Hu-Fu: Efficient and Secure Spatial Queries over Data Federation

Yongxin Tong[†], Xuchen Pan[†], Yuxiang Zeng[‡], Yexuan Shi[†], Chunbo Xue[†], Zimu Zhou[#], Xiaofei Zhang[§], Lei Chen[‡], Yi Xu[†], Ke Xu[†], Weifeng Lv[†]
[†]SKLSDE Lab, BDBC and IRI, Beihang University, China
[‡]The Hong Kong University of Science and Technology [#]Singapore Management University [§]University of Memphis
[†]{yxtong, panxuchen, skyxuan, xuechunbo, xuyi, kexu, lwf}@buaa.edu.cn, [‡]{yzengal, leichen}@cse.ust.hk,
[#]zimuzhou@smu.edu.sg, [§]xiaofei.zhang@memphis.edu

ABSTRACT

Data isolation has become an obstacle to scale up query processing over big data, since sharing raw data among data owners is often prohibitive due to security concerns. A promising solution is to perform secure queries over a federation of multiple data owners leveraging secure multi-party computation (SMC) techniques, as evidenced by recent federation work over relational data. However, existing solutions are highly inefficient on spatial queries due to excessive secure distance operations for query processing and their usage of general-purpose SMC libraries for secure operation implementation. In this paper, we propose Hu-Fu, the first system for efficient and secure spatial query processing on a data federation. The idea is to decompose the secure processing of a spatial query into as many plaintext operations and as few secure operations as possible, where fewer secure operators are involved and all secure operators are implemented dedicatedly. As a working system, Hu-Fu supports not only query input in native SQL, but also heterogeneous spatial databases (*e.g.*, PostGIS, Simba, GeoMesa, and SpatialHadoop) at the backend. Extensive experiments show that Hu-Fu usually outperforms the state-of-the-arts in running time and communication cost while guaranteeing security.

PVLDB Reference Format:
Yongxin Tong, Xuchen Pan, Yuxiang Zeng, Yexuan Shi, Chunbo Xue, Zimu Zhou, Xiaofei Zhang, Lei Chen, Yi Xu, Ke Xu, and Weifeng Lv. Hu-Fu: Efficient and Secure Spatial Queries over Data Federation. PVLDB, 14(1): XXX-XXX, 2020.
doi:XX.XX/XXX.XX

PVLDB Artifact Availability:
The source code, data, and technical report have been made available at https://github.com/BUAA-BDA/Hu-Fu.

1 INTRODUCTION

Efficient processing of spatial queries over large-scale data is essential for a wide spectrum of smart city applications including taxi-calling [66], logistics planning [35], map service [68], and contact-tracing [37] to name a few. Although the volume of spatial data continues to grow, it becomes increasingly difficult for these applications to take full advantage of the big spatial data due to the data isolation problem (*a.k.a.* isolated data) [9, 13, 64]. Spatial datasets at city or nation scale are often privately possessed and separately owned by multiple parties, where sharing raw data among parties or uploading raw data to a third party (*e.g.*, a cloud) is prohibitive due to legal regulations (*e.g.*, GDPR [55]) or commercial reasons.

A promising paradigm to tackle the data isolation problem is to perform *secure* queries over a *data federation* [4], which consists of multiple data owners *a.k.a.* data silos [31, 37], who agree on the same schema and manage their own data autonomously. Note that this paradigm differs from conventional federated databases [26, 49] in the extra security requirement. In general, secure query processing over data federation can be solved by well-known techniques such as secure multi-party computation (SMC) [7]. Yet, only recently did pioneer studies such as SMCQL [4] and Conclave [56] take the first step towards practice with efficient query execution plans upon SMC libraries for (relational) data federation. Unsurprisingly, more applications are being built on federations of spatial data owners.

Example 1. During COVID-19, several mobile network operators (*e.g.*, China Mobile [39] and China Telecom [52]) have cooperated as a spatial data federation to identify who has been to infectious areas through their location data [54]. Executing spatial queries (*e.g.*, range query or distance join) over a spatial data federation can help identify contacts in infectious areas across multiple organizations' spatial data without leaking privacy.

Example 2. AMAP (*a.k.a.* GaoDe Map) [3] has united over 8 Chinese travel companies into an integrated taxi-calling platform to offer users the taxis resources from all participating companies [53]. A spatial data federation can protect the distribution of taxis' locations of each company, which could be a business secret, from leaking to others.

Nevertheless, directly adapting the state-of-the-art data federation solutions [4, 56] to spatial data can be inefficient. From our empirical study (Sec. 2.2) of a kNN query on a real dataset, they are at least 142× slower, and have at least $1,216\times$ higher communication cost than plaintext query processing. There are two reasons for such inefficiency. *(i)* Existing solutions process spatial queries with excessive secure distance operations, which occupy over 90% of the time cost. For example, SMCQL [4] and Conclave [56] would securely sort spatial objects by distances to the query point and pick the top-k objects, where each sorting involves numerous secure distance comparisons. *(ii)* Previous studies [4, 56] are built on general-purpose SMC libraries, which may sacrifice the efficiency of specific operations for other considerations. For example, our experiment shows that the secure summation in ObliVM [36], the

Proceedings of the VLDB Endowment, Vol. 14, No. 1 ISSN 2150-8097.
doi:XX.XX/XXX.XX

SMC library adopted by SMCQL [4], can be accelerated by 15× via dedicated implementations [19].

In this paper, we aim at efficient and secure spatial queries over a data federation, which we call as *federated spatial queries*. We mainly study five queries (federated range query/counting, kNN query, distance join and kNN join) commonly seen in spatial database research [16, 44, 63] and follow the semi-honest adversary model adopted by previous work [4, 23, 56, 59]. Moreover, we develop a more practical solution than [4, 56] by eliminating the need for an honest broker and supporting more data silos (these works support no more than 3 data silos whereas we tested up to ten).

To this end, we propose Hu-Fu, a system for efficient and secure processing of federated spatial queries. As explained above, secure operations are usually slow and easily become the efficiency bottleneck. Thus, the **key idea** of Hu-Fu is to decompose a federated spatial query into as many plaintext operations and as few secure operations as possible without compromising security, where *(i)* no secure distance-related operations are involved and *(ii)* the secure operations have implementations faster than those in general-purpose SMC libraries. To realize this idea and implement a practical system, Hu-Fu consists of three components: an query rewriter with novel decomposition plans, a set of drivers adaptable to heterogeneous databases and an easy-to-use query interface with SQL support. Specifically, the query rewriter identifies a set of plaintext and secure operators to cover the queries of interest, and adopts novel decomposition plans to minimize the usage of secure operators while ensuring security. The drivers provide the implementations of secure operators with dedicated SMC protocols and plaintext operations as interfaces on top of the heterogeneous spatial databases adopted by different data silos. The query interface supports spatial queries in native SQL for easy usage.

Our main contributions and results are summarized as follows.

- To the best of our knowledge, Hu-Fu is the first system on efficient and secure spatial queries over a data federation.
- We devise novel decomposition plans for federated spatial queries. After decomposition, an execution plan involves only a limited number of secure operators that can be effectively supported with fast and dedicated implementations.
- Hu-Fu is an efficient, easy-to-use system that supports query input in SQL and heterogeneous spatial databases, *e.g.*, PostGIS [46], MySQL [60], SpatiaLite [50], Simba [63], GeoMesa [27], and SpatialHadoop [16].
- Extensive evaluations show that Hu-Fu usually outperforms the state-of-the-arts [4, 56] in efficiency. Compared with two strong baselines, namely SMCQL-GIS and Conclave-GIS, which are extended from SMCQL [4] and Conclave [56] to spatial queries, Hu-Fu is up to 4 orders of magnitude faster and 5 orders of magnitude lower in communication than SMCQL-GIS and Conclave-GIS with the same security level.

In the rest of this paper, we define our problem scope and identify the inefficiency of existing solutions in Sec. 2. We present an overview of Hu-Fu in Sec. 3 and elaborate on its functional components in Sec. 4, Sec. 5 and Sec. 6. Finally, we present the evaluations in Sec. 7, review the related work in Sec. 8, and conclude in Sec. 9.

2 PROBLEM STATEMENT

This section clarifies our problem scope (Sec. 2.1) and highlights the challenges (Sec. 2.2) that motivate the design of Hu-Fu.

2.1 Problem Scope

We consider a data federation F ("federation" for short) consisting of n data silos ("silos" for short, denoted by F_i), where each silo holds multiple *spatial objects*. Each spatial object o has a location l_o and other attributes a_o. The federation supports *federated spatial queries* over the spatial objects of all silos under the following settings.

- *Spatial queries:* The federation F should support mainstream spatial queries including range query, range counting, kNN query, distance join, and kNN join [51, 63].
- *Autonomous databases:* Each silo is an autonomous database that does not share its raw spatial objects with other silos. This is aligned with real-world data federations [4–6, 56].
- *Semi-honest adversaries:* Each silo honestly executes queries received and returns authentic results, but may attempt to infer data from other silos during query execution. For example, when processing range counting over a data federation, if each silo sends the counting result of its data to one silo (*e.g.*, the semi-honest adversary) for a further sum up, the adversary will know the sensitive counting result of other silos. This assumption is common in query processing over a data federation [4, 56].

We focus on query processing with the following requirements.

- **Efficiency requirements.** We care about the *running time* and *communication cost* to execute *exact* queries over multiple silos. Short running time is desirable since real applications may process massive queries and a long latency can have bad effects (*e.g.*, it may cause an extended spread of diseases for contact tracing or degraded user experience for taxi-calling). Minimal communication cost is critical in distributed query processing [17, 44] and secure query processing [29]. Approximate query processing over data federation [6, 14] is out of our scope because applications such as contact tracing require accurate results. We consider multiple silos as aligned with real-world applications. Similar to existing federated query solutions [4, 56], the storage efficiency, which mainly depends on silos themselves, is not our primary concern.
- **Security requirements.** We target the scenario where input queries are public to all silos yet neither the query user nor any silo could deduce extra information from the final results. For instance, in federated kNN query, the query user can only know the final result (*i.e.* k nearest neighbors), and cannot infer the ownership of these k nearest neighbors. Such requirements are common in secure query processing [7].

Security is often of utmost priority due to laws (*e.g.*, GDPR [45] and CCPA [40]) on data protection. To satisfy the security requirement, existing systems [4, 56] rely on an honest broker to securely collect the partial answers (which may have sensitive data) from each silo. For other operations, they still rely on secure protocols (*e.g.*, summing the local counts from each silo to answer a range counting). However, real-world brokers (*e.g.*, Acxiom [1] and Experian [21]), which need to be paid a lot for data broker services, may still leak sensitive data for profit or by accident [48]. Thus, we do not assume an honest broker in Hu-Fu.

Formally, we define the federated spatial queries of interest below. They are common in existing spatial data systems [16, 44, 63]. Here, function $d(p, o)$ is the distance between spatial objects p and o.

Definition 1 (Federated Range Query/Counting)**.** Given a federation $F = \{F_1, \cdots, F_n\}$, and a query range $\mathcal{R}$, *federated range query* returns all objects $o \in F$ located in $\mathcal{R}$; *federated range counting* returns the number of these objects. These two queries should only return the final results without revealing any information of F_i (*e.g.*, the ownership of objects, the number of objects) to F_j $(j \neq i)$.

Definition 2 (Federated kNN Query)**.** Given a federation $F = \{F_1, \cdots, F_n\}$, a query point p and an integer k, *federated kNN query* returns a set $\mathrm{kNN}(F, p, k)$ of k spatial objects such that

$$\forall o \in \mathrm{kNN}(F, p, k), \forall o' \in F - \mathrm{kNN}(F, p, k), d(p, o) \leq d(p, o')$$

without revealing information except the returned set to any F_i.

Definition 3 (Federated Distance Join)**.** Given an input dataset of spatial objects R, a federation $F = \{F_1, \cdots, F_n\}$ and a radius r, *federated distance join* returns each $o \in R$ with each $o' \in F$ that satisfies $d(o, o') \leq r$ as pairs, without revealing the ownership of $o' \in F_i$ to F_j $(j \neq i)$.

$$R \bowtie_r F = \{(o, o') | o \in R, o' \in F, d(o, o') \leq r\}$$

Definition 4 (Federated kNN Join)**.** Given an input dataset of spatial objects R, a federation $F = \{F_1, \cdots, F_n\}$ and k, *federated kNN join* returns each $o \in R$ with each $o' \in \mathrm{kNN}(F, o, k)$ as pairs, without revealing information except the returned set to F_i.

$$R \bowtie_{\mathrm{kNN}} F = \{(o, o') | o \in R, o' \in \mathrm{kNN}(F, o, k)\}$$

2.2 Challenges

Federated spatial queries can be realized by secure multi-party computation (SMC) [7], as in prior studies for relational data [4, 56]. Nevertheless, our empirical study shows that they are highly inefficient on spatial queries, as explained below.

2.2.1 *Inefficiency on Federated Spatial Queries.*

As an illustrative study, we perform federated kNN query by extending SMCQL [4] and Conclave [56], two representative solutions to secure query processing on (relational) data federations.

