The VLDB Journal (2025) 34:19
https://doi.org/10.1007/s00778-024-00896-3

SPECIAL ISSUE PAPER

Hu-Fu: efficient and secure spatial queries over data federation

Yongxin Tong[1] · Yuxiang Zeng[1] · Yang Song[1] · Xuchen Pan[1] · Zeheng Fan[1] · Chunbo Xue[1] · Zimu Zhou[2] · Xiaofei Zhang[3] · Lei Chen[4,5,6] · Yi Xu[1] · Ke Xu[1] · Weifeng Lv[1]

Received: 30 June 2024 / Revised: 24 October 2024 / Accepted: 28 December 2024

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 studies on 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. Hu-Fu seamlessly supports five mainstream spatial queries at scale, while ensuring both data and query privacy (*i.e.*, sensitive spatial information of data owners and query users). 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 of them 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, 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.

Keywords Federated database · Spatial database · Query processing · Data privacy

1 Introduction

Efficient processing of spatial queries over large-scale data is essential for a wide spectrum of smart city applications, such as taxi-calling [59] and logistics planning [60]. Although the volume of spatial data continues to grow, it becomes increas-

✉ Yuxiang Zeng
yxzeng@buaa.edu.cn

Yongxin Tong
yxtong@buaa.edu.cn

Yang Song
songyangbuaa@buaa.edu.cn

Xuchen Pan
panxuchen@buaa.edu.cn

Zeheng Fan
fanzh@buaa.edu.cn

Chunbo Xue
xuechunbo@buaa.edu.cn

Zimu Zhou
zimuzhou@cityu.edu.hk

Xiaofei Zhang
xiaofei.zhang@memphis.edu

Lei Chen
leichen@cse.ust.hk

Yi Xu
xuy@buaa.edu.cn

Ke Xu
kexu@buaa.edu.cn

Weifeng Lv
lwf@buaa.edu.cn

[1] State Key Laboratory of Complex & Critical Software Environment and Beijing Advanced Innovation Center for Future Blockchain and Privacy Computing, Beihang University, Beijing, China

[2] Department of Data Science, City University of Hong Kong, Hong Kong SAR, China

[3] The University of Memphis, Memphis, USA

[4] The Hong Kong University of Science and Technology, Hong Kong SAR, China

[5] Hong Kong University of Science and Technology (GZ), Guangzhou, China

[6] HKUST Shenzhen-Hong Kong Collaborative Innovation Research Institute, Shenzhen, China

Published online: 27 January 2025

ingly difficult for these applications to take full advantage of the big spatial data due to the data isolation problem (*a.k.a.* isolated data) [39, 45, 47]. 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 [49]) or commercial reasons.

A promising paradigm to tackle the data isolation problem is to perform *secure* queries over a *data federation* [14], which consists of multiple data owners (*a.k.a.* data silos) who agree on the same schema and manage their own data autonomously. Note that this paradigm differs from conventional federated databases [41] 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) [24]. Yet, only recently did pioneer studies such as SMCQL [14] and Conclave [50] 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 AMAP [3] (GaoDe Map in China) has united over 8 Chinese travel companies into an integrated taxi-calling platform to offer users the taxis resources from all participating companies. A spatial data federation can protect the distribution of taxis' locations of each company (*i.e.*, data silo), which could be a business secret, from leaking to others. This privacy concern for data silos is commonly referred to as *data privacy* [27].

Example 2 During COVID-19, several mobile network operators (*e.g.*, China Mobile [4] and China Telecom [5]) cooperated as a data federation to identify individuals who had contacts with infectious patients through their location data [6]. Executing spatial queries (*e.g.*, range query) over a data federation helps identify contacts of infectious patients across multiple organizations' spatial data without compromising privacy. Here, strict privacy requirements go beyond the data privacy since the locations of patients, which appear in queries, also need protections. The privacy concern for spatial data in queries is referred to as *query privacy* [27].

Due to legal regulations (*e.g.*, GDPR [49]), protecting *data privacy* is now common in real-life scenarios, especially when spatial location implies the travel patterns or personal trajectories of a user. *Query privacy* is equally important, but perhaps gets less attention in existing research on data federation. Query privacy also has numerous real-world applications, such as navigation, location-based social networking, location-based advertising, and POI search [17, 28].

Nevertheless, directly adapting the state-of-the-art data federation solutions [14, 50] 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× 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 [14] and Conclave [50] 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 [14, 50] 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 [38], the SMC library adopted by SMCQL [14], can be accelerated by 15× via dedicated implementations [23].

In this paper, we aim at efficient and secure spatial queries over a data federation, which we call *federated spatial queries*. We mainly study five queries (federated range query, range counting, kNN query, distance join, and kNN join) commonly seen in spatial database research [21, 57] and follow the semi-honest adversary model adopted by previous work [14, 50, 53]. Moreover, we develop a more practical solution than [14, 50] by eliminating the need for an honest broker and supporting more data silos (these studies support at most three data silos whereas we tested up to ten).

To this end, we propose Hu-Fu [7], 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 while minimizing secure distance-related operations without compromising privacy. The decomposition aims to achieve two goals: *(i)* reduce the number of distance-related operations to the minimum, and *(ii)* implement secure operations 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 for the query execution plan to handle the queries of interest. It ensures diverse privacy requirements, as explained in Examples 1 and 2: data privacy only, or both data and query privacy. The drivers provide implementations of secure operators with dedicated SMC protocols and plaintext operators 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.

