GENOME	320,000	Sequence	Edit Distance
OSM	1,189,761	Spatial	Euclidean Distance

FedKNN algorithms and the security primitives components operate within the Intel SGX enclave to ensure the integrity and prevent privacy leakage during query execution.

Data Provider Implementation. On the data provider's side, FedKNN interacts with the underlying database through three APIs, i.e., LocalKLB, LocalKNN, and LocalKNNwithPadding, to execute FedKNN's queries. FedKNN does not make any assumptions about the databases used by data providers, allowing it to support heterogeneous databases and various data types.

6 EVALUATION

6.1 Experimental Setup

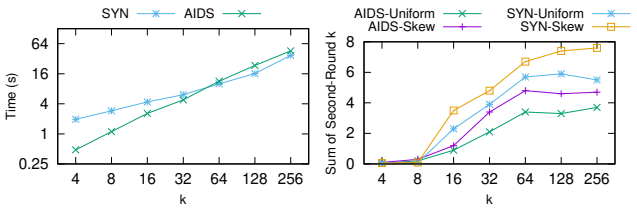
Datasets and Query Workload. We conduct experiments on three different types of data: graph, sequence, and spatial data. Graph and sequence data represent the data types where local kNN computation costs are dominant, while spatial data represents the data type where communication costs are dominant. We use two datasets for graph data: a real-world dataset AIDS and a synthetic graph dataset SYN. We use two real-world datasets for sequence data: DBLP and GENOME. For spatial data, we use OpenStreetMap (OSM) data as our experiment dataset. Table 3 provides the statistical information of all datasets. Below are detailed descriptions of each dataset:

- AIDS: An open NCI database of antivirus screening chemical compounds published by the Developmental Therapeutics Program in October 1999.[4] This dataset is widely used by previous graph similarity search works [9, 34].
- SYN: A synthetic dataset generated by GraphGen.[5] To evaluate the scalability of different algorithms under various scales, we sampled subsets under different ratios: 20%, 40%, 60%, 80%, and 100%. We use 60% as the default.
- DBLP: A dataset containing a bibliographic collection of computer science research publications, which include information such as authors, titles, and publishing details.[6]
- GENOME: A dataset consisting of large-scale genome sequences obtained from Chromosome 20 of 50 individuals from the Personal Genomes Project.[7]
- OSM: A dataset commonly used for spatial analysis [16, 41]. For our experiment, we extracted all locations of points of interest from the OpenStreetMap in East Asia.

For all datasets, we employ a Zipf distribution to distribute the data among the data providers to form the data federation. As for the query workload, we randomly sample 50 data objects from each dataset to serve as query data objects.

Metrics. We evaluate the query processing efficiency in terms of the average running time of the query workload, which is defined as the latency from when the broker receives the query from a user to when it returns the query results to the user.

[4]https://cactus.nci.nih.gov/download/nci/AID2DA99.sdz
[5]https://www.cse.cuhk.edu.hk/~jcheng/graphgen1.0.zip
[6]https://dblp.uni-trier.de/db/
[7]https://github.com/kedayuge/Search



(a) kNN Cost w.r.t. k (b) DANN Second-Round $\Sigma k_i'$

Fig. 4. Preliminary Results w.r.t. Local kNN Cost and Lemma 4.7

Algorithms for Evaluation. We compare our three algorithms in FedKNN with the state-of-the-art algorithm Hu-Fu [41], which is a secure multiparty computation (SMC)-based system for federated spatial queries. Hu-Fu tackles the federated kNN queries by transforming them into multiple rounds of range data counting, to search an appropriate range and then retrieve data objects within that range. To ensure a fair comparison, we implemented and extended the kNN algorithm of Hu-Fu in Rust to support graph data and sequence data. We also replaced SMC with Intel SGX to perform secure counting and set union operations. We refer to this algorithm as HuFu-Ext. It is important to note that HuFu-Ext offers weaker security compared to differential obliviousness, as it discloses the distances associated with kNN searches in each round. In contrast, in our FedKNN, the baseline offers full obliviousness and DANN* offers differential obliviousness.

Regarding kNN search, we use the library[8] from the state-of-the-art method [9] for graph similarity computation and extend it to support local graph kNN search. For sequence data, we adopt the source code of [53] as the local computation algorithm, which is a state-of-the-art approach for sequence kNN search. For spatial data, we use PostGIS[9] as the underlying kNN search engine. Regarding the lower-bound estimation in DANN and DANN*, we employ different algorithms based on the type of data. Specifically, we use the label-based filtering algorithm [9] for graph data, the q-gram-based filtering algorithm [53] for sequence data, and the R-tree for spatial data.