Overview of Existing Solutions. The general framework to apply SMC techniques for secure query processing over data federations is to decouple query execution into first *plaintext* queries within each silo and then *secure* computations of the final results across silos [4, 56]. This is because SMC protocols are slow and such a framework accelerates query processing without compromising security. Existing solutions differ in the underlying SMC techniques they apply for secure operations, where garbled circuit (GC) and secret sharing (SS) are two mainstream SMC techniques [7]. Specifically, SMCQL [4], the first solution for secure query processing over a data federation, uses ObliVM [36], a prevalent GC based library. Since ObliVM only supports two silos, Conclave [56] adopts an SS based technique (Sharemind [11]), which enables query processing on three silos.

Setup. We extend SMCQL [4] and Conclave [56] to federated kNN queries as follows. Following the "plaintext + secure" processing pipeline, each silo first conducts a plaintext kNN query and returns the k nearest points (along with their distances) to the query point. Then, the final k nearest neighbors are derived from these returned points, which are securely sorted by their distances to the query point and the k nearest ones are picked. We experiment with two

Table 1: Percentage of time spent for *plaintext* or *SMC* operations for a federated kNN query via existing solutions.

System	Plaintext	SMC
SMCQL-GIS	0.14%	99.86%
Conclave-GIS	0.10%	99.90%

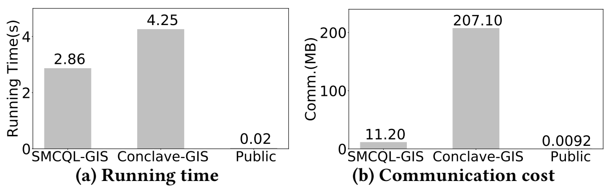


Figure 1: Inefficiency of Conclave-GIS and SMCQL-GIS on federated kNN query, where SMCQL-GIS and Conclave-GIS are our extensions on SMCQL [4] and Conclave [56] to spatial queries (see Sec. 7.1).

silos with $k = 16$. Other implementations and experimental setup details are in Sec. 7.1.

Results. Fig. 1 plots the running time and communication cost to process a single federated kNN query leveraging existing solutions. The results are averaged over 50 queries. Compared with Public, *i.e.* plaintext kNN query execution without the security requirement, the secure counterpart incurs 142× to 212× longer running time and 1, 216× to 22, 510× higher communication cost. Although SMCQL-GIS yields a shorter running time and a lower communication cost than Conclave-GIS, SMCQL-GIS is *only applicable to the scenario of two silos* for its usage of GC based SMC techniques. Yet it still takes 2.86 seconds for a single federated kNN query, which can hurt user experiences in applications where time efficiency is critical.

2.2.2 *Understanding the Efficiency Bottleneck.*

Prior studies are inefficient on federated spatial queries for the following reasons.

- **Excessive Secure Distance Operations**. When processing a federated kNN query, over 99% time is spent on SMC operations (*e.g.*, secure distance comparisons) as shown in Table 1. For example, SMCQL-GIS and Conclave-GIS adopt sorting to find k nearest neighbors among nk candidates by using $O(nk \log(nk))$ secure distance comparisons, and a single secure distance comparison takes 209 ms in SMCQL-GIS and 248 ms in Conclave-GIS, which equals to the time of at least 10^6 plaintext comparisons.
- **Reliance on General-Purpose Libraries.** Existing methods use general-purpose libraries to implement SMC operations (*e.g.*, ObliVM [36] in SMCQL [4]). General-purpose libraries sometimes sacrifice efficiency for generalization or compatibility. For example, the secure summation we used can be 16× faster than that in ObliVM (see Sec. 7). As will be shown in Sec. 4, we can process federated spatial queries with only a few secure operations. This facilitates acceleration with libraries dedicated to such operations [11, 19, 28].

Takeaways. Our study shows that existing secure query processing solutions (*e.g.*, [4, 56]) for data federations are inefficient for spatial queries. The inefficiency comes from *(i)* massive secure distance operations, and is exacerbated by *(ii)* adopting general-purpose libraries for these SMC operations. In response, we propose Hu-Fu, a solution with *(i)* a novel execution plan for federated spatial queries that involve notably fewer secure operations (see Sec. 4) and *(ii)* each secure operator can be implemented in high efficiency via dedicated algorithms (see Sec. 5). As next, we give an overview of Hu-Fu and elaborate on its functional modules in the following.

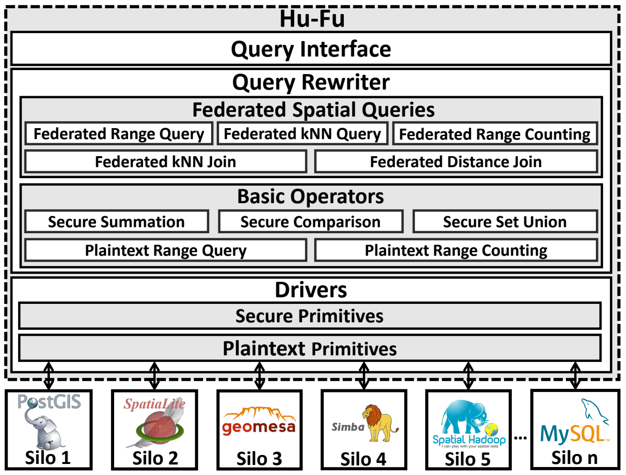


Figure 2: Illustration of Hu-Fu architecture.

3 HU-FU OVERVIEW

Hu-Fu is a solution that enables efficient and secure spatial queries over a data federation. It addresses the inefficiency of federated spatial query processing (see Sec. 2.2.2) via two modules: *(i)* a novel **query rewriter** that decomposes federated spatial queries into *plaintext and secure operators*, with the former executed within each silo and the latter across silos; *(ii)* **drivers** that implement these operators as *plaintext and secure primitives* leveraging dedicated algorithms and optimizations. Hu-Fu also contains a transparent **query interface** to support federated spatial queries written in native SQL. We briefly explain its architecture and workflow below.

3.1 Architecture

Fig. 2 illustrates the architecture of Hu-Fu, which consists of three modules: the query interface, the query rewriter and drivers. From a functional perspective, the query rewriter and drivers optimize the *efficiency* of federated spatial queries, and the query interface improves the *usability* of Hu-Fu.

Query Rewriter (Sec. 4). It decomposes federated spatial queries into plaintext operators (executed within silos) and secure operators (executed across silos). We define two *plaintext* operators (plaintext range query and range counting) and three *secure* operators (secure summation, comparison and set union) as the basic operators, upon which we design novel execution plans that decompose mainstream federated spatial queries (federated range query, range counting, kNN query, distance join and kNN join) into these basic operators.

Drivers (Sec. 5). Hu-Fu's drivers implement the basic operators defined in the query rewriter as efficient *primitives* that can adapt to heterogeneous spatial databases at the backend. Each operator is implemented by a specific primitive. Specifically, secure operators are implemented as *secure primitives* with dedicated optimizations [11, 19, 28]. Plaintext operators are implemented as *plaintext primitives* on top of the underlying spatial databases, which support various systems, *e.g.*, PostGIS [46], SpatiaLite [50], MySQL [60], GeoMesa [27], Simba [63] and SpatialHadoop [16].

Query Interface (Sec. 6). This module *(i)* provides a transparent and unified federation view to users, and *(ii)* supports federated spatial queries written in SQL. We implement the query interface by extending the schema manager and parser of Calcite [8]. We also provide interfaces such as JDBC for easy integration of Hu-Fu to users' programs.

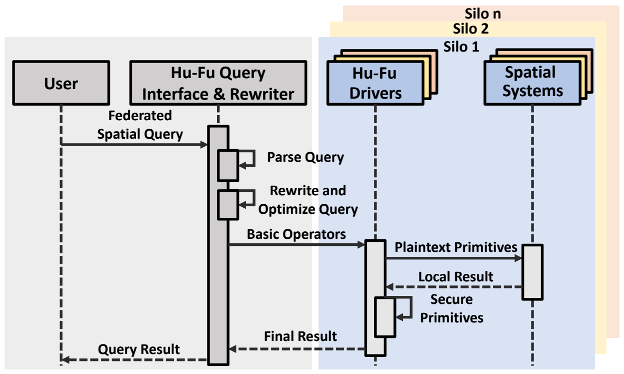


Figure 3: Illustration of Hu-Fu workflow.

3.2 Workflow

Fig. 3 shows the workflow of Hu-Fu with a user querying a data federation of n silos. The query interface and query rewriter are deployed on the user machine to provide a portal for spatial services. Each silo runs an instance of Hu-Fu drivers to interact with its underlying spatial databases.

Suppose the user's spatial service issues a federated spatial query written in SQL. When a federated spatial query comes in, it is first parsed by the query interface. Then the query rewriter transforms and optimizes the query into a sequence of plaintext and secure operators. These operators are then sent to drivers for execution as plaintext and secure primitives. First, the plaintext primitives are executed on the underlying spatial databases at each silo to get the local results. Afterward, the local results are collected to perform the secure primitives for the final query result, which is returned to the user by the query interface.

4 QUERY REWRITER

This section presents Hu-Fu's query rewriter, which decomposes federated spatial queries into multiple basic operators. We first define the basic operators in Sec. 4.1 before explaining the overall decomposition strategies in Sec. 4.2. Specifically, we categorize the five federated spatial queries into *radius-known* and *radius-unknown* queries, and elaborate on their decomposition in Sec. 4.3 and Sec. 4.4. We discuss other practical issues in Sec. 4.5.

4.1 Basic Operators

Our acceleration strategy is to *decompose queries into basic operators such that distance-related operations are restricted within silos in plaintext, leaving only secure operations across silos.* The selection of basic operators is explained below.

4.1.1 Operator Selection Principles. We propose two categories of basic operators: *plaintext* and *secure* operators. The plaintext operators perform local queries within each individual silo, while the secure operators securely collect the local query results from different silos as the final output.

- **Plaintext Operators.** They can involve the distance-related operations compulsory in spatial queries, but should be common operations widely supported by diverse spatial databases.
- **Secure Operators.** They should avoid distance-related operations, and efficiently implemented operators are preferable.

Following these principles, we choose two plaintext operators (*plaintext range query*, *plaintext range counting*) and three secure operators (*secure summation*, *secure comparison*, *secure set union*). We define each basic operator and justify our selections below.

4.1.2 Plaintext Operators. We define two plaintext operators: *plaintext range query* and *plaintext range counting*. These operators are performed within each silo F_i. Hence, they can be conducted in plaintext without compromising security.

Definition 5 (Plaintext Range Query/Counting)**.** For a silo F_i, given a query range $\mathcal{R}$, the *plaintext range query* $\mathsf{PRQ}_{F_i}(\mathcal{R})$ returns in plaintext the spatial objects in F_i within $\mathcal{R}$, and the *plaintext range counting* $\mathsf{PRC}_{F_i}(\mathcal{R})$ further returns the number of such objects.

The plaintext operators comply with the principles described in Sec. 4.1.1, because *(i)* the returned results can be securely collected without secure distance operations (see Sec. 4.1.3) and *(ii)* they are supported by almost all spatial databases [44]. These operators are implemented as *plaintext primitives* in Hu-Fu drivers, which we defer to Sec. 5.1. The query range can be a circle, a rectangle or other shapes. For ease of presentation, we focus on a circular range in Sec. 4.2-4.4 and discuss extensions to other shapes in Sec. 4.5.

4.1.3 Secure Operators. Based on the secure multi-party computation techniques [11, 19, 28], we define three secure operators: *secure summation*, *secure comparison*, and *secure set union*. These operators are performed across silos and responsible for secure result collection from local query results returned by plaintext operators.

Definition 6 (Secure Summation)**.** For a federation $F = \{F_1, \cdots, F_n\}$, where each silo F_i holds a number β_i, this operator calculates the sum $\sum_{i=1}^{n} \beta_i$, while avoiding leaking β_i to F_j $(j \neq i)$.

$$\mathsf{SSM}_F(\beta_1, \cdots, \beta_n) = \sum_{i=1}^{n} \beta_i$$

Definition 7 (Secure Comparison)**.** For a federation $F = \{F_1, \cdots, F_n\}$, where each silo F_i holds a number β_i, and a value k, this operator compares $\sum_{i=1}^{n} \beta_i$ with k without leaking $\sum_{i=1}^{n} \beta_i$ or β_i to any F_i.

$$\mathsf{SCP}_F(\beta_1, \cdots, \beta_n, k) = sign\Big(\sum_{i=1}^{n} \beta_i - k\Big)$$

Definition 8 (Secure Set Union)**.** For a federation $F = \{F_1, \cdots, F_n\}$, where each silo F_i holds a set of spatial objects $S_i = \{o_1^i, \cdots, o_{m_i}^i\}$, this operator computes the union of spatial objects from all silos, without leaking the ownership of each $o \in S_i$ to F_j $(j \neq i)$.

$$\mathsf{SSU}_F(S_1, \cdots, S_n) = \bigcup_{i=1}^{n} S_i$$

The secure operators comply with the principles in Sec. 4.1.1 since *(i)* they do not involve distance operations and *(ii)* there are dedicated techniques for efficient implementations (see Sec. 5.2).