Contribution. 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, and is also available on GitHub [7].
- 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 [10], Simba [57], GeoMesa [26], SpatiaLite [11], and SpatialHadoop [21].
- Extensive evaluations show that Hu-Fu usually outperforms the state-of-the-arts [14, 50] in efficiency. Compared with two strong baselines, namely SMCQL-GIS and Conclave-GIS, which are extended from SMCQL [14] and Conclave [50] to spatial queries, Hu-Fu is up to 4 orders of magnitude faster and 5 orders of magnitude lower in communication overhead than SMCQL-GIS and Conclave-GIS with the same security level.

Compared with the preliminary version [46] of this work, we have made the following new contributions. *(i)* We expand our scope to a new and challenging setting where both data and query privacy must be preserved. Hu-Fu also provides the corresponding SQL *query interface*. *(ii)* The *query rewriter* is extended and optimized to handle all five spatial queries in this new setting. *(iii)* In *drivers*, two additional secure operators are tailored to fulfill the extra privacy requirement. *(iv)* Extensive evaluations are conducted to show the performance.

Roadmap. 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 the three 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 and highlights the technical challenges when developing Hu-Fu.

2.1 Problem scope

A data federation F ("federation" as short) consists of n data silos $\{F_i\}$ ("silos" as short), where each silo holds massive *spatial objects*. Each spatial object o has a location l_o and (optionally) other attributes. The federation supports *federated spatial queries* over the spatial objects of all silos under the following settings.

- *Spatial Queries*. The federation supports mainstream spatial queries like range query, range counting, kNN query, distance join, and kNN join [40, 57].
- *Autonomous Databases*. Each data silo is an autonomous database that manages (*e.g.*, deletes and inserts its own spatial objects and prohibits sharing its spatial objects in plaintext with the other data silos [14–16, 50].
- *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. This assumption is common in query processing over a data federation [14–16, 50].

Moreover, the query processing methods should consider the following requirements thoughtfully.

- *Efficiency Requirements*. We care about the *running time* and *communication cost* to execute *exact* queries over multiple silos. Short running time is often desirable since real-life applications may process massive queries and expect prompt responses. Minimal communication cost is critical in distributed query processing [41] and secure query processing [27]. Approximate query processing over data federation [16, 20, 61] 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 solutions [14, 50], the storage efficiency, which mainly depends on silos themselves, is not our primary concern.
- *Privacy Requirements*. We consider *two* different types of privacy requirements [24, 27].
  1. *Data privacy*: each data silo should not deduce any sensitive data from others, and no additional sensitive data should be revealed to the query user, except for the final query answer.
  2. *Query privacy* (optional): the spatial location of a user's query cannot be revealed to data silos.

Remark. In practice, the need for query privacy may vary across applications. For example, in scenarios such as a passenger requesting a taxi-calling service through AMAP [3], query privacy may be unnecessary. This is because the platform ultimately needs to know the passenger's pickup location. Conversely, in situations like performing contact tracings based on a patient's location, query privacy becomes crucial to prevent the disclosure of sensitive spatial information to data silos. For ease of presentation, we classify federated spatial queries into two kinds: *asymmetric queries* (which require data privacy only) and *symmetric queries* (which require both data privacy and query privacy).

To satisfy the privacy requirements, some existing systems [14, 50] assume the existence of a trusted broker responsible for collecting partial answers, which may contain sensitive data, from each silo. In reality, even if brokers (*e.g.*, Acxiom [2]) charge high fees for their data broker services, there is still a risk of them leaking sensitive data for personal gain [1]. Thus, we explore solutions without reliance on a trusted broker.

2.2 Main challenges

Federated queries can be realized by secure multi-party computation (SMC) [24], as in prior studies for relational data [14, 50]. Nevertheless, our empirical study shows that they are highly inefficient on spatial queries.

2.2.1 Inefficiency on federated spatial queries

As an illustrative study, we perform an asymmetric federated kNN query by extending SMCQL [14] and Conclave [50], two representative solutions to secure query processing on (relational) data federations.

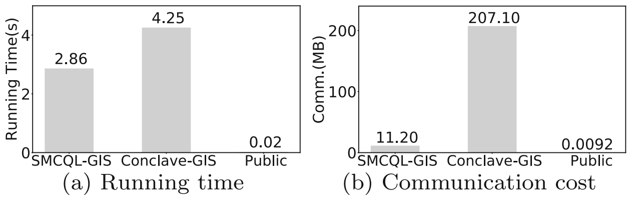
Overview of Existing Solutions. The common framework [14, 50] for secure query processing over a data federation decomposes query execution into *plaintext* queries within each silo and *secure* computations of the partial results across silos. Existing solutions differ in the SMC techniques used for secure operations, with garbled circuits (GC) and secret sharing (SS) as the mainstreams [24]. For example, SMCQL-GIS [14] uses a prevalent GC based library, ObliVM [38], to support two silos. Conclave-GIS [50] adopts an SS based technique (Sharemind [18]), which enables query processing on three silos.

Setup. SMCQL-GIS [14] and Conclave-GIS [50] are extended to asymmetric 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 kNNs are derived by a top-k operation from these returned points, which are securely sorted by their distances to the query point. We experiment with two silos with $k = 16$. Other details of experimental setups are elaborated in Sec. 7.1.