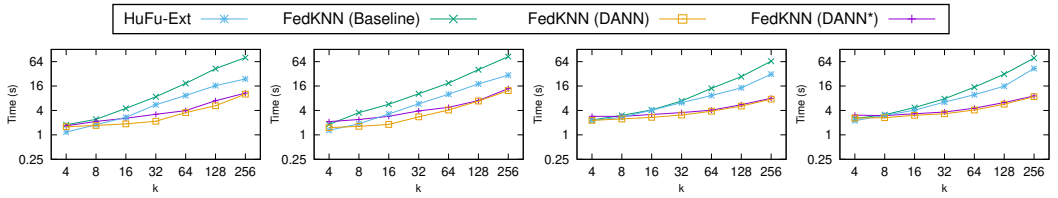
Environment. We run the broker on an Intel SGX-enabled Azure Standard_DC4s_v3 instance with four CPU cores and 32GB RAM, which is deployed in the East US region. Each data provider P_i runs on an Azure Standard_E4ds_v4 instance deployed in the West US region that has 4-vCPU and 32GB RAM. The broker and data providers are connected via a wide-area network and the average latency is 64ms.

6.2 Preliminary Results

In our first set of experiments, we aim to validate the local kNN computation cost and Lemma 4.7. Specifically, we use the AIDS dataset and 60% of the SYN dataset and run the local kNN algorithm on a single server.

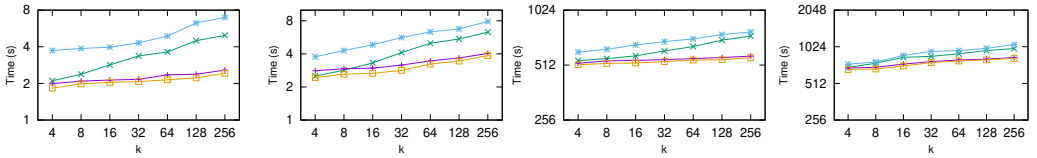
In the local kNN cost experiment (Figure 4(a)), we can observe that the cost of kNN search is generally proportional to the value of k. When k varies from 4 to 256, the running time increases 95× on AIDS and 19× on SYN. In the experiment of verifying Lemma 4.7, we set m to 8 and the Zipf factor to 0.0 and 0.6 to simulate uniformly distributed data and skewed data, respectively. As shown in Figure 4(b), the second-round $\sum_i k_i'$ is much smaller than the expected value of $O(m^2)$ (i.e., 64). Moreover, even when k is large, $\sum_i k_i'$ remains small compared with k.

[8]https://github.com/LijunChang/Graph_Edit_Distance
[9]https://www.postgis.org/



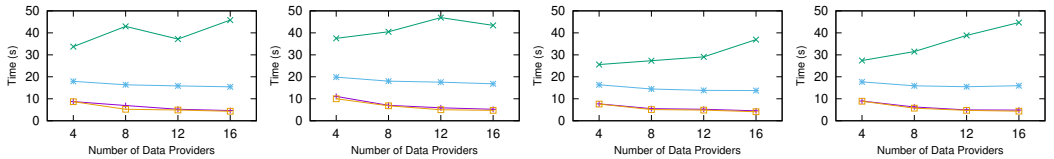
(a) Uniform (AIDS) (b) Skew (AIDS) (c) Uniform (SYN) (d) Skew (SYN)

Fig. 5. Running Time of Varying k on Graph Data



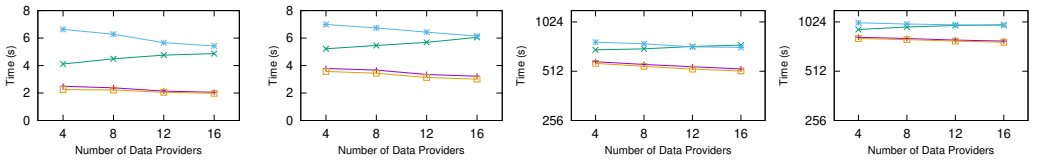
(a) Uniform (DBLP) (b) Skew (DBLP) (c) Uniform (GENOME) (d) Skew (GENOME)

Fig. 6. Running Time of Varying k on Sequence Data



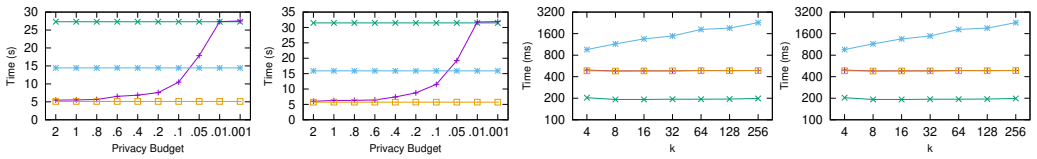
(a) Uniform (AIDS) (b) Skew (AIDS) (c) Uniform (SYN) (d) Skew (SYN)