4.2 Overall Decomposition Strategies

The principle of query rewriter is to decompose federated spatial queries into as many plaintext operators and as few secure operators as possible such that a large portion of the query can be executed in plaintext without compromising security. At a high level, a federated spatial query is first executed as plaintext operators in each silo, where the results are then securely assembled as the final result. At the minimum, one secure operator is compulsory, and additional secure operators may be necessary if there are extra interactions across silos. Given the basic operators defined in Sec. 4.1, we classify federated spatial queries into two categories and explain their decomposition strategies as follows (see Table 2).

- **Radius-Known Queries.** For a radius-known query, it needs only one secure operator for result collection in the ideal case. This is because our plaintext operators already support plaintext range query and counting. For result collection, a secure set union or summation operator is required. We introduce the decomposition plans for radius-known queries in Sec. 4.3.
- **Radius-Unknown Queries.** For a radius-unknown query, *e.g.*, a federated kNN query, we convert the query into multiple rounds of radius-known queries. There can be cross-silo communication between rounds, and extra secure operators are necessary, which is secure comparison in our case. We adopt binary search to minimize the number of rounds and secure operators involved. We explain the decomposition plans for radius-unknown queries in Sec. 4.4.

4.3 Decomposing Radius-Known Queries

Among the five federated spatial queries, range query, range counting, and distance join belong to radius-known queries.

Decomposition Plan. *Federated range query* can be decomposed into n plaintext range queries with radius r, where each plaintext range query retrieves the local result from each one of n silos. Similarly, *federated range counting* can be decomposed into n plaintext range counting. Observing that a distance join can be viewed as $|R|$ times of range queries with different query points but the same radius, *federated distance join* is decomposed into $|R| \times n$ plaintext range queries with radius r. For result collection across silos, *federated range counting* needs a secure summation to aggregate the counts without revealing the count of any silo. For *federated range query*, it needs a secure set union to assemble the result without revealing the objects' ownership. For *federated distance join*, it also first assembles the results of the plaintext range query in each silo and then executes only one secure set union across silos.

Complexity Analysis. For ease of presentation, we denote the time complexity of plaintext range query/counting as T^q and T^c, respectively. For federated range query/counting, each silo executes a plaintext range query/counting and a secure set union/summation. Thus, their time complexities are $O(T^q + n + |S|)$ and $O(T^c + n^3)$, where $|S|$ is the size of returned set. The communication costs are $O(n + |S|)$ and $O(n^2)$, respectively. For federated distance join, each silo executes $|R|$ plaintext range queries, resulting in a time cost of $O(|R|T^q + n + |S|)$ with $O(n + |S|)$ communication cost. The complexity analysis of secure operators is deferred to Sec. 5.2.

4.4 Decomposing Radius-Unknown Queries

Federated kNN query and kNN join are radius-unknown queries, because there is no specific range in these queries. Thus, their decomposition plan is to first get an appropriate range and then filter the points in the range, as explained in detail below.

Decomposition Plan. Similar to the relation between federated range query and federated distance join in Sec. 4.3, federated kNN join can be viewed as $|R|$ independent federated kNN queries. Hence, we mainly explain how to decompose a federated kNN query.

- **Basic Idea.** Recall from Sec. 4.2, the strategy to decompose radius-unknown queries is to convert them into multiple rounds of radius-known queries. We first derive a radius (denoted by $thres$) via a binary search and then retrieve the spatial objects within this radius. Note that obtaining the exact counting result during the binary search via secure summation may leak extra information of silos. For example, the query user can get the

Algorithm 1: Rewriter of the federated kNN query $\mathrm{kNN}(F, p, k)$ over a data federation $F = \{F_1, \cdots, F_n\}$

```
1  $[l, u] \leftarrow [0, v_0]$, where $v_0$ is a predefined upper bound;
2  while $u - l \geq \epsilon_0$ do
3      $thres \leftarrow (l + u)/2$;
4      $\beta_i \leftarrow$ plaintext range counting $\mathrm{PRC}_{F_i}(\mathrm{circle}(p, thres))$;
5      $sgn \leftarrow$ secure comparison $\mathrm{SCP}_F(\beta_1, \cdots, \beta_n, k)$;
6      if $sgn = -1$ then $l \leftarrow thres$;
7      else if $sgn = 1$ then $u \leftarrow thres$;
8      else break;
9  $S_i \leftarrow$ plaintext range query $\mathrm{PRQ}_{F_i}(\mathrm{circle}(p, thres))$;
10 query answer $res \leftarrow$ secure set union $\mathrm{SSU}_F(S_1, \cdots, S_n)$;
```

number of objects within a range, which reveals the federation's data distribution. Hence, we only judge whether the counting result is larger than k and adopt a secure comparison instead. As long as *thres* is between the k^{th} and the $(k+1)^{th}$ nearest distance, the retrieved objects should be the k^{th} nearest neighbors.

- **Algorithm Details.** Alg. 1 illustrates decomposing a federated kNN query. Lines 1-8 derive the radius. We initialize a lower bound ($l = 0$) and upper bound ($u = v_0$) of the radius, where v_0 can be set as the diameter of the area or defined by the user. We then perform a binary search in lines 2-8, where ϵ_0 is the precision lower bound of distance. In each iteration, *thres* is set as $(l + u)/2$ in line 3. For the current radius *thres*, we perform a *plaintext range counting* for each silo in line 4 and a *secure comparison* between the sum of each silo's count (β_i) and the integer k in line 5. Lines 6-8 adjust the boundary of the searching radius. If the total count is smaller than k, the current radius is too short, and we update l to *thres* as the new lower bound (line 6). If the total count is larger than k, it means there are sufficient points within *thres* and we update the upper bound u as *thres* in line 7. The binary search guarantees that *thres* is sufficiently close to the k^{th} nearest distance. In the last round (lines 9-10), a *plaintext range query* $\mathrm{PRQ}_{F_i}(circle(p, thres))$ is performed on each silo and we use a *secure set union* to get the final result.

Complexity Analysis. Alg. 1 takes at most $O(\log \frac{v_0}{\epsilon_0})$ rounds to get the threshold (lines 2-8). In each round, the plaintext range counting (line 4) takes $O(T^c)$ time, and the secure comparison (line 5) takes $O(n)$ time. The adjustment of the binary search boundary (lines 6-8) takes $O(1)$ time. After obtaining the final threshold, the algorithm calls a plaintext range query in $O(T^q)$ time to get local results (line 9) and a secure set union in $O(n+k)$ time to assemble the results. Thus, the total time complexity is $O(T^q+k+(n+T^c) \log \frac{v_0}{\epsilon_0})$. In secure comparison (line 5), each silo communicates with the other $n-1$ silos. Thus, the communication cost for a single round is $O(n^2)$ and there are $O(n^2 \log \frac{v_0}{\epsilon_0})$ rounds in total. The communication of secure set union (line 10) is $O(n + k)$. The time complexity of federated kNN join is similar to federated kNN query, multiplied by a factor $|R|$, *i.e.* $O(|R|T^q + |R|k + |R|(n + T^c) \log \frac{v_0}{\epsilon_0})$.

Example 3. We illustrate the execution of a federated kNN query with query point $(4, 4)$ and $k = 4$ over 3 silos in Fig. 4, and the objects marked with the same color belong to the same silo. The query rewriter decomposes this query into multiple rounds of radius-known queries. In the 1st round, a plaintext range counting with center $(4, 4)$ and radius 4 is sent to each silo and a secure comparison with k is performed across silos. And we get 9 objects, which is greater than k. Hence in the 2nd round, the radius decreases to 2 and resent to silos for plaintext range counting and secure comparison. There are 2 objects, which is smaller than k. Thus in the 3rd round, the radius increases to 3 and the procedure continues, where the range counting result equals to k and the search terminates. Finally, a plaintext range query with center $(4, 4)$ and radius 3 plus a secure set union are performed to get the 4 objects.

Remark. The parameters v_0 and ϵ_0 control the upper bound and precision of the searching radius, respectively. In real-world data, we can derive suitable settings for v_0 and ϵ_0 based on the diameter of the spatial area and the precision of the locations' coordinates. For example, in our real dataset collected in Beijing, $v_0 = 200$ km roughly equals the diameter of the Beijing city and $\epsilon_0 = 1$ m represents the precision of the GPS coordinates in this dataset.

4.5 Discussions

We highlight the following discussions on the query rewriter.

Security of Rewriter. We prove the security of our query rewriter based on the composition lemma in [24] (Section 7.3.1). The idea is to show the decomposition plans for radius-known queries and radius-unknown queries will not reveal any extra information other than the final result due to the usage of secure operators. We also present a case study that proves it is hard for a semi-honest adversary to attack Hu-Fu. Please refer to Appendix A of our full paper [15] for the proof and case study due to the page limitation.

Differential Privacy to Accelerate Radius-Unknown Queries. We exploit differential privacy [34] to further accelerate federated kNN query and federated kNN join from two aspects.

- **Tighten Predefined Upper Bound.** We ask each F_i to conduct a local kNN query in plaintext and return the k^{th} object's distance to the query point d_i^k. Since directly returning such value may expose the real distances of silos, we apply the truncated Laplace mechanism [5] on it. That is, let each silo add a positive noise and get the perturbed value $d_i'^k$. We can tighten the upper bound as the smallest distance in all silos, *i.e.* $v_0 = \min_i d_i'^k$, since there are at least k points in this range.
- **Reduce Running Time and Communication Cost in Secure Comparison.** The secure comparison in Alg. 1 compares $\sum_1^n \beta_i$ with k, which incurs at least $O(n^2)$ running time and communication cost. It can be reduced to $O(n)$ when $\sum_1^n \beta_i$ notably differs from k. In this case, each silo can add a Laplacian noise [34] on its local counting result to hide the real counts of each silo, and then aggregate the perturbed results. If the perturbed result is much smaller/larger than k, we directly adjust the threshold.

Beyond Mainstream Spatial Queries. The decomposition plan for radius-known queries applies to federated range query/counting with other query range types (*e.g.*, rectangle). This is because the plaintext range query/counting with arbitrary shapes of query ranges is supported in each silo's underlying spatial data systems (*e.g.*, PostGIS). The query rewriter also supports aggregation queries, *e.g.*, the aggregate attribute on the result of kNN query or range

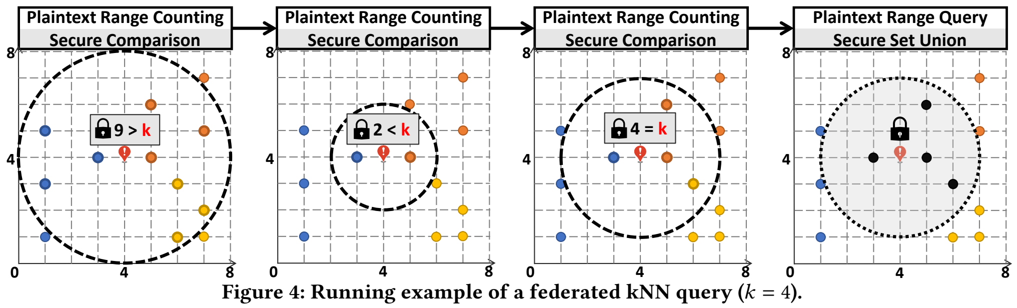


Figure 4: Running example of a federated kNN query ($k = 4$).

Table 2: The number of basic operators in the decomposition plans of federated spatial queries. Radius-known queries only involve one secure operator (secure set union/summation) for secure result collection. Radius-unknown queries are executed in multiple rounds which require extra secure operator (secure comparison) to ensure security. Here, n is the number of silos.

Category	Federated Spatial Query	Number of Plaintext Operator		Number of Secure Operator	
		Range Query	Range Counting	Comparison	Set Union/Summation
Radius-Known	Federated Range Query	n	0	0	1/0
	Federated Range Counting	0	n	0	0/1
	Federated Distance Join	$n\|R\|$ [†]	0	0	1/0
Radius-Unknown	Federated kNN Query	n	$O(n \log \frac{v_0}{\epsilon_0})$ [‡]	$O(\log \frac{v_0}{\epsilon_0})$	1/0
	Federated kNN Join	$n\|R\|$	$O(n\|R\| \log \frac{v_0}{\epsilon_0})$	$O(\|R\| \log \frac{v_0}{\epsilon_0})$	1/0

[†] $|R|$ is the size of the input dataset R in the federated distance join and federated kNN join.

[‡] v_0 and ϵ_0 are user-defined parameters for processing the federated kNN query and federated kNN join.

query. Specifically, the aggregation of kNN query can be decomposed the same as the federated kNN query, by only replacing the last secure set union with a secure summation. The range aggregate query can be decomposed similarly to a federated range counting.

5 DRIVERS

This section presents Hu-Fu's drivers, which offer interfaces and implementations on top of silos' spatial databases for the unified and efficient execution of decomposition plans generated by the query rewriter. A driver, which consists of *plaintext primitives* and *secure primitives*, is deployed on each silo. Upon receiving a decomposition plan, plaintext operators are first executed at each silo with plaintext primitives and then secure operators are performed via the secure primitives for result assembling. As next, we elaborate on plaintext primitives (Sec. 5.1) and secure primitives (Sec. 5.2) to efficiently implement the basic operators defined in the query rewriter.

5.1 Plaintext Primitives

The plaintext primitives implement plaintext range query and plaintext range counting. They are implemented as an *interface* on top of the underlying spatial databases for portability and to harness existing range query and range counting implementations.