Result. Figure 1 plots the (average) running time and communication cost to process an asymmetric federated kNN query leveraging existing solutions [14, 50]. The results are averaged over 50 queries. Compared with Public, *i.e.*, plaintext kNN query execution without any privacy protection, the secure counterpart incurs 142× to 212× longer running time and $1,216\times$ to $22,510\times$ higher communication cost. Although the method SMCQL-GIS yields shorter running time and lower communication overhead than Conclave-GIS, it is *limited to scenarios with only two silos* due to its reliance on garbled circuits (GC). Yet it still takes 2.86 seconds to answer a federated spatial query, which can hurt user experiences in applications where query time efficiency is critical.

Table 1 Percentage of time spent for *plaintext* or *secure* operations in an asymmetric federated kNN query

Existing solution	Plaintext(%)	Secure(%)
SMCQL-GIS [14]	0.14	99.86
Conclave-GIS [50]	0.10	99.90



(a) Running time (b) Communication cost

Fig. 1 Inefficiency of Conclave-GIS and SMCQL-GIS on asymmetric federated kNN query, where SMCQL-GIS and Conclave-GIS are our extensions on SMCQL [14] and Conclave [50] to spatial queries (see Sec. 7.1)

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 the test query, over 99% time is spent on secure operations (*e.g.*, secure distance comparisons) as shown in Table 1. Specifically, SMCQL-GIS and Conclave-GIS adopt sorting to find kNNs among nk candidates by using $O(nk \log(nk))$ secure distance comparisons. A single secure distance comparison in SMCQL-GIS takes 209 ms, while in Conclave-GIS it takes 248 ms, which equals the time required for at least 10^6 plaintext comparisons.
- *Reliance on General-Purpose Libraries.* Existing methods use general-purpose libraries to implement secure operations (*e.g.*, ObliVM [38] in SMCQL [14]). General-purpose libraries sometimes sacrifice efficiency for generalization or compatibility. For example, the secure summation used in Hu-Fu can be 16× faster than that in ObliVM (see Sec. 7). As will be shown in Sec. 4, federated spatial queries can be processed with only a few secure operations. This facilitates acceleration by dedicated protocols specifically tailored for these secure operations.

Takeaway. Our study shows that existing secure query processing solutions (*e.g.*, [14, 50]) for data federations are

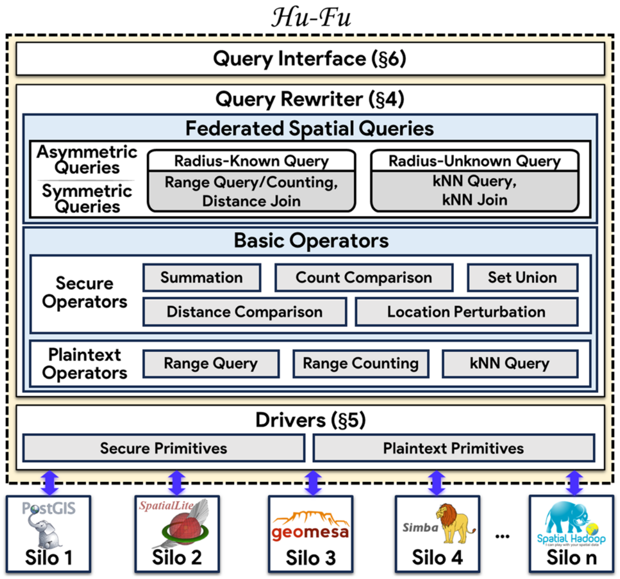


Fig. 2 Foundation architecture of Hu-Fu

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 methods (see Sec. 5). As next, we give an overview of Hu-Fu and elaborate on its functional modules in the following.

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 via two modules: *(1)* 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; *(2) 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 will briefly explain its architecture and workflow as follows.

3.1 Architecture

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

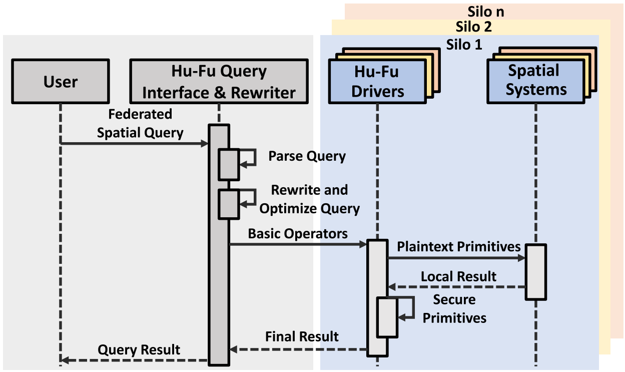


Fig. 3 Illustration of Hu-Fu workflow

Query Rewriter (Sec. 4). It decomposes mainstream spatial queries (federated range query, range counting, kNN query, distance join, and kNN join) into plaintext operators (executed within silos) and secure operators (executed across silos). We define three *plaintext* operators (plaintext range query, range counting, and kNN query) and five *secure* operators (secure summation, count comparison, set union, distance comparison, and location perturbation) as the basic operators. We also design novel execution plans that decompose these federated spatial queries into 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 [18, 23, 32, 34]. Plaintext operators are implemented as *plaintext primitives* on top of the underlying spatial databases, which support various systems, *e.g.*, PostGIS [10], SpatiaLite [11], MySQL [8], GeoMesa [26], Simba [57] and SpatialHadoop [21].

Query Interface (Sec. 6) This module *(1)* provides a transparent and unified federation view to users, and *(2)* supports both asymmetric and symmetric federated spatial queries written in SQL. We implement the query interface by extending the schema manager and parser of Calcite [19]. We also provide interfaces such as JDBC for easy integration of Hu-Fu to users' programs.