Fig. 7. Running Time of Varying m on Graph Data



(a) Uniform (DBLP) (b) Skew (DBLP) (c) Uniform (GENOME) (d) Skew (GENOME)

Fig. 8. Running Time of Varying m on Sequence Data



(a) Uniform (SYN) (b) Skew (SYN)

Fig. 9. Running Time of Varying ε

(a) Uniform (OSM) (b) Skew (OSM)

Fig. 10. Running Time of Sptial kNN Query

6.3 Performance of kNN Algorithms

In this section, we conduct experiments to compare the performance of the kNN algorithms on the four datasets where the local computation cost dominates. We also perform experiments using the OSM dataset to evaluate the performance when the communication cost dominates. For all algorithms, unless otherwise specified, we set k to 128, m to 8, and the Zipf distribution factor to 0.0 and 0.6 to simulate uniformly distributed data and skewed data, respectively. For DANN*, we set ε to 1.0 and λ to 0.05 by default.

Impact of varying k. We vary k from 4 to 256 and show the running time of the four algorithms in Figure 5 and Figure 6. For the two uniformly distributed graph datasets shown in Figure 5, DANN*

is up to 7.5× and 2.3× faster than the baseline and HuFu-Ext on AIDS. Similarly, it is up to 8.2× and 3.9× faster than the baseline and HuFu-Ext on SYN. In the case of skewed distribution, DANN* achieves a similar performance improvement, being 6.1× and 2.1× faster than the baseline and HuFu-Ext on AIDS. On the larger dataset SYN, DANN* exhibits even greater performance gains, being 8.6× and 4.8× faster than the baseline and HuFu-Ext.

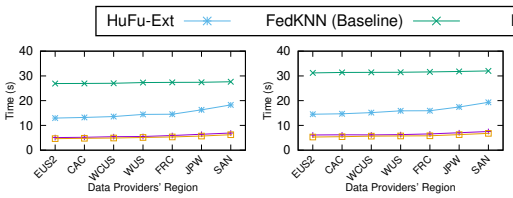
We can also make two interesting observations. First, DANN* starts outperforming HuFu-Ext when $k \geq 16$ on both datasets and distributions. This is because DANN* requires at least one data object per data provider, so the minimum number of kNN computations is related to the number of data providers m. In the experimental setup where $m = 8$, DANN* does not have an advantage for $k = 4$ and $k = 8$. Instead, HuFu-Ext converts the kNN query into multiple rounds of data counting to find the appropriate query range, reducing the local computation overhead for the data providers that do not need to provide data. As a result, HuFu-Ext outperforms DANN* when $k \leq m$. Second, as k increases, the running time of DANN* gets closer to that of the non-secure DANN algorithm. This is because the noise and the noise offset have a higher impact on the local NN count of DANN* when k is small, but this impact is mitigated as k grows.

The results for sequence data are similar to those for graph data, with DANN* consistently outperforming the baseline and HuFu-Ext. Specifically, on DBLP and GENOME, DANN* achieves a 1.9× and 29% speedup over the baseline and a 2.7× and 36% speedup over HuFu-Ext, as shown in Figure 6. We also observe that HuFu-Ext underperforms the baseline for sequence data, due to its large search space for finding a suitable range boundary, which incurs an excessive local computation overhead and slows down the query. On the other hand, the limited performance of the lower-bound distance estimation algorithm [53] employed for sequence data restricts DANN*'s ability to accurately predict the distribution of kNN results, which reduces DANN*'s advantage over the baseline algorithm.

Impact of varying m. We vary the number of data providers m from 4 to 16 to evaluate the scalability of the algorithms with respect to multiple data providers. For graph data, we show the results in Figure 7. On AIDS, DANN* is up to 9.8× and 8.3× faster than baseline and 3.1× and 3.2× faster than HuFu-Ext as m increases. On SYN, DANN* is up to 8.2× and 9.3× faster than the baseline and 3.1× and 3.3× faster than HuFu-Ext. We can observe that the running time of both DANN* and HuFu-Ext decreases as m increases. For DANN*, this is because the time complexity is independent of m. Therefore, increasing m reduces the amount of data that each data provider needs to process when the total amount of data is fixed. This leads to faster local kNN computation, reducing the overall query time. For HuFu-Ext, this is because it searches the kNN range in a global view, and m does not affect the value of the kNN range. As a result, the reduced data on each data provider improves the query performance. However, the baseline algorithm's total local computation is linear to m. Hence, as m grows, the overall computation increases, which may result in local kNN computations involving more complex data objects that are not in the final results. This can cause the running time to increase.