Primitive Implementation. The implementation of plaintext primitives is dependent on the underlying spatial databases.

- For databases where range query and range counting are available, *e.g.*, Simba [63] and PostGIS [46], we directly call the corresponding functions for plaintext range query or counting. For example, in PostGIS, a plaintext range counting on silo F_i with the center p and radius r of a circular range can be implemented by calling the SQL below.

```
SELECT COUNT(*) FROM F_i
WHERE ST_DWithin(p, F_i.location, r);
```

- For databases without such queries, drivers provide a default implementation of range query and range counting. For example, GeoMesa [27] only provides an interface of range query. Thus, we implement range counting by first running a range query, and then counting the cardinality of the returned set.

Time Complexity. The time complexity of plaintext primitives depends on the native implementation in spatial databases. For example, plaintext range counting takes $O(\log m)$ time with spatial indices [47], where m is the data size. Yet plaintext range query may need $O(\log m + |S|)$ time, where S is the query result.

Discussions. We make two notes on the plaintext primitives.

- To support the differential privacy based acceleration for federated kNN query (see Sec. 4.5), Hu-Fu drivers provide an optional plaintext kNN query interface. The plaintext kNN is implemented by a function call on spatial databases with native kNN query (*e.g.*, PostGIS [46] and Simba [63]).
- Since the time complexity of plaintext primitives varies, the efficiency of federated spatial queries can be limited by the slowest plaintext primitive if silos are using heterogeneous spatial databases (see Sec. 7.4). Thus, more efficient plaintext range query/counting is out of our scope.

5.2 Secure Primitives

The Secure primitives implement secure summation, comparison, and set union, which are independent of the underlying spatial databases. Recall that secure primitives take the local results from plaintext primitives as inputs. To avoid idle waiting for slow silos and to reuse local results across silos, each silo buffers its local results of plaintext primitives executed on itself. We implement secure primitives on top of such a buffer, as explained next.

Primitive Implementation. Each secure primitive is implemented with a dedicated secure protocol for higher efficiency than the corresponding operation in general-purpose SMC libraries. We present the details of each secure primitive below.

Secure Summation. The implementation of secure summation is based on Ref. [19]. First, all of the n silos first agree on n different public parameters $U = \{u_1, u_2, \cdots, u_n\}$. Then, each silo F_i chooses a random $n-1$ degree polynomial $t_i(x) = (\sum_{k=1}^{n-1} a_{ik}x^k) + v_i$ and calculates n values of the polynomial, $t_i(u_1), \cdots, t_i(u_n)$. Specially, a_{ik} indicates the random coefficient independently generated by silo F_i, and v_i denotes the local counting result of silo F_i. These variables are held by silo F_i only and kept secret from others. Afterward, each silo F_i sends the value of polynomial $t_i(u_j)$ to all other $F_j (i \neq j)$. When any silo F_j receives all $\{t_i(u_j)|i \neq j\}$ from the other silos, it sums up those values $S(u_j) = \sum_{i=1}^{n} t_i(u_j) = (\sum_{k=1}^{n-1}(u_j^k \sum_{i=1}^{n} a_{ik})) + \sum_{i=1}^{n} v_i$ and sends the summations $S(u_j)$ to the query user. The user can regard $S(u_j)$ as a linear equation $S(u_j) = \sum_{k=1}^{n-1} u_j^k z_k + z_n$, where n unknown variables are $z_k = \sum_{i=1}^{n} a_{ik}$ (for $k = 1, \cdots, n-1$) and $z_n = \sum_{i=1}^{n} v_i$. Moreover, the user knows $\{u_1, u_2, \cdots, u_n\}$ and the values $\{S(u_1), S(u_2), \cdots, S(u_n)\}$. Thus, the user can solve the n unknown variables by Gauss elimination and get the value of $\sum_{i=1}^{n} v_i$ (*i.e.* the unknown variable z_n).

Secure Comparison. The primitive compares a user given constant k with the sum of each silo's local result (*e.g.*, $\{\beta_i\}$) and ensures that either β_i or $\sum_{i=1}^{n} \beta_i$ is confidential to any silos $F_j (j \neq i)$ and the query user. The main idea is calculating $X(\sum_{i=1}^{n} \beta_i - k)$ instead of $\sum_{i=1}^{n} \beta_i - k$, where X is a random and positive real number, because the latter result discloses the value of $\sum_{i=1}^{n} \beta_i$. Accordingly, we reduce our secure comparison into the classic secure multiplication and hence adopt the existing secure multiplication protocol [11] to ensure security. Specifically, the secure multiplication protocol requires two multipliers x and y are both divided into n shares $X = \sum_{i=1}^{n} x_i, Y = \sum_{i=1}^{n} y_i$ and each share is distributed into n silos, *e.g.*, x_i and y_i for silo F_i. This protocol can protect the values of X, Y, x_i, y_i from the attackers in all n silos. In our reduction, Y equals $\sum_{i=1}^{n} \beta_i - k$ and $y_i = \beta_i - \frac{k}{n}$. Since each silo has already known its local result β_i, the user only sends $\frac{k}{n}$ to all silos. After that, each silo randomly generates a positive real number x_i and calculates $XY = (\sum_{i=1}^{n} x_i)(\sum_{i=1}^{n}(\beta_i - \frac{k}{n}))$ by using the secure multiplication protocol (see [11] for more details), then returns XY to the user. Finally, the user derives the final result of our secure comparison by the sign of XY without leaking any sensitive information.

Secure Set Union. This primitive is based on an existing set union protocol [28] with differential privacy. Firstly, each silo generates some fake records based on the local query result set. Both the number of fake records and the attribute values in each fake record are perturbed by injecting the differential privacy noise. Then each silo incorporates those fake records into the local query result set, and sends it to the next silo. Note that the order of silos is randomly set for each processing. All the silos, except for the first one, need to mix the received set with their local set, and the last silo sends the mixed set to the first silo. After each silo receives and sends the result set, the first silo gets the result set of all silos, and prunes the fake records it has added before, then sends the pruned set to the next silo. Similarly, when other silos receive a set, they will remove their fake records and send the pruned set to the next one. After the last silo has pruned the fake records, it returns the final result set back to the query user. In the implementation, we observe there is too much idle wait, *e.g.*, non-first silo needs to wait for the former's noised result set. Thus, we divide the local result set into multiple parts, and perform the secure set union protocol on every part in parallel. Since each part has a smaller size, our optimization can reduce the idle wait and improve the performance.

Complexity Analysis. For secure summation, the time complexity to solve the linear equations is $O(n^3)$, with $O(n^2)$ communication cost [19]. For secure comparison, the time complexity of the secure multiplication is $O(n)$. It also has a communication cost of $O(n^2)$ [11]. The time complexity and communication cost of secure set union are both $O(n + |S|)$, where S is the final global set [28].

6 QUERY INTERFACE

This section presents the query interface of Hu-Fu. For easy usability, the interface offers a unified federation view to users (Sec. 6.1) and supports federated spatial queries in SQL (Sec. 6.2).

6.1 Unified Federation View

Hu-Fu's query interface provides a federation view to the query user, while the detailed information of silos is hidden. This allows the user to send queries without worrying about the silo organization and also protects the data security of individual silos.

We implement the unified federation view by extending the schema manager of Calcite [8], a popular query processing framework. In Calcite's schema manager, each table is independent and indivisible. We add silo as an abstraction layer below the table of schema manager. Thus each table contains multiple silo objects, and each object records the identity information of the corresponding silo. The silo identity information is used when executing secure primitives. Specifically, the query rewriter will attach the identifying information of all silo-level tables in the table of schema manager when distributing secure operators. Each silo only executes the corresponding secure primitives if the attached identity information matches the one locally stored.

6.2 Federated Spatial Queries in SQL

Based on the unified federation view, Hu-Fu query interface supports federated spatial queries in SQL by extending the SQL parser of Calcite. The semantics are almost the same as regular SQL queries. Specifically, we add two keywords: `DWithin` and `kNN`.

For example, a federated range counting on a circular range centered at the point p with radius r can be written in SQL as

```
SELECT COUNT(*) FROM F WHERE DWithin(p, F.location, r);
```

where `DWithin`$(p, F.location, r)$ returns whether the distance between p and an object $o \in F$ is shorter than r. A federated kNN join on a dataset R and federation F with k can be written in SQL as

```
SELECT R.id, F.id FROM R JOIN F
ON kNN(R.location, F.location, k);
```

where `kNN`$(R.location, F.location, k)$ indicates whether a spatial object $o' \in F$ is in the kNN set of query point $o \in R$. Other queries can be written as SQL similarly with these two keywords.

7 EVALUATION

This section presents the evaluations of Hu-Fu. We first introduce the experimental setup (Sec. 7.1), and then present the overall performance (Sec. 7.2), scalability (Sec. 7.3) and results with heterogeneous spatial databases across silos (Sec. 7.4).

7.1 Experimental Setup

Datasets. We conduct experiments on the following two datasets, where each spatial object has a location and a unique ID.

- **Multi-company Spatial Data in Beijing (BJ).** This dataset was collected by 10 companies in Beijing, in June 2019, which has 1, 029, 081 spatial objects in total. The locations of these objects fall into an area from 39.5°N ∼ 42.0°N and 115.5°E ∼ 117.2°E. We use the dataset to simulate a real-world federation, where each company can be naturally regarded as a silo. We do not alter the distributions of spatial objects across silos, and only vary the silo number n or query-specific parameters (*e.g.*, k for federated kNN query) during the evaluation.
- **OpenStreetMap (OSM).** This is a popular open dataset to evaluate large-scale spatial analytics [16, 44, 63]. We mainly use this dataset in the scalability test, where we sample 10^4-10^9 spatial objects from the Asia dataset in the OpenStreetMap [41]. Specifically, to simulate the spatial overlaps as in the BJ dataset, we assign a random silo ID for each point in the dataset and make each silo have the same number of data points. Please refer to Appendix E [15] for the experiment on datasets generated by geographical partitions, which shows similar results as Sec. 7.3.

Baselines. We compare Hu-Fu with the following baselines.

- **Public.** It directly collects local results from each silo without any secure operation, and serves as the upper bound of query processing efficiency.
- **SMCQL-GIS & SMCQL-GISext.** It adopts the principles of SMCQL [4], a garbled circuit (GC) based solution for relational data, to support spatial queries. We implement SMCQL-GIS and SMCQL-GISext with ObliVM [36], which is also used in SMCQL and only supports two silos. So we only provide the results of SMCQL-GIS and SMCQL-GISext over two silos. And SMCQL-GISext is a variant of SMCQL-GIS without assuming an honest broker, and uses our secure set union to assemble results.
- **Conclave-GIS & Conclave-GISext.** It adopts the principles of Conclave [56], the state-of-the-art secret sharing (SS) based federation solution for relational data, to support spatial queries. Note that we implement Conclave-GIS and Conclave-GISext with a different SS based library, MP-SPDZ [29], rather than Sharemind [11] in the original Conclave. Because Sharemind is devised for only three silos [7] and it is a commercial library. In contrast, MP-SPDZ is a popular open-source library that supports more than three silos based on secret sharing. And Conclave-GISext is a variant of Conclave-GIS without assuming an honest broker, and uses our secure set union to assemble results.

These secure baselines implement federated spatial queries by exploiting similar queries for relational data in SMCQL or Conclave. Our extensions follow the strategy of having plaintext spatial queries within each silo's database and securely computing the final results. Specifically, for *federated range query*, these baselines execute plaintext range query in each silo and collect the partial results by either the honest broker or our secure set union. For *federated range counting*, they execute plaintext range counting and use secure summation to compute the final result. For *federated kNN query*, we regard it as a top-k query with a user-defined function (UDF). For example, each silo runs plaintext kNN query to compute k candidate neighbors along with their distances to the query object. Then, all n silos securely find the k nearest neighbors among nk candidates. For *federated distance join/kNN join*, we refer to their query plans for join queries and regard a federated distance/kNN join as multiple federated range/kNN queries.

Metrics. We assess the query processing efficiency by two metrics.

- **Running time.** It is the time cost from receiving the query from a user to returning the query result to the user.
- **Communication cost.** It is the total network communication among the user and all silos for this query.

All the experimental results are the average of 50 repetitions.

Environment. We run all experiments on a cluster of 11 machines. Each machine has 32 Intel(R) Xeon(R) Gold 5118 2.30GHz processors and 64GB memory with Ubuntu 18.04 LTS. The network bandwidth between machines is up to 10 GB/s. Among the 11 machines, one is as the user and the honest broker for SMCQL-GIS and Conclave-GIS, and the other 10 are data silos. We use PostgreSQL 10.15 with PostGIS extension as the default spatial database for all silos. To show the support of heterogeneous spatial database systems by Hu-Fu, we also use MySQL 5.7 [60], Sqlite3 with SpatiaLite extension [50], GeoMesa 3.0.0 [27], Simba 1.0 [63] and SpatialHadoop 2.4.3 [16] as different silos, as will be explained in Sec. 7.4. They all use spatial indexes (R-Tree in PostGIS, Simba, SpatialHadoop and MySQL, R*-Tree in SpatiaLite, and Z-Curve in GeoMesa) to speed up plaintext primitives by up to 2042× (in Appendix F [15]).