3.2 Workflow

Figure 3 depicts the workflow of Hu-Fu for querying a data federation of n silos. The *query interface* and *query rewriter* are deployed on the user's machine to provide a portal for spatial services, while each silo runs a *driver* that interacts with its underlying spatial database.

In the workflow, a user's SQL-based federated spatial query 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 executed as primitives by the *drivers*: plaintext primitives run locally on spatial databases to produce intermediate results, while secure primitives assemble these partial results to obtain the final answer, which is returned to the user via the query interface.

4 Query rewriter

This section presents the design of the query rewriter in Hu-Fu, which decomposes a federated spatial query into basic operators to form the query execution plan. Specifically, we first introduce the involved basic operators in Sec. 4.1. Next, we explain the overall decomposition strategies in Sec. 4.2. Then, we introduce the rewriter of *asymmetric* federated spatial queries in Sec. 4.3 from two categories: *radius-known* and *radius-unknown* queries. Since asymmetric federated spatial queries do not assume query privacy, we also propose efficient solutions to *symmetric* federated spatial queries in Sec. 4.4. Finally, we discuss practical issues in Sec. 4.5.

4.1 Basic operators

Our acceleration strategy is to *decompose queries into basic operators so that the majority of distance-related operations occur within silos in plaintext*, thereby reducing the need for secure operations across silos. The selection principle of basic operators is explained below.

4.1.1 Operator selection principles

There are two categories of basic operators in Hu-Fu: *plaintext* and *secure* operators. The *plaintext* operators handle local queries within each individual silo, while the *secure* operators perform atomic computations over sensitive data in a privacy-preserving manner.

- *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 unless strictly necessary, and efficiently implemented operators are preferable.

Adhering to these principles, we select three plaintext operators and five secure operators, which will be elaborated in Sec. 4.1.2 and Sec. 4.1.3, respectively.

4.1.2 Plaintext operators

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

Definition 1 (Plaintext Range Query/Counting) Given a silo F_i and a query range $\mathcal{R}$, the *plaintext range query* $\mathsf{RQ}(F_i, \mathcal{R})$ retrieves the spatial objects in F_i that fall within $\mathcal{R}$, and the *plaintext range counting* $\mathsf{RC}(F_i, \mathcal{R})$ returns the number of such objects.

Definition 2 (Plaintext kNN Query) Given a silo F_i, a query object q, and a positive integer k, the *plaintext kNN query* $\mathsf{kNN}(F_i, q, k)$ retrieves the k nearest spatial objects in F_i to the query object q.

The plaintext operators comply with the principles described in Sec. 4.1.1, because they are supported by almost all spatial databases. They are implemented as *plaintext primitives* in Hu-Fu drivers, which we defer to Sec. 5.1. The query range can be various shapes, such as circles and rectangles. For ease of presentation, we mainly focus on circular ranges in this section and discuss extensions to other shapes in Sec. 4.5.

4.1.3 Secure operators

We define five secure operators: *summation, count comparison, set union, distance comparison, and location perturbation*. The first three secure operators are designed to preserve data privacy, while the latter two secure operators aim to protect query privacy.

Definition 3 (Secure Summation) Given n data silos $\{F_i\}$ each holding a private value v_i, this operator SUM sums up these values, *i.e.*, $\mathsf{SUM}(v_1, \cdots, v_n) = \sum_{i=1}^{n} v_i$, while protecting the privacy of v_i in silo F_i from all other silos F_j $(\forall j \neq i)$.

Definition 4 (Secure Count Comparison) Given n data silos $\{F_i\}$ each holding a private count v_i and a public constant k, this operator CMP compares the sum of these counts with k, *i.e.*, $\mathsf{CMP}(v_1, \cdots, v_n, k) = sign(\sum_{i=1}^{n} v_i - k)$, without leaking the sum $\sum_{i=1}^{n} v_i$ or the count v_i in silo F_i to any other silos F_j $(\forall j \neq i)$.

Definition 5 (Secure Set Union) Given n data silos $\{F_i\}$ each holding a set of spatial objects $S_i = \{o_1^i, \cdots, o_{m_i}^i\}$, this operator SUN computes the union of spatial objects from all silos, *i.e.*, $\mathsf{SUN}(S_1, \cdots, S_n) = \cup_{i=1}^{n} S_i$, without leaking the spatial objects S_i in silo F_i to any other silos F_j $(\forall j \neq i)$.

Definition 6 (Secure Distance Comparison) Given a query user holding a private location l_q, a data silo holding a private location l_o, and a distance threshold r, this operator DCMP compares the distance between l_q and l_o with the threshold r, *i.e.*, $\mathsf{DCMP}(l_q, l_o, r) = sign(\mathsf{dist}(l_q, l_o) - r)$, without leaking the location of either party (*i.e.*, the user or silo) to the other.

Definition 7 (Secure Location Perturbation) Given a private location $x \in \mathbb{R}^2$, this operator GeoI obfuscates it into a location $z = \mathsf{GeoI}(x)$ while satisfying (ϵ, δ)-Geo-Indistinguishability (Geo-I) [54]. The privacy requirement of (ϵ, δ)-Geo-I is satisfied iff the following inequality holds for any two locations x, x' in the location set and any location subset Z:

$$\Pr[\mathsf{GeoI}(x) \in Z] \leq e^{\epsilon \mathsf{dist}(x,x')} \cdot \Pr[\mathsf{GeoI}(x') \in Z] + \delta \qquad (1)$$

where $\Pr[\mathsf{GeoI}(x) \in Z]$ is the probability that the perturbed location belongs to the subset Z, and ϵ, δ represent the privacy preservation level of Geo-I.