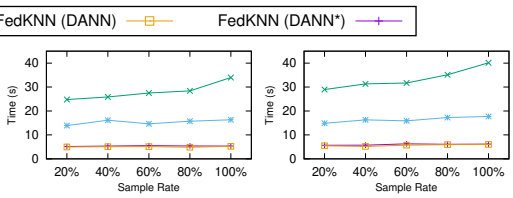
For sequence data, we observe similar results as for graph data, as shown in Figure 8. For DBLP, when increasing m, DANN* achieves up to a 2.4× and 1.9× speedup over the baseline and a 2.7× and 1.9× speedup over HuFu-Ext. A similar trend is found for the GENOME dataset. Under a uniform distribution, we achieve up to a 39% and 35% speedup over the baseline and HuFu-Ext, respectively. Under a skewed distribution, we achieve up to 26% and 25% speedup over the baseline and HuFu-Ext, respectively. For both types of data, we confirm that DANN* offers better performance when the number of data providers increases.

Impact of varying ε **on DANN***. In this experiment, we vary the privacy budget ε from 0.001 to 2.0 to test the effect of different ε settings on our DANN* algorithm. We use DANN as the performance



(a) Uniform (SYN) (b) Skew (SYN)

Fig. 11. Network Latency Test



(a) Uniform (SYN) (b) Skew (SYN)

Fig. 12. Scalability Test

lower bound and the baseline as the upper bound. We conducted experiments on SYN. The results are shown in Figure 9. With a large privacy budget ($\varepsilon \geq 0.8$), DANN* is essentially in line with the efficiency of the non-secure DANN algorithm, as the additional noise and noise offset only added less than 11% overhead to the computation. With a lower privacy budget ($0.8 > \varepsilon \geq 0.1$), DANN* still maintains high efficiency compared to the baseline and HuFu-Ext on both data distributions. When the privacy budget is further reduced, DANN* begins to approach the baseline and coincides with it when $\varepsilon = 0.01$, which means DANN* is fully oblivious at that point. This experiment demonstrates that DANN* can maintain its efficiency with low privacy budgets, and it outperforms other baselines even when the privacy budget is as low as $\varepsilon = 0.1$.

Impact of local kNN computation cost. We also test the performance of the kNN algorithms on spatial data where the communication cost dominates. The results are shown in Figure 10. The baseline algorithm has the best performance, up to 2.5× and 11.9× faster than DANN* and HuFu-Ext, respectively. This is because the baseline algorithm only requires one round of communications to complete the query. In contrast, DANN and DANN* require three rounds of communications, which makes their performance worse than the baseline. On the other hand, HuFu-Ext has the worst performance because it needs to find the appropriate search range to complete the kNN query through binary search, which incurs high communication costs. This experiment also highlights the importance of the cost estimation module in the FedKNN query optimizer, which can improve the performance of our system's kNN queries for different types of data.

Impact of network latency. To assess the impact of WAN distance and network latency on algorithm performance, we perform tests using data providers located in seven Azure regions: East US 2 (EUS2), Canada Central (CAC), West Central US (WCUS), West US (WUS), France Central (FRC), Japan West (JPW), and South Africa North (SAN). The latency between the data providers located in these regions and the broker deployed in East US ranges from 7ms to 243ms.[10] The evaluation results are shown in Figure 11. In the case of a uniform distribution, as the network latency increases, the query time of the baseline, HuFu-Ext, DANN, and DANN* algorithms increases by 3%, 41%, 34%, and 36%, respectively. For the skewed distribution, the query time increases by 3% for the baseline, 33% for HuFu-Ext, 28% for DANN, and 23% for DANN*. It is important to note that the baseline algorithm, which requires only one communication round, is the least affected by latency. HuFu-Ext is the most sensitive to latency due to its multi-round range query transformation. DANN and DANN*, with three rounds of communication, are also impacted by latency but remain faster than both the baseline and HuFu-Ext in all cases tested.

Breakdown time of algorithms. We further conduct experiments to analyze the time taken by various components of each algorithm using the SYN dataset. As shown in Table 4, it is clear that the local kNN search is the most time-intensive component for all algorithms, significantly dominating the overall query time. Specifically, the local kNN search accounts for over 99% and

[10]https://learn.microsoft.com/azure/networking/azure-network-latency

Table 4. Breakdown of Running Time (s) on SYN

Algorithm	Uniform				Skew			
	Network	Local Search	LB	Aggregation	Network	Local Search	LB	Aggregation
HuFu-Ext	0.76	13.69	\	0.01	0.84	15.04	\	0.01
Baseline	0.15	27.13	\	0.03	0.14	31.31	\	0.03
DANN	0.43	4.57	0.08	0.04	0.46	5.13	0.10	0.04
DANN*	0.45	4.95	0.08	0.04	0.45	5.70	0.09	0.04

94% of the query time for the baseline and HuFu-Ext algorithms, respectively. The time spent on network and broker aggregation is insignificant in both algorithms. In contrast, both DANN and DANN* effectively reduce the time required for the local kNN search by 63%–83% through the use of a lower-bound estimation phase. Furthermore, the estimation process incurs only a very small overhead, representing less than 2% of the total query time. Consequently, the overall query time of DANN and DANN* is greatly improved compared to the baseline and HuFu-Ext algorithms.