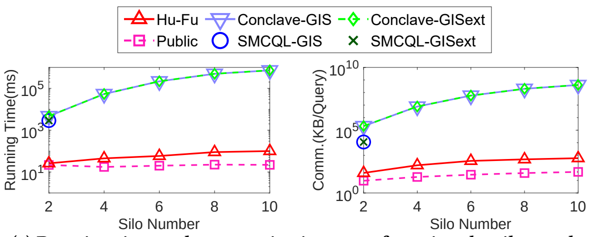
7.2 Results on Real Dataset

This series of experiments compare the efficiency of different methods for all five federated spatial queries on the real dataset BJ. All the query points are randomly sampled from the dataset. We vary the number of silos from 2 to 10, and also test the impact of query-specific parameters. We set k to 16 for federated kNN query and kNN join, and the default query area of federated range query, range counting and distance join as 0.001%, and vary them from 4 to 64 and 0.00001% to 0.1% respectively. The range of these query-specific parameters is aligned with previous study [63]. When evaluating the query-specific parameters, we use 6 silos by default.

7.2.1 *Performance of Federated kNN Query.*

Fig. 5a shows the running time and communication cost of federated kNN query. Hu-Fu is 109.6× to 7, 198.8× faster than SMCQL-GIS and Conclave-GIS, and has 2 to 5 orders of magnitude lower communication cost. When the number of silos increases from 2 to 10, the running time and communication cost of Hu-Fu only increase by up to 2.9× and 13.9×, while those of Conclave-GIS drastically increase by up to 153.3× and 1, 884.3×. Both metrics of Hu-Fu increase because the secure comparison and secure set union used in this query grow linearly with the silo number. Compare with Conclave-GIS and SMCQL-GIS, the running time and communication cost of Conclave-GISext and SMCQL-GISext increase marginally (less than 20 ms and 200 KB, respectively), which shows that our secure set union can efficiently assemble query results without an honest broker.

We also vary k from 4 to 64 and plot the running time and communication cost in Fig. 5b. As k increases from 4 to 64, the running time and communication cost of Hu-Fu only increase by 0.1× and 1.1×, while those of Conclave-GIS increase by 51.3× and 50.7×. The impact of k is less obvious than the silo number on Hu-Fu, because only the secure set union is linearly dependent on k. Again, the


(a) Running time and communication cost of varying the silo number

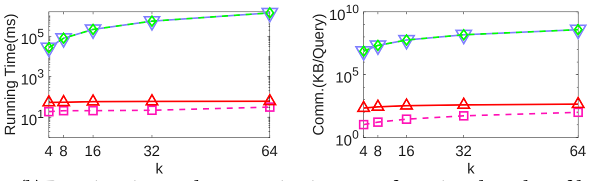

(b) Running time and communication cost of varying the value of k

Figure 5: Performance of federated kNN query.

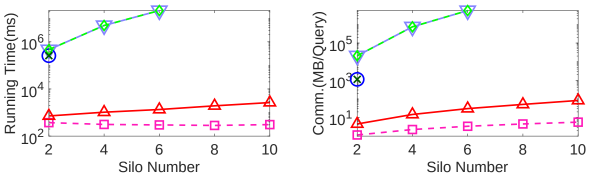
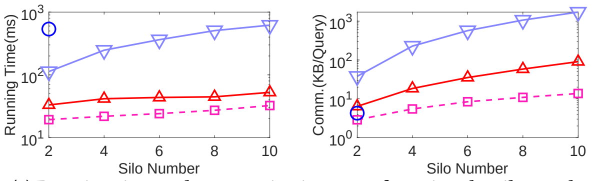


Figure 6: Performance of federated kNN join.


(a) Running time and communication cost of varying the silo number

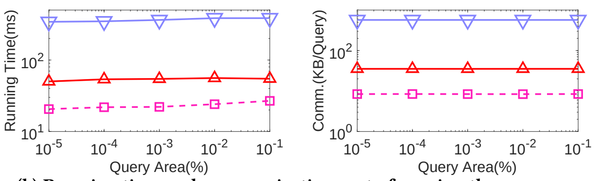

(b) Running time and communication cost of varying the query area

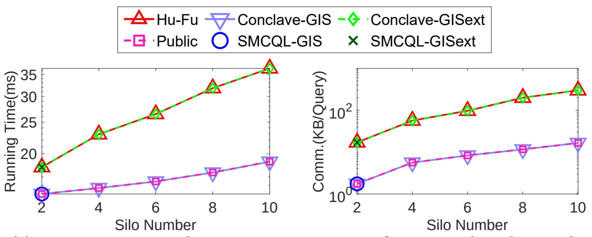
Figure 7: Performance of federated range counting.

efficiency of Conclave-GISext is similar to that of Conclave-GIS. The drastic increase in running time and communication cost of Conclave-GIS and Conclave-GISext is expected because it involves many secure primitives that are time-consuming.

Recall that we apply differential privacy (DP) to accelerate kNN queries (see Sec. 4.5). To prove the gain of the optimization, we list the running time and communication cost with and without DP in Table 3. With DP, the running time is reduced by up to 19.6%, and the communication cost by up to 47.7%. Compared with the improvement, the overhead of injecting the DP noise is very marginal, which takes 2 μs time cost and less than 1 KB communication cost when processing one federated kNN query. Such a notable improvement is because the complexity of DP noise injection is $O(1)$ and the summation only requires for transmission of n integers, while a secure comparison has $O(n)$ time complexity and $O(n^2)$ communication cost.

7.2.2 *Performance of Federated kNN Join.*

Fig. 6 shows the performance of federated kNN join. The results of Conclave-GIS and Conclave-GISext with over 8 silos are omitted since they incur over


(a) Running time and communication cost of varying the silo number

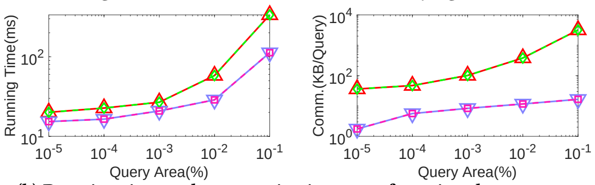

(b) Running time and communication cost of varying the query area

Figure 8: Performance of federated range query.

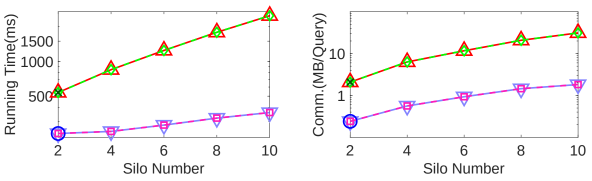


Figure 9: Performance of federated distance join.

Table 3: Ablation of DP optimization for federated kNN query.

Silo Number		2	4	6	8	10
Running Time (ms)	Hu-Fu	26.1	45.1	58.6	89.6	100.5
	Hu-Fu without DP	26.9	50.3	72.9	107.0	116.9
Comm. (KB)	Hu-Fu	39.4	160.8	357.7	475.1	588.3
	Hu-Fu without DP	58.5	234.8	493.0	784.0	1125.2

6 hours for a single query. Hu-Fu is still the most efficient, which is up to 360.2×/15, 814.2× faster than SMCQL-GIS/Conclave-GIS with 247.8×/185, 151.0× lower communication cost. The running time and communication cost of SMCQL-GISext and Conclave-GISext slightly increase over SMCQL-GIS and Conclave-GIS. The impact of k is similar to federated kNN query (see Appendix B [15]).

7.2.3 *Performance of Federated Range Counting.*

Fig. 7 shows the results of federated range counting. This query only returns the counting result and thus does not need a secure set union to protect data ownership. Hence, we exclude SMCQL-GISext and Conclave-GISext since they only differ from SMCQL-GIS and Conclave-GIS with an extra secure set union, which is unnecessary in this query. Hu-Fu is up to 15.2× faster than SMCQL-GIS with a slightly higher communication cost (within 7 KB). Considering the increasing network bandwidth, the gap in communication cost is acceptable. Compared with Conclave-GIS, Hu-Fu is up to 10.8× faster with 17.9× lower communication cost. The running time and communication cost of Hu-Fu increase by 0.6× and 13.2× respectively when silo number increases to 10, mainly due to the secure summation.

We also demonstrate the impact of the query area on query efficiency in Fig. 7b. As is shown, the running time of all methods is relatively stable. It is expected because secure operations are the bottleneck of running time whereas the larger query area only increases the running time of plaintext operations.

7.2.4 *Performance of Federated Range Query.*

Fig. 8 illustrates the results of federated range query. The efficiency of SMCQL-GIS and Conclave-GIS is the same as Public (*i.e.* the non-secure baseline), because they both rely on an honest broker to securely collect partial answers in each silo without leaking them to any others. Under this assumption, all systems can be reduced to Public, which uses a server (*e.g.*, an honest broker in SMCQL-GIS and a center server in Public) to directly collect local range query result from each silo. For example, Hu-Fu with an honest broker also has the same efficiency as Public (see Appendix D [15]). Under a more general setting without this assumption, Hu-Fu, SMCQL-GISext and Conclave-GISext have the same efficiency because they all use our secure set union for results assembling. The use of secure set union only leads to a marginal increase in running time (within 250 ms) and communication cost (lower than 3.1 MB) over Public. Note that the order of increase in running time and communication cost matches the complexity analysis for the secure set union in Sec. 5.2, which grows linearly with the silo number and the amount of data returned. As shown in Fig. 8b, when the query area expands, all methods have a higher running time and communication cost, due to the increase of the number of spatial objects in the final result.

7.2.5 *Performance of Federated Distance Join.*

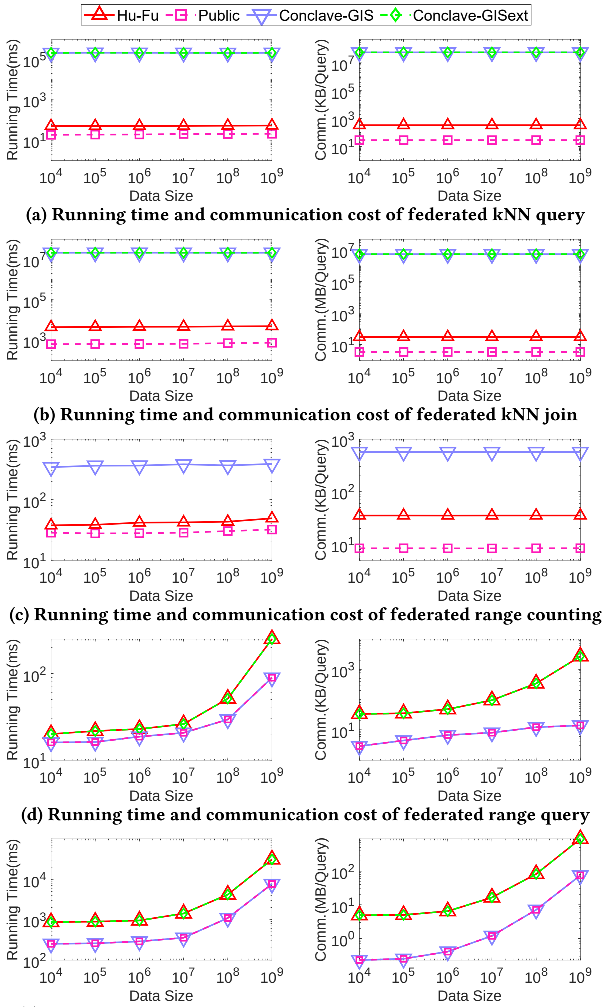
Fig. 9 presents the performance of federated distance join. Note that all the methods treat federated distance join as multiple independent federated range queries, where the total number of these range queries is $|R| = 100$ in this test. Thus, it is reasonable that the ranking of all the methods is similar to that in federated range query (see Fig. 8). The impact of query area is similar to federated range query (see Appendix B [15]).

Takeaways. Overall, Hu-Fu is up to $15,814.2\times$ faster than SMCQL-GIS and Conclave-GIS, with up to 5 orders of magnitude lower communication cost. The efficiency gain of Hu-Fu over the baselines is more notable in federated kNN query, kNN join, and range counting, which is at least $2.4\times$ faster in running time and $4.9\times$ lower in communication cost than Conclave-GIS. SMCQL-GIS and Conclave-GIS are more efficient in federated range query and distance join, because these baselines are reduced to Public and need no secure operation with the honest broker. Note that for federated range query and distance join, Hu-Fu achieves the same efficiency as SMCQL-GISext and Conclave-GISext, the variants of SMCQL-GIS and Conclave-GIS without an honest broker.

7.3 Results on Scalability Test

In this experiment, we scale the total number of spatial objects from 10^4 to 10^9 over OSM dataset to assess the scalability of Hu-Fu. Other parameters are set to the default values as in Sec. 7.2. For example, the number of silos is 6, $k = 16$ for federated kNN query and kNN join, and the query area for federated range query, range counting and distance join is 0.001%. Recall that SMCQL-GIS and SMCQL-GISext only support two silos and are excluded since 6 silos are used in this test. The running time and communication cost on the five spatial queries are shown in Fig. 10.

For a fixed data size, we observe that Hu-Fu is notably more efficient than Conclave-GIS and Conclave-GISext on federated kNN query, kNN join and range counting (see Fig. 10a-10c). For federated range query and distance join, Conclave-GIS behaves the same as Public due to the honest broker, while Hu-Fu achieves the same efficiency as Conclave-GISext, which requires no honest broker.