ϵ-Geo-I [13] adapts the de facto standard privacy notion, *differential privacy* [36], to protect location data, where ϵ is known as the privacy budget. (ϵ, δ)-Geo-I relaxes the definition of ϵ-Geo-I by allowing a small failure probability δ. This way of relaxation has gained widespread usages in (standard) differential privacy [36].

Remark. The secure operators comply with the principles in Sec. 4.1.1 since *(1)* most of them do not involve distance operations (with the exception of secure distance comparison) and *(2)* all of them have dedicated and efficient implementations (see Sec. 5.2 for details).

4.2 Overview of our decomposition strategies

In the following, we formally define the federated spatial queries and introduce our taxonomy to categorize them (Sec. 4.2.1). We then elaborate on the main ideas of our decomposition strategies for each category (Sec. 4.2.2).

4.2.1 Federated spatial queries and taxonomy

Before diving into our decomposition strategies, we first define the five federated spatial queries. The privacy requirement below includes either data privacy alone or both data and query privacy defined in Sec. 2.1.

Definition 8 (Federated Range Query/Counting) Given a federation F of n data silos $\{F_i\}$, and a query range $\mathcal{R}$, a *federated range query* retrieves all spatial objects located within $\mathcal{R}$, while a *federated range counting* returns the number of such objects. Both queries need to satisfy the privacy requirement.

Definition 9 (Federated Distance Join) Given a federation F of n data silos $\{F_i\}$, an input dataset Q of spatial objects, and a distance radius r, a federated distance join retrieves all pairs of objects (q, o) where $q \in Q, o \in F$ such that the distance $\mathsf{dist}(l_q, l_o) \leq r$, while satisfying the privacy requirement, *i.e.*,

$$Q \bowtie_r F = \{(q, o) \mid q \in Q, o \in F, \mathsf{dist}(l_q, l_o) \leq r\}.$$

Definition 10 (Federated kNN Query/Join) Given a federation F of n data silos $\{F_i\}$, a query object q, and a positive integer k, a *federated kNN query* retrieves the k nearest objects in F to the query object q, *i.e.*,

$$\forall o \in \mathrm{kNN}(q), \forall o' \in F \setminus \mathrm{kNN}(q), \mathsf{dist}(q, o) \leq \mathsf{dist}(q, o').$$

When the query objects form an input dataset Q, a *federated kNN join* retrieves all pairs of objects (q, o) where $q \in Q$ and o belongs to the kNN of q in F, *i.e.*,

$$Q \bowtie_{\mathrm{kNN}} F = \{(q, o) \mid q \in Q, o \in \mathrm{kNN}(q)\}.$$

Both queries need to satisfy the privacy requirement.

Taxonomy. The above queries can be categorized from *two orthogonal dimensions*: the scope of the privacy requirement and whether the searching radius (of the query range) is explicitly given. Specifically, based on whether query privacy is included in the privacy requirement, the queries are classified into *asymmetric* and *symmetric* queries (see the differences in Sec. 2.1). Based on whether the searching radius is explicitly given, the queries are classified into *radius-known* and *radius-unknown* queries. Intuitively, the federated range query, range counting, and distance join belong to radius-known queries, while the federated kNN query and kNN join belong to radius-unknown queries.

4.2.2 Main idea of our decomposition strategies

Basic Principle. In Hu-Fu, the core principle of the 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 initially processed using plaintext operators within each silo, and their results are then securely assembled to form the final outcome. At the minimum, one secure operator is compulsory, and additional secure operators may be required if there are interactions across silos.

Based on the aforementioned basic operators and taxonomy of queries, we now introduce the *main ideas* of decomposing different categories of queries.

Main Idea for Asymmetric Queries with Data Privacy Solely. Our idea is elaborated as follows:

- *Radius-Known Queries.* A radius-known query (*e.g.*, federated range query and range counting) can be decomposed into the corresponding plaintext operators within each silo and only one secure operator (*e.g.*, a secure set union or a secure summation) for assembling the partial results across silos.
- *Radius-Unknown Queries.* Each radius-unknown query (*e.g.*, federated kNN query) is viewed as an iterative process of trialing different search radii until exactly k spatial objects are found within the circular search area. This execution can be converted into multiple radius-known queries (*e.g.*, federated range counting), with the number of radius-known queries minimized by a binary search. Each iteration utilizes a secure operator to ensure data privacy.

Key Insight for Symmetric Queries with Both Data and Query Privacy. When additionally considering query privacy, our key insights are as follows:

- *Radius-Known Queries.* A native solution employs a secure distance comparison operator for every spatial object, but leads to excessive secure distance operations. Instead, we first obfuscate the sensitive query location using a secure location perturbation operator to create a noised location that can be safely published to each silo. Then, leveraging the previous idea for radius-known queries, each silo identifies a small set of candidates. For each candidate, a secure distance comparison operator verifies if its distance to the query location is within the specified radius, while protecting query privacy.
- *Radius-Unknown Queries.* Similar to the aforementioned ideas for decomposing radius-unknown queries, we can still decompose them into a series of radius-known queries.

4.3 Decomposing asymmetric queries with data privacy only

This subsection proposes our methods for decomposing radius-known queries (Sec. 4.3.1) and radius-unknown queries (Sec. 4.3.2), which only consider data privacy. The decomposition plans are summarized in Table 2.