6.4 Performance on Scalability Test

To evaluate the performance of all algorithms on large-scale data, we vary the sample ratio of SYN from 20% to 100%. Other parameters remain the same as in Section 6.3 and Figure 12 shows the results. Similar to the previous results, we observe that under the same dataset size, DANN* is significantly better than the baseline and HuFu-Ext. When the data is uniformly distributed, DANN* is up to 6.3× and 3.1× faster than the baseline and HuFu-Ext. DANN* is up to 6.5× and 2.9× faster when the data distribution is skewed.

It is also observed that DANN* and HuFu-Ext are not sensitive to the growth of the data size. DANN*'s running time is increased by only 11% and 16% as the data volume of SYN grows from 200,000 to 1,000,000, while HuFu-Ext is increased by 17% and 19%, respectively. This is because both algorithms tend to search the results of kNN queries in a small search space, and hence the data size does not significantly affect their efficiency. In contrast, for the baseline algorithm, the growth in data size leads to a more significant reduction in efficiency. For example, when the data size grows from 200,000 to 1,000,000, the running time of the baseline increases by 37% and 39% for the two data distributions, respectively. This is because the baseline requires the same number of local results from each data provider, so as the data size increases, the local kNN computation time also increases. This can affect query efficiency, making it slower as the data size grows.

6.5 Ablation Studies of FedKNN

Finally, we conduct ablation studies to examine which component contributes to the performance improvement of DANN and DANN* in federated kNN queries. By setting all return values of the lower-bound distance to infinity, we simulate the scenario where our DANN and DANN* algorithms remove lower-bound estimation. We denote these modified algorithms as DANN (No LB) and DANN* (No LB), respectively.

We evaluate the performance on four skewed datasets where local computation costs dominate. The results are shown in Figure 13. We observe that on the graph datasets AIDS and SYN, the use of lower-bound estimation significantly reduces the query time of DANN and DANN*, achieving savings of up to 81% and 78%, respectively. When it comes to sequence data, the lower-bound distance estimation algorithm is less effective, as mentioned in Section 6.3. Consequently, the impact of using lower-bound estimation on the query performance of our algorithms is diminished. We achieve savings of 42% and 40% on DBLP, and 15% and 14% on GENOME. Conversely, without the use of lower-bound estimation, DANN (No LB) and DANN* (No LB) perform similarly or even

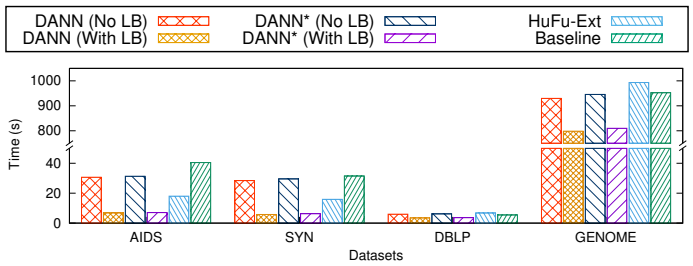


Fig. 13. Ablation Studies of FedKNN on Four Datasets

worse than HuFu-Ext. This highlights the critical role of lower-bound estimation in reducing local kNN computations and improving the query performance of our algorithms.

7 RELATED WORKS

Secure Federated Analytics. Federated data analytics facilitates collaboration among mutually distrusting participants, allowing them to use their combined data to answer queries. A series of related works have explored secure federated analytics for relational databases [3, 4, 13, 24, 35, 42, 55] and spatial databases [41]. Some works [3, 4, 24, 35, 41, 42] based on secure multi-party computation (SMC). Specifically, SMCQL [3] translates SQL queries into garbled circuits for two-party relational database queries. Conclave [42], built on OblivC [51], supports secure querying among up to three parties. Senate [35] and Scape [24] address the threat of malicious participants in data federations. SAQE [4] tackles approximate query problems by combining SMC and differential privacy, achieving a balance between performance and accuracy. SECRECY [32] considers the scenario of outsourcing SQL query computations. Hu-Fu [41] optimizes federated spatial queries by decomposing them into plaintext and secure operators. On the other hand, some other works leverage hardware enclaves (e.g., Intel SGX) [13, 55]. Despite the encryption of data within enclaves, hardware enclaves cannot protect access patterns. To tackle this problem, Opaque [55] and OCQ [13] employ oblivious algorithms within enclaves to achieve oblivious cooperative analytics. Another related topic is federated kNN learning [15, 33, 37]. However, these works are not directly comparable to our study, due to the different query requirements (exact search vs. approximate search [15]), data partition schemes (horizontal vs. vertical [33]), and computation paradigms (kNN search vs. NN classification [37]).