(a) Running time and communication cost of federated kNN query

(b) Running time and communication cost of federated kNN join

(c) Running time and communication cost of federated range counting

(d) Running time and communication cost of federated range query

(e) Running time and communication cost of federated distance join

Figure 10: Scalability test of federated spatial queries.

We are more interested in the efficiency with the increase of data size. We observe that the efficiency of federated kNN query, kNN join and range counting is insensitive to the increase of the data size. This is because the increase of data size mainly affects the time cost of plaintext primitives, which only accounts for a small portion (due to efficient indexes in each silo) in the running time. In contrast, the running time and communication cost of federated range query and distance join notably increase with the increase of the data size because more spatial objects are retrieved in each silo, which leads to a higher cost for both plaintext range query and secure set union.

Takeaways. Hu-Fu trivially scales with data size for federated kNN query, kNN join and range counting because these queries are relatively insensitive to data size. Both metrics of Hu-Fu increase with the data size for federated range query and distance join, yet Hu-Fu is still reasonably efficient for them on large-scale data. For example, in Hu-Fu, a federated range query takes 250 ms running time and 2.6 MB communication cost on the data size of 10^9.

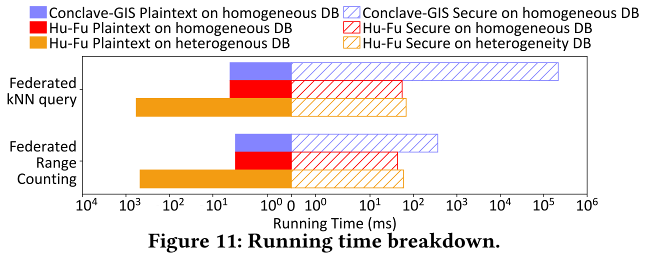


Figure 11: Running time breakdown.

7.4 Results on Heterogeneous Silos

This experiment aims to demonstrate the feasibility of Hu-Fu on heterogeneous spatial databases. Specifically, we use 6 different databases for each silo on the BJ dataset: PostGIS [46], MySQL [60], SpatiaLite [50], Simba [63], GeoMesa [27], and SpatialHadoop [16]. Other parameters are set as the default values as in Sec. 7.2.

Fig. 11 plots the running time breakdown *i.e.* plaintext vs. secure primitives for radius-unknown (*i.e.* federated kNN query) and radius-known (*i.e.* federated range counting) queries (see Appendix C [15] for more results). We make the following observations.

- Given homogeneous underlying spatial databases (PostGIS), our Hu-Fu significantly reduces the running time of secure primitives *e.g.*, $3,935.4\times$ compared with Conclave-GIS for federated kNN query. Such acceleration in secure primitives is the primary contributor to Hu-Fu's gain in running time.
- Heterogeneous underlying spatial databases affect the running time. Specifically, the running time of plaintext primitives is limited by the slowest spatial database, which may increase the overall query processing time. In this experiment, the running time of plaintext primitives notably increases from 4 ms to 579 ms when replacing PostGIS with heterogeneous databases (where SpatiaLite and MySQL are the slowest), which takes even longer than the secure primitives in Hu-Fu. The running time of secure primitives also marginally increases due to idle waiting for the local results from the slowest silo.

Takeaways. Hu-Fu functions with silos running heterogeneous databases. Although Hu-Fu dramatically speeds up the secure primitives in a federated spatial query, the efficiency of plaintext primitives in each silo's databases may affect the overall running time. Particularly, the time cost of plaintext primitives can be limited by the slowest database in the federation. To unleash the full potential of Hu-Fu, fast spatial databases in each silo are recommended.

8 RELATED WORK

Distributed spatial database systems are popular solutions to query processing on big spatial data. These systems improve query processing via data partition and indexing techniques (*e.g.*, R-tree [47]) in Hadoop (*e.g.*, SpatialHadoop [16] and Hadoop-GIS [2]) or Spark (*e.g.*, Simba [63], GeoSpark [67], and LocationSpark [51]). However, the data partition techniques are inapplicable in a data federation since the entire data is held by the autonomous data silos. Moreover, security is not the major concern in these systems.

Past studies of secure spatial query processing mainly focus on encrypted databases [25], where data is encrypted and stored in a third-party platform (*e.g.*, a cloud platform) to process queries securely. For example, existing work [18, 30, 61, 65] study the secure kNN query on encrypted databases and prior studies [58, 62] focus on securely processing range queries. In these studies, a data owner outsources its data and hence the sensitive data is encrypted before being uploaded to a third party. Intuitively, homomorphic encryption techniques (*e.g.*, Paillier [43] and SEAL [38]) are used to guarantee security. Different from this setting, in a data federation, data silos autonomously manage their own data and hence do not need to encrypt their own data and upload it to a third party.

Rather than the general distributed databases or outsourced databases, our work is more aligned with the problem settings of federated databases and data federation, where the entire dataset is held in multiple autonomous databases. The research on federated databases dates back to 1979 (see surveys [26, 49]). Early efforts focused on finding solutions to access data in autonomous databases [42], while recent studies on federated databases support diverse data types, *e.g.*, on federated graph databases [57]. Note that the autonomous database here means that data can be only managed by its held silo which is different from a self-driving database [32, 33].

Data federation is an emerging concept developed from federated databases. It shares a similar architecture with federated databases. Yet, the *major difference* is that a data federation imposes certain secure requirements during query processing, while a federated database does not. For example, SMCQL [4] is the first secure query processing solution over a data federation and Conclave [56] is the state-of-the-art solution. Wang *et al.* [59] explored join-aggregate queries over a data federation of two silos and Ge *et al.* [23] studied secure functional dependency discovery in a data federation. All these studies adopt SMC techniques to achieve secure query processing for *relational* data with *exact* results.

Other studies investigate *approximate* query processing over a *relational* data federation. For example, Shrinkwrap [5], SAQE [6] and Cryptϵ [14] use differential privacy to trade off between accuracy and efficiency in query processing. In contrast, we focus on *exact* query processing, since accurate results can be crucial for spatial applications like contact tracing [22].

In short, our work is inspired by the emerging trend of secure query processing over a data federation, yet focuses on spatial queries with exact results. Our Hu-Fu significantly improves the efficiency of federated spatial queries over the extensions of SMCQL [4] and Conclave [56], the state-of-the-arts for relational data.

9 CONCLUSION

In this paper, we propose the first system Hu-Fu for efficient and secure spatial queries over a data federation. Existing solutions are inefficient to process such queries due to excessive secure distance operations and the usage of general-purpose secure multi-party computation (SMC) libraries for implementing secure operators. To overcome the inefficiency, we design a novel query rewriter to decompose the spatial queries into as many plaintext operators and as few secure operators as possible. In particular, our secure operators involve no distance operation and have dedicated implementations faster than general-purpose SMC libraries. Moreover, Hu-Fu supports heterogeneous spatial databases (*e.g.*, PostGIS, Simba, GeoMesa, and SpatialHadoop), as well as query input in native SQL. Finally, extensive experiments show that Hu-Fu is up to 4 orders of magnitude faster and takes 5 orders of magnitude lower communication cost than the state-of-the-arts. In the future study, we plan to support more spatial queries and analytics in Hu-Fu, *e.g.*, spatial keyword search.

REFERENCES

[1] Acxiom. 2021. https://www.acxiom.com/

[2] Ablimit Aji, Fusheng Wang, Hoang Vo, Rubao Lee, Qiaoling Liu, Xiaodong Zhang, and Joel H. Saltz. 2013. Hadoop-GIS: A High Performance Spatial Data Warehousing System over MapReduce. *PVLDB* 6, 11 (2013), 1009–1020.

[3] AMAP. 2021. https://www.amap.com

[4] Johes Bater, Gregory Elliott, Craig Eggen, Satyender Goel, Abel N. Kho, and Jennie Rogers. 2017. SMCQL: Secure Query Processing for Private Data Networks. *PVLDB* 10, 6 (2017), 673–684.

[5] Johes Bater, Xi He, William Ehrich, Ashwin Machanavajjhala, and Jennie Rogers. 2018. ShrinkWrap: Efficient SQL Query Processing in Differentially Private Data Federations. *PVLDB* 12, 3 (2018), 307–320.

[6] Johes Bater, Yongjoo Park, Xi He, Xiao Wang, and Jennie Rogers. 2020. SAQE: Practical Privacy-Preserving Approximate Query Processing for Data Federations. *PVLDB* 13, 11 (2020), 2691–2705.

[7] Fattaneh Bayatbabolghani and Marina Blanton. 2018. Secure Multi-Party Computation. In *CCS*. 2157–2159.

[8] Edmon Begoli, Jesús Camacho-Rodríguez, Julian Hyde, Michael J. Mior, and Daniel Lemire. 2018. Apache Calcite: A Foundational Framework for Optimized Query Processing Over Heterogeneous Data Sources. In *SIGMOD*. 221–230.

[9] Paolo Bellavista, Luca Foschini, and Alessio Mora. 2021. Decentralised Learning in Federated Deployment Environments: A System-Level Survey. *ACM Computing Surveys* 54, 1 (2021), 15:1–15:38.

[10] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. 1988. Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation (Extended Abstract). In *STOC*. 1–10.

[11] Dan Bogdanov, Sven Laur, and Jan Willemson. 2008. Sharemind: A Framework for Fast Privacy-Preserving Computations. In *ESORICS*. 192–206.

[12] David Chaum, Claude Crépeau, and Ivan Damgård. 1988. Multiparty Unconditionally Secure Protocols (Extended Abstract). In *STOC*. 11–19.

[13] Yong Cheng, Yang Liu, Tianjian Chen, and Qiang Yang. 2020. Federated learning for privacy-preserving AI. *Communications of the ACM* 63, 12 (2020), 33–36.

[14] Amrita Roy Chowdhury, Chenghong Wang, Xi He, Ashwin Machanavajjhala, and Somesh Jha. 2020. Cryptϵ: Crypto-Assisted Differential Privacy on Untrusted Servers. In *SIGMOD*. 603–619.

[15] Hu-Fu: Efficient and Secure Spatial Queries over Data Federation (Technical Report). 2021. https://github.com/BUAA-BDA/Hu-Fu/blob/dev/Hu-Fu_Technical_Report.pdf

[16] Ahmed Eldawy and Mohamed F. Mokbel. 2015. SpatialHadoop: A MapReduce framework for spatial data. In *ICDE*. 1352–1363.

[17] Ahmed Eldawy and Mohamed F. Mokbel. 2017. The Era of Big Spatial Data. *PVLDB* 10, 12 (2017), 1992–1995.

[18] Yousef Elmehdwi, Bharath K. Samanthula, and Wei Jiang. 2014. Secure k-nearest neighbor query over encrypted data in outsourced environments. In *ICDE*. 664–675.

[19] Fatih Emekçi, Ozgur D. Sahin, Divyakant Agrawal, and Amr El Abbadi. 2007. Privacy preserving decision tree learning over multiple parties. *Data & Knowledge Engineering* 63, 2 (2007), 348–361.

[20] David Evans, Vladimir Kolesnikov, and Mike Rosulek. 2018. A Pragmatic Introduction to Secure Multi-Party Computation. *Foundations and Trends in Privacy and Security* 2, 2-3 (2018), 70–246.

[21] Experian. 2021. https://www.experian.com/

[22] Luca Ferretti, Chris Wymant, Michelle Kendall, Lele Zhao, Anel Nurtay, Lucie Abeler-Dörner, Michael Parker, David Bonsall, and Christophe Fraser. 2020. Quantifying SARS-CoV-2 transmission suggests epidemic control with digital contact tracing. *Science* 368, 6491 (2020), eabb6936.

[23] Chang Ge, Ihab F. Ilyas, and Florian Kerschbaum. 2019. Secure Multi-Party Functional Dependency Discovery. *PVLDB* 13, 2 (2019), 184–196.

[24] Oded Goldreich. 2009. *Foundations of cryptography: volume 2, basic applications*. Cambridge university press.

[25] Hakan Hacigümüs, Balakrishna R. Iyer, Chen Li, and Sharad Mehrotra. 2002. Executing SQL over encrypted data in the database-service-provider model. In *SIGMOD*. 216–227.

[26] Dennis Heimbigner and Dennis McLeod. 1985. A Federated Architecture for Information Management. *ACM Transactions on Information Systems* 3, 3 (1985), 253–278.

[27] James N. Hughes, Andrew Annex, Christopher N. Eichelberger, Anthony Fox, Andrew Hulbert, and Michael Ronquest. 2015. GeoMesa: a distributed architecture for spatio-temporal fusion. In *SPIE*. 94730F.

[28] Pawel Jurczyk and Li Xiong. 2011. Information Sharing across Private Databases: Secure Union Revisited. In *SocialCom/PASSAT*. 996–1003.

[29] Marcel Keller. 2020. MP-SPDZ: A Versatile Framework for Multi-Party Computation. In *CCS*. 1575–1590.

[30] Manish Kesarwani, Akshar Kaul, Prasad Naldurg, Sikhar Patranabis, Gagandeep Singh, Sameep Mehta, and Debdeep Mukhopadhyay. 2018. Efficient Secure k-Nearest Neighbours over Encrypted Data. In *EDBT*. 564–575.