4.3.1 Decomposing radius-known queries

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

Decomposition Plan. (1) *Federated range query* can be decomposed into n *plaintext range queries*, each with a radius r, where each plaintext operator retrieves the partial result within each one of the n silos. Afterwards, a *secure set union* operator assembles these partial result while maintaining data privacy. (2) Similarly, *federated range counting* can be decomposed into n *plaintext range counting* operators to obtain n partial counts. These partial counts will later be aggregated by a *secure summation* operator. (3) *Federated distance join* is equivalent to requesting federated range queries $|R|$ times, each of which follows the previous plan.

Complexity Analysis. Let T_{RQ} and T_{RC} denote the time complexity of plaintext range query and range counting, respectively. $|S|$ denotes the size of returned set. Based on the complexities of secure operators (see Sec. 5.2), the time complexity and communication cost of the radius-known queries are as follows. (1) *Federated range query* takes $O(T_{\mathsf{RQ}} + n + |S|)$ time and $O(n + |S|)$ communication cost. (2) *Federated range counting* takes $O(T_{\mathsf{RC}} + n^3)$ time and $O(n^2)$ communication cost. (3) *Federated distance join* takes $O(|R| \cdot T_{\mathsf{RQ}} + n + |S|)$ time and $O(n + |S|)$ communication cost.

4.3.2 Decomposing radius-unknown queries

Federated kNN query and kNN join are classified as radius-unknown queries due to the absence of an explicitly given range. 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. 4.3.1, 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 r via a binary search and then retrieve the spatial objects within this search range. For each radius r, we securely check whether the counting result is smaller than k. As long as r falls between the kth and the $(k+1)$th nearest distance to the query object q, the spatial objects within this range are precisely the k nearest neighbors.
- *Algorithm Details.* Alg. 1 illustrates the decomposition of a federated kNN query. Lines 1-8 derive the radius r. We initialize a lower bound ($l = 0$) and upper bound ($u = U$) of the radius, where U can be set as the spatial area's diameter or a user-defined value. A binary search is then performed to find the appropriate radius until reaching the distance precision ϵ_0 (lines 2-9). In each iteration, r is set to $(l + u)/2$. For each r, a *plaintext range counting* operator is executed within each silo, and a *secure count comparison* operator is invoked to compare the total count with the integer k (lines 4-5). If the total count is less than k (*i.e.*, $sign < 0$), indicating an undersized radius, l will be increased to r. Conversely, if the total

Table 2 The number of basic operators in the decomposition plans of *asymmetric* federated spatial queries

Category	Federated spatial query	#(Plaintext operator)		#(Secure operator)	
		Range query	Range counting	Count comparison	Set union/Summation
Radius-known	Range query	n	0	0	1/0
	Range counting	0	n	0	0/1
	Distance join	$n\|R\|$	0	0	$\|R\|$/0
Radius-unknown	kNN query	n	$O(n \log U)$	$O(\log U)$	1/0
	kNN join	$n\|R\|$	$O(n\|R\| \log U)$	$O(\|R\| \log U)$	$\|R\|$/0

Radius-known queries only involve one type of secure operators (secure summation or set union). Radius-unknown queries are executed in multiple rounds which additionally require secure count comparisons to ensure security. n is the number of silos, R is the input dataset in spatial joins, and U is the upper bound for the binary-search radius

count exceeds k (*i.e.*, $sign > 0$), u will be decreased to r. The binary search ensures that the final radius r is sufficiently close to the kth nearest distance. Finally, a *plaintext range query* is executed on each silo, and the partial results are collected using a *secure set union* (lines 9-10).

In Alg. 1, the distance precision ϵ_0 is initially set based on the application requirement. For example, many spatial applications (*e.g.*, taxi-calling) have a minimum distance precision requirement that is typically measured in meters. Then, ϵ_0 can be set to 1 meter.

Complexity Analysis. Alg. 1 requires $O(\log \frac{U}{\epsilon_0}) = O(\log U)$ iterations to obtain the final radius, where ϵ_0 is a constant to denote this radius's precision. In each iteration, the plaintext range counting takes $O(T_{\mathsf{RC}})$ time, and the secure count comparison takes $O(n)$ time and $O(n^2)$ communication cost. In lines 9-10, Alg. 1 performs a plaintext range query that takes $O(T_{\mathsf{RQ}})$ time and a secure set union that takes $O(n+k)$ time and $O(n+k)$ communication cost. Overall, the total time complexity is $O((T_{\mathsf{RC}} + n) \cdot \log U + T_{\mathsf{RQ}} + k)$, and the communication cost is $O(n^2 \cdot \log U + k)$. Intuitively, the complexity of federated kNN join is equal to that of federated kNN query, multiplied by a factor $|R|$.