Privacy-Preserving Queries over Outsourced Data. Privacy-preserving queries over outsourced data constitute another pertinent field of related works [10, 18, 23, 29, 31, 39, 43–45]. Numerous practical strategies have been developed, including SMC, searchable encryption [40], and homomorphic encryption [19], among others. While these techniques can be adapted to the federated setting, their inherent focus on strong security guarantees and the limitation of the number of participants might negatively impact performance. To enhance utilization, the concept of differential obliviousness is proposed in [8], which requires only the access patterns during query processing to satisfy differential privacy, Some works on relational databases have adopted this model [36, 46].

Differentially Private Padding Mechanism. Adding positive differential privacy noise is essential in many application scenarios, including private set intersection [22, 25] and database queries [6]. The state-of-the-art mechanism for positive differentially private padding is the improved Bayesian-update mechanism proposed in [22]. The core idea is to incorporate prior knowledge of the actual result distribution to minimize the offset caused by the simple Laplace mechanism. Nevertheless, the simple Laplace mechanism continues to be widely used [6, 25], as the prior knowledge required

by [22] is unattainable in many scenarios. Similarly, in the context of federated kNN queries, it is not possible to obtain the distribution of actual results beforehand.

8 CONCLUSION

In this paper, we have proposed FedKNN, an efficient and secure system for performing kNN queries on federated databases. Our proposed Distribution-Aware kNN (DANN) algorithm minimizes local kNN computations by estimating the distribution of query results across data providers through a fast pre-processing round. We have also developed a secure version of DANN, called DANN*, which satisfies differential obliviousness to ensure the privacy of data providers. Moreover, we have implemented a prototype system for FedKNN that incorporates our proposed algorithms. Our evaluation shows that FedKNN outperforms current state-of-the-art solutions, achieving up to 9.8× and 4.8× improvement on federated graph kNN search compared to the baseline and the state-of-the-art solution, respectively, and up to 2.4× and 2.7× improvement on federated sequence kNN search. We believe that our work provides a valuable contribution to the field of federated kNN search.

ACKNOWLEDGMENTS

This work is supported by Hong Kong RGC Grants (Project No. 12202221 and C2004-21GF), NSF of China (Project No. 62372128 and 62072132), and NSF of Guangdong Province (Project No. 2023A1515030273). Jianliang Xu is the corresponding author.

REFERENCES

[1] 13th National People's Congress of the People's Republic of China. 2021. Personal Information Protection Law of the People's Republic of China. http://en.npc.gov.cn.cdurl.cn/2021-12/29/c_694559.htm.
[2] Arvind Arasu and Raghav Kaushik. 2013. Oblivious query processing. *Preprint arXiv* (2013).
[3] Johes Bater, Gregory Elliott, Craig Eggen, Satyender Goel, Abel N Kho, and Jennie Rogers. 2017. SMCQL: Secure Query Processing for Private Data Networks. *PVLDB* (2017), 673–684.
[4] Johes Bater, Yongjoo Park, Xi He, Xiao Wang, and Jennie Rogers. 2020. Saqe: practical privacy-preserving approximate query processing for data federations. *PVLDB* (2020), 2691–2705.
[5] Paolo Bellavista, Luca Foschini, and Alessio Mora. 2021. Decentralised learning in federated deployment environments: A system-level survey. *Comput. Surveys* 54, 1 (2021), 1–38.
[6] Dmytro Bogatov, Georgios Kellaris, George Kollios, Kobbi Nissim, and Adam O'Neill. 2021. εpsolute: Efficiently Querying Databases While Providing Differential Privacy. In *ACM CCS*. 2262–2276.
[7] California State Legislature. 2018. California Consumer Privacy Act. https://oag.ca.gov/privacy/ccpa.
[8] T-H. Hubert Chan, Kai-Min Chung, Bruce M. Maggs, and Elaine Shi. 2019. Foundations of Differentially Oblivious Algorithms. In *ACM SODA*. 2448–2467.
[9] Lijun Chang, Xing Feng, Kai Yao, Lu Qin, and Wenjie Zhang. 2022. Accelerating Graph Similarity Search via Efficient GED Computation. *IEEE TKDE* (2022), 4485–4498.
[10] Zhao Chang, Dong Xie, Sheng Wang, and Feifei Li. 2022. Towards Practical Oblivious Join. In *ACM SIGMOD*. 803–817.
[11] King Lum Cheung and Ada Wai-Chee Fu. 1998. Enhanced nearest neighbour search on the R-tree. *ACM SIGMOD Record* (1998), 16–21.
[12] Victor Costan and Srinivas Devadas. 2016. Intel SGX explained. *Cryptology ePrint Archive* (2016).
[13] Ankur Dave, Chester Leung, Raluca Ada Popa, Joseph E Gonzalez, and Ion Stoica. 2020. Oblivious coopetitive analytics using hardware enclaves. In *ACM EuroSys*. 1–17.
[14] Colin de la Higuera and Francisco Casacuberta. 2000. Topology of strings: Median string is NP-complete. *Theoretical computer science* 230, 1-2 (2000), 39–48.
[15] Yichao Du, Zhirui Zhang, Bingzhe Wu, Lemao Liu, Tong Xu, and Enhong Chen. 2022. Federated Nearest Neighbor Machine Translation. *ICLR* (2022).
[16] Ahmed Eldawy and Mohamed F Mokbel. 2015. Spatialhadoop: A mapreduce framework for spatial data. In *IEEE ICDE*. 1352–1363.
[17] European Parliament and Council of the European Union. 2016. General Data Protection Regulation. https://eur-lex.europa.eu/eli/reg/2016/679/oj.