[31] Jinkyu Kim, Heonseok Ha, Byung-Gon Chun, Sungroh Yoon, and Sang K. Cha. 2016. Collaborative analytics for data silos. In *ICDE*. 743–754.

[32] Jan Kossmann, Martin Boissier, Alexander Dubrawski, Fabian Heseding, Caterina Mandel, Udo Pigorsch, Max Schneider, Til Schniese, Mona Sobhani, Petr Tsayun, Katharina Wille, Michael Perscheid, Matthias Uflacker, and Hasso Plattner. 2021. A Cockpit for the Development and Evaluation of Autonomous Database Systems. In *ICDE*. 2685–2688.

[33] Guoliang Li, Xuanhe Zhou, Ji Sun, Xiang Yu, Yue Han, Lianyuan Jin, Wenbo Li, Tianqing Wang, and Shifu Li. 2021. openGauss: An Autonomous Database System. *PVLDB* 14, 12 (2021), 3028–3041.

[34] Ninghui Li, Min Lyu, Dong Su, and Weining Yang. 2016. *Differential Privacy: From Theory to Practice.* Morgan & Claypool Publishers.

[35] Xijun Li, Weilin Luo, Mingxuan Yuan, Jun Wang, Jiawen Lu, Jie Wang, Jinhu Lü, and Jia Zeng. 2021. Learning to Optimize Industry-Scale Dynamic Pickup and Delivery Problems. In *ICDE*. 2511–2522.

[36] Chang Liu, Xiao Shaun Wang, Kartik Nayak, Yan Huang, and Elaine Shi. 2015. ObliVM: A Programming Framework for Secure Computation. In *S&P*. 359–376.

[37] Xiaoyuan Liu, Ni Trieu, Evgenios M. Kornaropoulos, and Dawn Song. 2020. BeeTrace: A Unified Platform for Secure Contact Tracing that Breaks Data Silos. *IEEE Data Engineering Bulletin* 43, 2 (2020), 108–120.

[38] Microsoft SEAL (release 3.6). 2021. https://github.com/Microsoft/SEAL

[39] China Mobile. 2021. https://www.chinamobileltd.com

[40] State of California Department of Justice. 2018. California Consumer Privacy Act (CCPA). https://oag.ca.gov/privacy/ccpa

[41] OpenStreetMap. 2021. https://www.openstreetmap.org

[42] M. Tamer Özsu and Patrick Valduriez. 2020. *Principles of Distributed Database Systems, 4th Edition.* Springer.

[43] Pascal Paillier. 1999. Public-Key Cryptosystems Based on Composite Degree Residuosity Classes. In *EUROCRYPT*. 223–238.

[44] Varun Pandey, Andreas Kipf, Thomas Neumann, and Alfons Kemper. 2018. How Good Are Modern Spatial Analytics Systems? *PVLDB* 11, 11 (2018), 1661–1673.

[45] European Parliament and The Council of the European Union. 2016. The general data protection regulation (GDPR). https://eugdpr.org

[46] PostGIS. 2021. https://www.postgis.org/

[47] Hanan Samet. 2006. *Foundations of multidimensional and metric data structures*. Academic Press.

[48] Facebook scandal: Who is selling your personal data? 2018. https://www.bbc.com/news/technology-44793247

[49] Amit P. Sheth and James A. Larson. 1990. Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases. *ACM Computing Surveys* 22, 3 (1990), 183–236.

[50] SpatiaLite. 2021. http://live.osgeo.org/en/overview/spatialite_overview.html

[51] MingJie Tang, Yongyang Yu, Qutaibah M. Malluhi, Mourad Ouzzani, and Walid G. Aref. 2016. LocationSpark: A Distributed In-Memory Data Management System for Big Spatial Data. *PVLDB* 9, 13 (2016), 1565–1568.

[52] China Telecom. 2021. http://www.chinatelecom-h.com/

[53] Numerous Beijing Taxi Brands to Collectively Connect to Amap's Ride-hailing Platform to Enable Online Operation. 2021. https://aag.cc/newsinfo/517126.html

[54] Communication travel card. 2021. https://xc.caict.ac.cn/

[55] Paul Voigt and Axel Von dem Bussche. 2017. *The EU General Data Protection Regulation (GDPR): A Practical Guide.* Vol. 10. Springer International Publishing.

[56] Nikolaj Volgushev, Malte Schwarzkopf, Ben Getchell, Mayank Varia, Andrei Lapets, and Azer Bestavros. 2019. Conclave: secure multi-party computation on big data. In *EuroSys*. 3:1–3:18.

[57] Xuan-Son Vu, Addi Ait-Mlouk, Erik Elmroth, and Lili Jiang. 2019. Graph-based Interactive Data Federation System for Heterogeneous Data Retrieval and Analytics. In *WWW*. 3595–3599.

[58] Peng Wang and Chinya V. Ravishankar. 2013. Secure and efficient range queries on outsourced databases using Rp-trees. In *ICDE*. 314–325.

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

[60] MySQL (with supports to GIS). 2021. https://www.mysql.com/

[61] Wai Kit Wong, David Wai-Lok Cheung, Ben Kao, and Nikos Mamoulis. 2009. Secure kNN computation on encrypted databases. In *SIGMOD*. 139–152.

[62] Songrui Wu, Qi Li, Guoliang Li, Dong Yuan, Xingliang Yuan, and Cong Wang. 2019. ServeDB: Secure, Verifiable, and Efficient Range Queries on Outsourced Database. In *ICDE*. 626–637.

[63] Dong Xie, Feifei Li, Bin Yao, Gefei Li, Liang Zhou, and Minyi Guo. 2016. Simba: Efficient In-Memory Spatial Analytics. In *SIGMOD*. 1071–1085.

[64] Qiang Yang, Yang Liu, Tianjian Chen, and Yongxin Tong. 2019. Federated Machine Learning: Concept and Applications. *ACM Transactions on Intelligent Systems and Technology* 10, 2 (2019), 12:1–12:19.

[65] Bin Yao, Feifei Li, and Xiaokui Xiao. 2013. Secure nearest neighbor revisited. In *ICDE*. 733–744.

[66] Jieping Ye. 2019. Transportation: A Data Driven Approach. In *SIGKDD*. 3183.

[67] Jia Yu, Jinxuan Wu, and Mohamed Sarwat. 2015. GeoSpark: a cluster computing framework for processing large-scale spatial data. In *SIGSPATIAL*. 70:1–70:4.

[68] Yu Zheng, Xing Xie, and Wei-Ying Ma. 2010. GeoLife: A Collaborative Social Networking Service among User, Location and Trajectory. *IEEE Data Engineering Bulletin* 33, 2 (2010), 32–39.

A SECURITY ANALYSIS

Security proof. The security analysis of our query rewriter is based on the composition lemma in [24] (Sec. 7.3.1). The lemma states that given a secure protocol $\pi^{g|f}$ that can securely compute g based on a plaintext query f and a secure protocol π^f for operation f, the operation g can be securely computed by executing protocol $\pi^{g|f}$ but substitutes every plaintext query f with an execution of π^f. We prove the security of query rewriter for radius-known and radius-unknown queries separately.

- **Security of Radius-Known Queries.** Federated range query, range counting and distance join belong to *radius-known queries*. For *federated range query* and *distance join*, the secure protocol $\pi^{g|f}$ is the secure set union (Def. 8) based on the query results of one or multiple plaintext range queries over the federation. The protocol π^f corresponds to the plaintext range query (Def. 5). The protocol π^f is secure, since the plaintext range query only occurs in each silo, which will not leak any information to other silos. From the composition lemma, federated range query and distance join are secure against semi-honest adversaries. Federated range counting is also secure based on a similar proof (*i.e.* replacing $\pi^{g|f}$ with the secure summation in Def. 6 and π^f with the plaintext range counting in Def. 5).
- **Security of Radius-Unknown Queries.** The federated kNN query and kNN join belong to *radius-unknown queries*. For federated kNN query (Alg. 1), let protocol π^f be the secure comparison between k and the local counts. The security of π^f can be proved in the same way as before. Denote protocol $\pi^{g|f}$ as Alg. 1 based on the plaintext comparison query. We then show protocol $\pi^{g|f}$ is also secure. Assuming semi-honest adversaries, each silo can simulate its view of the execution of the protocol. Specifically, knowing the range $[l, u]$ used at the beginning of a round, each silo can compute *thres* used in that round. If *thres* is the same as the final radius, it concludes that the protocol must have ended in this round. Otherwise, it simply updates the range to that side of *thres* which contains the final radius. Along with the knowledge of the initial range, this shows that each silo can simulate the execution of the protocol, which means the protocol $\pi^{g|f}$ is secure. Thus, our Alg. 1 is secure against semi-honest adversaries based on the composition lemma. The security of federated kNN join can be proved similarly.

Table 4: A case study on hacking federated range counting in Hu-Fu.

Colluded silos	Attack success rate	Running time
1	0	No result for 2 months
2	0	No result for 2 months
3	0	No result for 2 months

Case Study. We have also conducted a case study to show how Hu-Fu defenses against the semi-honest adversaries when processing federated range counting.

- **Experimental Setup.** The case study is based on the BJ dataset with 6 silos (*i.e.* our default setting) and we vary the number m of colluded adversaries (*i.e.* silos) from 1 to 3. Then, when processing federated range counting, the attack will occur at the only secure operator (*i.e.* secure summation [19]). To attack the secure summation, we use the method introduced in [19]. In general, a secure operator successfully defenses against the attack, if adversaries cannot get any feasible result after a long time (*e.g.*, two months in our experiment). Furthermore, the other experiment parameters are the same as those in Sec. 7.2.
- **Experimental Result.** As shown in Table 4, even when 50% of the silos collude, they cannot infer any feasible result even after two months' cracking. Note that it is commonly assumed in existing work [10, 12] that the portion of colluded adversaries is less than 50%. Based on the computational security [20], this result proves Hu-Fu is hard to attack.

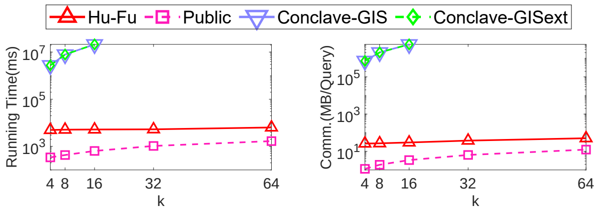


Figure 12: Performance of federated kNN join when varying k.

B EXPERIMENT ON REAL DATASET

Fig. 12 illustrates the results of federated kNN join on the BJ dataset when varying the value of k. Note that partial results of Conclave-GIS and Conclave-GISext (when $k > 16$) are not provided, since they take longer than 6 hours to process this query. As is shown, Hu-Fu is at least 553× faster and takes 27,404× lower communication cost than Conclave-GIS. As the parameter k of federated kNN join increases from 4 to 64, the running time and communication cost of Hu-Fu only increase by 28% and 48% respectively, because the parameter k has relatively marginal impact on the federated kNN query and a federated kNN join is decomposed into multiple federated kNN queries for all the compared solutions.

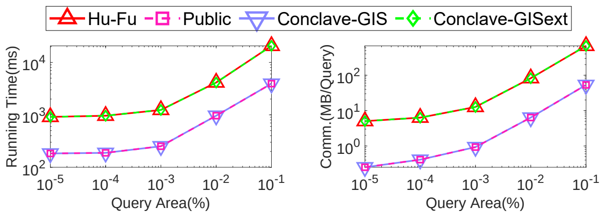


Figure 13: Performance of federated distance join when varying query area.

Fig. 13 presents the performance of federated distance join on the BJ dataset when varying the query area. The result of federated distance join when varying the query area shows a similar pattern with that of federated range query. This is because a federated distance join is decomposed into multiple federated range queries for all the compared solutions. The increase of both running time and communication cost is caused by the increase of the number of retrieved spatial objects.

The SMCQL-GIS and SMCQL-GISext are excluded from both Fig. 12 and Fig. 13 because they only support two silos, and the tests in this section are conducted on six silos.

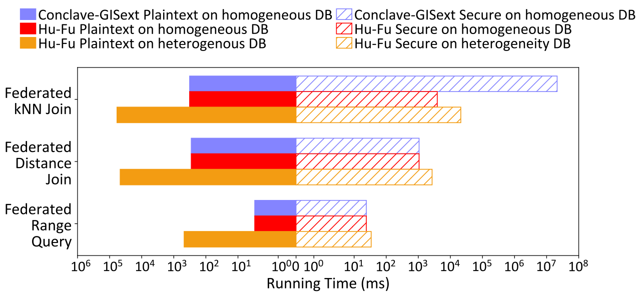


Figure 14: Running time breakdown.

C EXPERIMENT ON HETEROGENEOUS SILOS

Fig. 14 shows the results of federated range query, distance join and kNN join in the experiment of heterogeneous silos. We can first observe that our Hu-Fu can support all six spatial database systems, including PostGIS [46], MySQL [60], SpatiaLite [50], Simba [63], GeoMesa [27], and SpatialHadoop [16]. In addition, for all federated spatial queries in Fig. 14, we observe the running time of Hu-Fu becomes longer, compared with the results of homogeneous silos (*i.e.*, PostGIS in this test). This is because the time cost of Hu-Fu's plaintext primitives (*i.e.* plaintext range query) increases due to the slowest spatial database system. We can also observe that the experimental patterns of federated distance join and federated kNN join are similar to those of federated range query and federated kNN query. This is because a federated distance/kNN join is decomposed into a series of federated range/kNN query. Besides, we also recommend that all silos can use more efficient spatial database systems to further speed up the federated spatial queries.