Example 3 Figure 4 illustrates the procedure of Alg. 1 with a query point (4, 4) and $k = 3$ over 3 silos, where the objects 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 count comparison with k is performed across silos. And we get 10 objects, which is greater than k. Hence in the 2nd round, the radius decreases to 2 and is sent to each silo for plaintext range counting and secure count comparison. There are 2 objects, which is fewer than k. Thus, in the 3rd round, the radius increases to 3, and the procedure continues, where the secure count comparison results implies that $sign = 0$ and the search terminates. Finally, the basic operators, including the plaintext range query with the center (4, 4) and radius 3 and secure set union, are performed to retrieve the 3 query answers.

```
Algorithm 1: Asymmetric federated kNN query
Input: federation $F$, query object $q$, integer $k$
Output: the (exact) query answer $ans$
1  $[l, u] \leftarrow [0, U]$, where $U$ is a predefined upper bound;
2  while $u - l \geq \epsilon_0$ do
3      $r \leftarrow (l + u)/2$, $\mathcal{R} \leftarrow$ circle$(q, r)$;
4      foreach silo $F_i \in F$ do // perform in parallel
5          $v_i \leftarrow$ plaintext range counting $\mathsf{RC}(F_i, \mathcal{R})$;
6      $sign \leftarrow$ secure count comparison $\mathsf{CMP}(\{v_i\}, k)$;
7      if $sign < 0$ then $l \leftarrow r$;
8      else if $sign > 0$ then $u \leftarrow r$;
9      else break;
10 foreach silo $F_i \in F$ do // perform in parallel
11     $S_i \leftarrow$ plaintext range query $\mathsf{RQ}(F_i$, circle$(q, r))$;
12 return $ans \leftarrow$ secure set union $\mathsf{SUN}(S_1, \cdots, S_n)$;
```

Optimization via Differential Privacy. We exploit differential privacy [36] to further accelerate federated kNN query and federated kNN join from two aspects.

- *Tighten Predefined Upper Bound.* We ask each F_i to perform a plaintext kNN query operator and return the kth object's distance U_i to the query point . Since directly returning such values may violate the data privacy requirement, we apply the truncated Laplacian mechanism [15] on it. That is, let each silo add a positive noise and obtain the perturbed value $\tilde{U_i}$. We can tighten the upper bound as the shortest distance in all silos, *i.e.*, $U = \min\{\tilde{U_i}\}$, since there must be at least k objects in this range.
- *Reduce Running Time and Communication Cost in Secure Count Comparison.* The secure count comparison in Alg. 1 compares $\sum_1^n v_i$ with k, resulting in $O(n^2)$ running time and communication cost. However, when $\sum_1^n v_i$ differs significantly from k, this can be reduced to $O(n)$ by using the Laplacian mechanism [36] in differential privacy. This mechanism injects a noise into the local

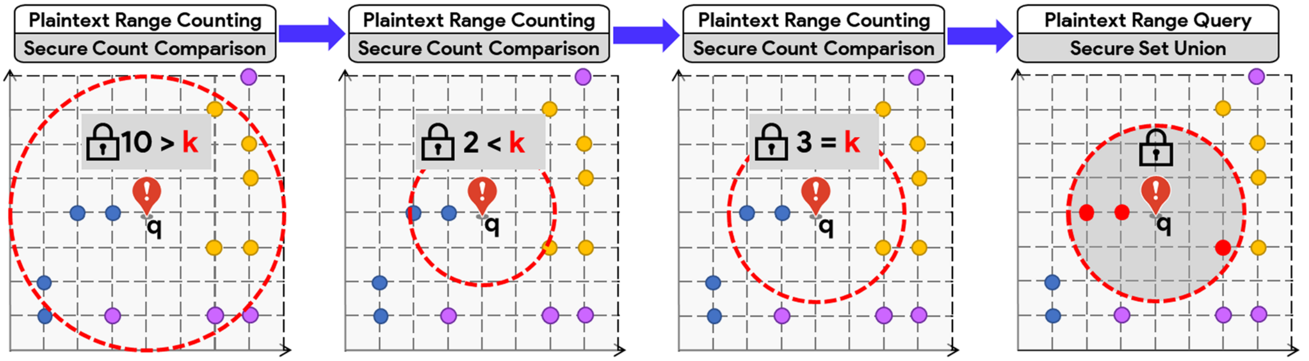


Fig. 4 Running example for (asymmetric) federated kNN query ($k = 3$)

Algorithm 2: Symmetric federated range query

```
    Input: federation $F$, query object $q$, radius $r$
    Output: the (exact) query answer $ans$
1   $q' \leftarrow$ secure location perturbation $\mathsf{GeoI}(q)$;
2   $r' \leftarrow r + \mathsf{dist}(l_q, l_{q'})$, range $\mathcal{R}' \leftarrow$ circle$(q', r')$;
3   foreach silo $F_i \in F$ do // perform in parallel
4       $Cand_i \leftarrow$ plaintext range query $\mathsf{RQ}(F_i, \mathcal{R}')$
5   for silo $F_i \in F$ do // perform in parallel
6       foreach candidate spatial object $o \in Cand_i$ do
7           $sign \leftarrow$ secure distance comparison $\mathsf{DCMP}(l_q, l_o, r)$;
8           if $sign \leq 0$ then $ans \leftarrow ans \cup \{o\}$
```

count in each silo, and then perturbed counts are aggregated in plaintext. If the perturbed total count is much smaller or larger than k, we directly adjust the threshold without running the secure operator.

4.4 Decomposing symmetric queries with both data privacy and query privacy

This subsection presents our methods for decomposing radius-known queries (Sec. 4.4.1) and radius-unknown queries (Sec. 4.4.2) with both data and query privacy. The decomposition plans are summarized in Table 3.

4.4.1 Decomposing radius-known queries

Since query location must be protected in symmetric queries, radius-known queries can no longer be decomposed into plaintext range query/counting within each silo directly. Instead, we use the Geo-I mechanism [13, 54] to preserve the query privacy, as detailed below.