[18] Xinrui Ge, Jia Yu, Hanlin Zhang, Chengyu Hu, Zengpeng Li, Zhan Qin, and Rong Hao. 2019. Towards achieving keyword search over dynamic encrypted cloud data with symmetric-key based verification. *IEEE TDSC* 18, 1 (2019), 490–504.

[19] Craig Gentry. 2009. Fully homomorphic encryption using ideal lattices. In *ACM STOC*. 169–178.

[20] Oded Goldreich. 1987. Towards a theory of software protection and simulation by oblivious RAMs. In *ACM STOC*. 182–194.

[21] Karam Gouda and Mosab Hassaan. 2016. CSI_GED: An efficient approach for graph edit similarity computation. In *IEEE ICDE*. 265–276.

[22] Adam Groce, Peter Rindal, and Mike Rosulek. 2019. Cheaper private set intersection via differentially private leakage. *Proceedings on Privacy Enhancing Technologies* 2019, 3 (2019).

[23] Yu Guo, Chen Zhang, Cong Wang, and Xiaohua Jia. 2022. Towards Public Verifiable and Forward-Privacy Encrypted Search by Using Blockchain. *IEEE TDSC* (2022).

[24] Feng Han, Lan Zhang, Hanwen Feng, Weiran Liu, and Xiangyang Li. 2022. Scape: Scalable Collaborative Analytics System on Private Database with Malicious Security. In *IEEE ICDE*. 1740–1753.

[25] Xi He, Ashwin Machanavajjhala, Cheryl Flynn, and Divesh Srivastava. 2017. Composing differential privacy and secure computation: A case study on scaling private record linkage. In *ACM CCS*. 1389–1406.

[26] Megha Jain, Sanjay Kumar, and VK Patle. 2015. Bitonic sorting algorithm: A review. *International Journal of Computer Applications* 113, 13 (2015).

[27] David Kaplan, Jeremy Powell, and Tom Woller. 2016. AMD memory encryption. *White paper* (2016).

[28] Jongik Kim. 2021. Boosting graph similarity search through pre-computation. In *ACM SIGMOD*. 951–963.

[29] Simeon Krastnikov, Florian Kerschbaum, and Douglas Stebila. 2020. Efficient oblivious database joins. *PVLDB* (2020), 2132–2145.

[30] Conglong Li, Minjia Zhang, David G Andersen, and Yuxiong He. 2020. Improving approximate nearest neighbor search through learned adaptive early termination. In *ACM SIGMOD*. 2539–2554.

[31] Xiang Li, Fabing Li, and Mingyu Gao. 2023. Flare: A Fast, Secure, and Memory-Efficient Distributed Analytics Framework. *PVLDB* (2023), 1439–1452.

[32] John Liagouris, Vasiliki Kalavri, Muhammad Faisal, and Mayank Varia. 2023. SECRECY: Secure collaborative analytics in untrusted clouds. In *USENIX NSDI*. 1031–1056.

[33] Zhaorong Liu, Leye Wang, and Kai Chen. 2020. Secure efficient federated knn for recommendation systems. In *The International Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery*. Springer, 1808–1819.

[34] Yun Peng, Byron Choi, Tsz Nam Chan, and Jianliang Xu. 2022. LAN: Learning-based Approximate k-Nearest Neighbor Search in Graph Databases. In *IEEE ICDE*. 2508–2521.

[35] Rishabh Poddar, Sukrit Kalra, Avishay Yanai, Ryan Deng, Raluca Ada Popa, and Joseph M Hellerstein. 2021. Senate: A Maliciously-Secure MPC Platform for Collaborative Analytics. In *USENIX Security*. 2129–2146.