D EXPERIMENT ON HU-FU WITH AN HONEST BROKER

In the following, we conduct an experiment to demonstrate the performance of Hu-Fu with an honest broker (denoted by "Hu-Fu-HB").

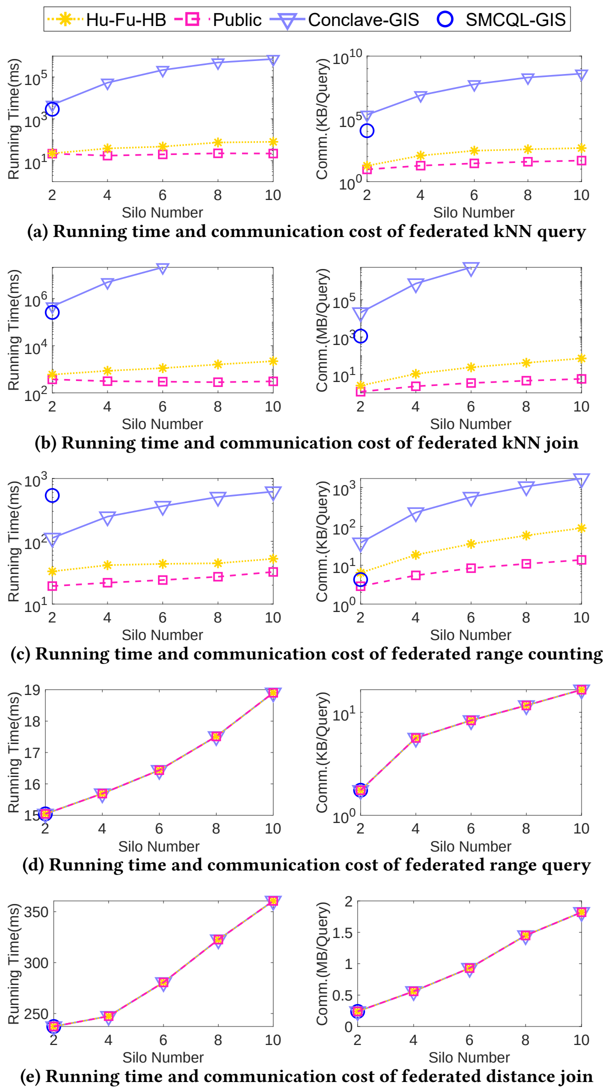
Experimental Setup. This experiment has tested all federated spatial queries on the real dataset BJ. We compare Hu-Fu-HB with SMCQL-GIS/Conclave-GIS in query processing efficiency measured in running time and communication cost. Besides, the other experiment settings are the same as those in Sec. 7.2.

Experimental Result. Fig. 15 and Fig. 16 present the experimental results of Hu-Fu-HB and SMCQL-GIS/Conclave-GIS in terms of running time and communication cost. First, We can observe that Hu-Fu-HB performs the same as SMCQL-GIS and Conclave-GIS on federated range query and distance join. This is because all the compared algorithms (including ours) are simplified to Public due to the honest broker who collects the partial results (*i.e.* sensitive data) of plaintext range query in each silo and is assumed to never reveal them to anyone else. Second, as for federated range counting, kNN query and kNN join, Hu-Fu-HB still outperforms SMCQL-GIS and Conclave-GIS with up to 4 orders of magnitudes faster in running time and 5 orders of magnitudes lower in communication cost. This is because our query writer is more effective to process queries on large-scale spatial data.

Summary. The experiment result illustrates that compared with SMCQL-GIS/Conclave-GIS, Hu-Fu-HB has similar efficiency in federated range query and distance join. Meanwhile, Hu-Fu-HB achieves better efficiency in federated range counting, kNN query and kNN join.

E EXPERIMENT ON GEOGRAPHICALLY PARTITIONED DATASET

To explore the performance of Hu-Fu in the dataset where each silo corresponds to a single country, we have conducted an experiment on a new synthetic dataset (denoted by "*OSM country*").



(a) Running time and communication cost of federated kNN query

(b) Running time and communication cost of federated kNN join

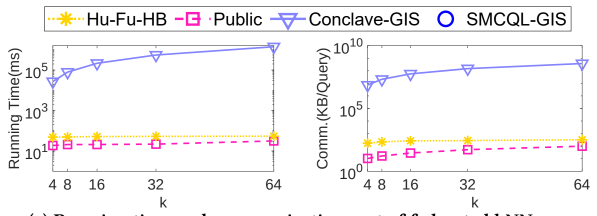
(c) Running time and communication cost of federated range counting

(d) Running time and communication cost of federated range query

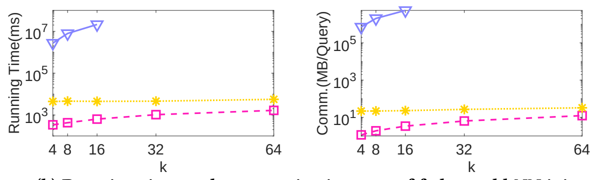
(e) Running time and communication cost of federated distance join

Figure 15: Hu-Fu with an honest broker varying the silo number.

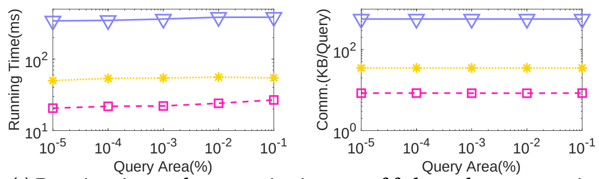
Experimental Setup. The *OSM country* dataset consists of spatial points from six countries in OpenStreetMap [41]. These countries include China, India, Japan, Malaysia, North Korea and South Korea, and each of them corresponds to a single data silo. As shown in Table 5, the volume of data in each silo follows the native data proportion of the corresponding country in OpenStreetMap. For example, the numbers of data points in China, India and Japan are much larger than those of Malaysia, North Korea and South Korea. Here, we vary the total number of all data points from 10^4 to 10^9. Besides, the other experiment parameters are the same as those in Sec. 7.3.



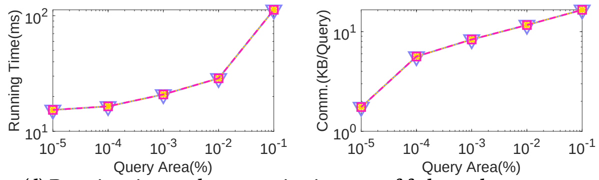
(a) Running time and communication cost of federated kNN query



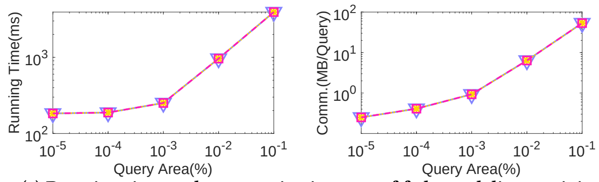
(b) Running time and communication cost of federated kNN join



(c) Running time and communication cost of federated range counting



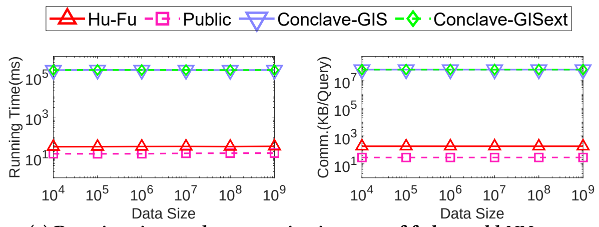
(d) Running time and communication cost of federated range query



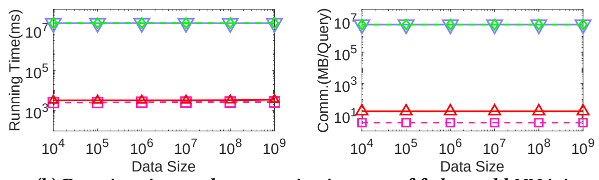
(e) Running time and communication cost of federated distance join

Figure 16: Hu-Fu with an honest broker varying the query-specific parameter.

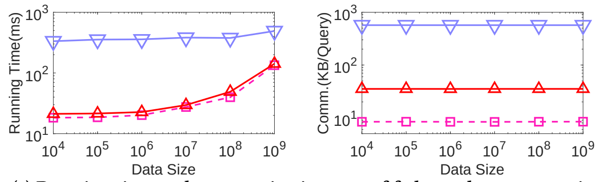
Experimental Result. The results of this experiment are shown in Fig. 17. First, we can observe that in the *OSM country* dataset, Hu-Fu also shows a significant improvement in both running time and communication cost over Conclave-GIS and Conclave-GISext on federated kNN query, kNN join and range counting. For example, Hu-Fu is at least 3 orders of magnitude faster than Conclave-GIS and Conclave-GISext in federated kNN query and kNN join, and costs 5 orders of magnitude lower network communication. Second, Hu-Fu achieves the same efficiency as Conclave-GISext on federated range query and distance join. Overall, the ranking of the compared solutions in terms of either running time or communication cost is consistent with the ranking in Sec. 7.3. Note that we exclude Conclave-GISext from the results of federated range counting as shown in Fig. 17c, because this query does not need the secure set union to protect data ownership in this query.



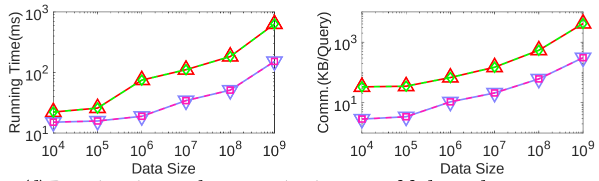
(a) Running time and communication cost of federated kNN query



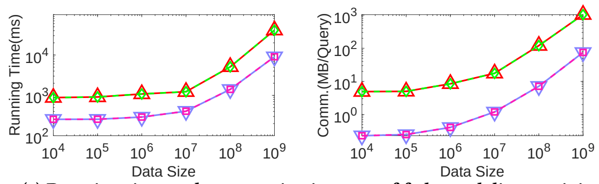
(b) Running time and communication cost of federated kNN join



(c) Running time and communication cost of federated range counting



(d) Running time and communication cost of federated range query



(e) Running time and communication cost of federated distance join

Figure 17: Scalability test on synthetic dataset partitioned by countries.

Summary. The experiment on *OSM country* dataset has a similar result with that in Sec. 7.3. Specifically, Hu-Fu outperforms SMCQL-GISext/Conclave-GISext on federated kNN query, kNN join and range counting. Meanwhile, Hu-Fu performs similarly as SMCQL-GISext/Conclave-GISext on federated range query and distance join.

F EXPERIMENT ON THE IMPROVEMENT BY THE INDEX IN EACH DATA SILO

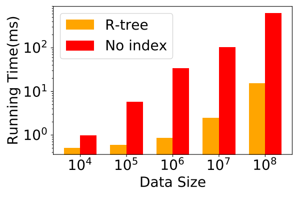
To demonstrate that the federated spatial queries have already been accelerated by local indexes in Hu-Fu, we have conducted a new experiment on the efficiency improvement caused by these local indexes (*i.e.* R-trees in our default setting).

Table 5: Percentage of data of each country in *OSM country*.

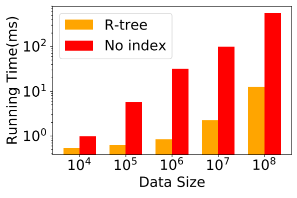
Country	Percentage of data
China	21.7%
India	29.8%
Japan	39.4%
Malaysia	4.2%
North Korea	1.2%
South Korea	3.7%

Experimental Setup. In this experiment, we test the running time of the plaintext spatial queries in Hu-Fu (*i.e.* plaintext range query, plaintext range counting and plaintext kNN query) on PostgreSQL with and without an R-tree. And the plaintext spatial queries are processed on the OSM dataset. Besides, we vary the data size from 10^4 to 10^8 and use the default setting of query area (0.001%) and k (16). The other experiment settings are the same as those in Sec. 7.3.

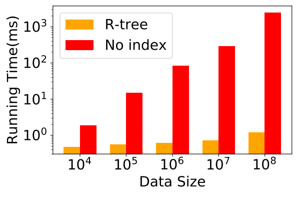
Experimental Result. As shown in Fig. 18, local indexes can improve the efficiency of plaintext spatial queries, especially when the data size is large. Specifically, the local index (R-tree) reduces the running time by up to 40×, 44× and 2042× when processing plaintext range query, range counting, and kNN query, respectively. Moreover, the plaintext spatial queries without local indexes cost even longer time than corresponding federated spatial queries in Hu-Fu. For instance, the running time of processing one federated kNN query by Hu-Fu takes 52 ms when the data size is 10^8, which is even faster than that of the plaintext kNN query without the local index (2465 ms). Thus, the experimental result proves that we have used local indexes in Hu-Fu to speed up the processing of the plaintext operators.



(a) Plaintext range query



(b) Plaintext range counting



(c) Plaintext kNN query

Figure 18: Running time of plaintext spatial queries with/without R-tree.