[36] Lianke Qin, Rajesh Jayaram, Elaine Shi, Zhao Song, Danyang Zhuo, and Shumo Chu. 2023. Adore: Differentially Oblivious Relational Database Operators. *PVLDB* (2023), 842–855.

[37] Parikshit Ram and Kaushik Sinha. 2022. Federated nearest neighbor classification with a colony of fruit-flies. In *AAAI*, Vol. 36. 8036–8044.

[38] Elaine Shi. 2020. Path oblivious heap: Optimal and practical oblivious priority queue. In *IEEE S&P*. 842–858.

[39] Saba skandarian and Matei Zaharia. 2019. ObliDB: Oblivious Query Processing for Secure Databases. *PVLDB* (2019), 169–183.

[40] Dawn Xiaoding Song, David Wagner, and Adrian Perrig. 2000. Practical techniques for searches on encrypted data. In *IEEE S&P*. 44–55.

[41] Yongxin Tong, Xuchen Pan, Yuxiang Zeng, Yexuan Shi, Chunbo Xue, Zimu Zhou, Xiaofei Zhang, Lei Chen, Yi Xu, Ke Xu, et al. 2022. Hu-Fu: efficient and secure spatial queries over data federation. *PVLDB* (2022), 1159–1172.

[42] Nikolaj Volgushev, Malte Schwarzkopf, Ben Getchell, Mayank Varia, Andrei Lapets, and Azer Bestavros. 2019. Conclave: secure multi-party computation on big data. In *ACM EuroSys*. 1–18.

[43] Songlei Wang, Yifeng Zheng, Xiaohua Jia, Hejiao Huang, and Cong Wang. 2022. OblivGM: Oblivious Attributed Subgraph Matching as a Cloud Service. *IEEE TIFS* 17 (2022), 3582–3596.

[44] Yilei Wang and Ke Yi. 2021. Secure Yannakakis: Join-Aggregate Queries over Private Data. In *ACM SIGMOD*. 1969–1981.

[45] Haotian Wu, Rui Song, Kai Lei, and Bin Xiao. 2022. Slicer: Verifiable, Secure and Fair Search over Encrypted Numerical Data Using Blockchain. In *IEEE ICDCS*. 1201–1211.

[46] Pengfei Wu, Qi Li, Jianting Ning, Xinyi Huang, and Wei Wu. 2021. Differentially Oblivious Data Analysis with Intel SGX: Design, Optimization, and Evaluation. *IEEE TDSC* (2021).

[47] Lee Xiong, Chenyan Xiong, Ye Li, Kwok-Fung Tang, Jialin Liu, Paul N. Bennett, Junaid Ahmed, and Arnold Overwijk. 2021. Approximate Nearest Neighbor Negative Contrastive Learning for Dense Text Retrieval. In *ICLR*.

[48] Yuanzhong Xu, Weidong Cui, and Marcus Peinado. 2015. Controlled-channel attacks: Deterministic side channels for untrusted operating systems. In *IEEE S&P*. 640–656.
[49] Zhong Yang, Bolong Zheng, Xianzhi Wang, Guohui Li, and Xiaofang Zhou. 2022. minIL: A Simple and Small Index for String Similarity Search with Edit Distance. In *IEEE ICDE*. 565–577.
[50] Minghe Yu, Jin Wang, Guoliang Li, Yong Zhang, Dong Deng, and Jianhua Feng. 2017. A unified framework for string similarity search with edit-distance constraint. *VLDBJ* 26, 2 (2017), 249–274.
[51] Samee Zahur and David Evans. 2015. Obliv-C: A language for extensible data-oblivious computation. *Cryptology ePrint Archive* (2015).
[52] Zhiping Zeng, Anthony KH Tung, Jianyong Wang, Jianhua Feng, and Lizhu Zhou. 2009. Comparing stars: On approximating graph edit distance. *PVLDB* (2009), 25–36.
[53] Haoyu Zhang and Qin Zhang. 2020. Minsearch: An efficient algorithm for similarity search under edit distance. In *ACM SIGKDD*. 566–576.
[54] Xinyi Zhang, Qichen Wang, Cheng Xu, Yun Peng, and Jianliang Xu. 2023. FedKNN: Secure Federated k-Nearest Neighbor Search (Technical Report). https://www.comp.hkbu.edu.hk/~db/fedknn_cr_tech_report.pdf.
[55] Wenting Zheng, Ankur Dave, Jethro G Beekman, Raluca Ada Popa, Joseph E Gonzalez, and Ion Stoica. 2017. Opaque: An oblivious and encrypted distributed analytics platform. In *USENIX NSDI*. 283–298.

Received July 2023; revised October 2023; accepted November 2023