Decomposition Plan for Federated Range Query. Alg. 2 presents the decomposition plan for federated range queries. Initially, a *secure location perturbation* operator is applied to generate an obfuscated object q'. Next, the search radius is increased by the distance from location l_q to $l_{q'}$ (line 2). This ensures that the expanded query range, denoted by a circle centered at q' with a radius $r' = r + \mathsf{dist}(l_q, l_{q'})$, completely covers the intended query area. Within each silo, a *plaintext range query* is then performed using the expanded query range (line 3). As a result, each silo obtains a set of candidates for the query answer. To refine these candidates, *secure distance comparison* operators are employed to filter out those outside the true query range and collect the final answer while satisfying both data privacy and query privacy (lines 4-7).

Example 4 Figure 5 presents an illustrative example for Alg. 2. Suppose the query object q is located at $(2.5, 2.5)$ and the radius r of the circular query range is 1.6. By using the *secure location perturbation* operator, q is obfuscated into q' located at $(4, 4)$, so the radius r' is increased to $1.6 + \sqrt{(2.5-4)^2 + (2.5-4)^2} = 3.7$ (lines 1-2 of Alg. 2). After executing the *plaintext range query* operator with the expanded query range, we identify 3, 2, and 5 candidates (marked in different colors) in all three silos (line 3). Finally, each candidate is further refined by the *secure distance comparison* operator.

Extension to Other Radius-Known Queries. The decomposition plan for *federated range counting* is almost identical to Alg. 2. The key difference lies in line 7, where federated range counting only needs to aggregate the counts. When dealing with *federated distance join*, it is initially converted into a series of (symmetric) *federated range queries*. Subsequently, each federated range query is decomposed by Alg. 2.

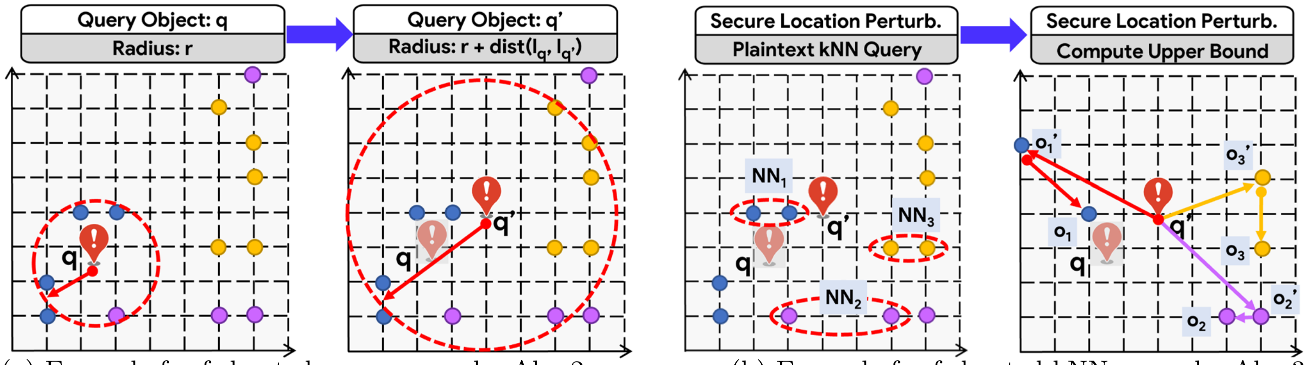
4.4.2 Decomposing radius-unknown queries

Similar to the binary search procedure in Alg. 1, a symmetric *federated kNN query* can be broken down into multiple rounds of (symmetric) radius-known queries. Besides, *federated kNN join* can still be decomposed into a series of independent federated kNN queries. Thus, we focus primarily on the necessary modifications for federated kNN queries in the following.

Table 3 The number of basic operators in the decomposition plans of *symmetric* federated spatial queries

Category	Federated spatial query	#(Plaintext operator)		#(Secure operator)		
		Range query	kNN query	Location perturbation	Distance comparison	Set Union
Radius-known	Range query	n	0	1	$\lvert Cand\rvert$	1
	Range counting	n	0	1	$\lvert Cand\rvert$	0
	Distance join	$n\lvert R\rvert$	0	$\lvert R\rvert$	$\lvert R\rvert\cdot\lvert Cand\rvert$	$\lvert R\rvert$
Radius-unknown	kNN query	$O(n\log U)$	n	$1+n$	$O(\lvert Cand\rvert\log U)$	1
	kNN join	$O(n\lvert R\rvert\log U)$	$n\lvert R\rvert$	$(1+n)\lvert R\rvert$	$O(\lvert Cand\rvert\log U\cdot\lvert R\rvert)$	$\lvert R\rvert$

n is the number of silos, $|R|$ is the size of the input dataset R in spatial joins, U is the upper bound for the binary-search radius, and $|Cand|$ is the total number of candidate objects from all data silos



(a) Example for federated range query by Alg. 2

(b) Example for federated kNN query by Alg. 3

Fig. 5 Examples for decomposing (symmetric) federated range query and kNN query

Naive Decomposition Plan. A naive extension of Alg. 1 can be time-consuming due to the trivial upper bound of the search radius. If the initial upper bound is set too high, the decomposed radius-known queries will require a large number of secure distance operations, which becomes the major efficiency bottleneck.

Optimized Decomposition Plan. To overcome the limitation of the naive method, we devise Alg. 3 to compute a tighter upper bound based on the perturbed location. Specifically, line 1 perturbs the query object q into q' privately. In line 2, plaintext kNN query is performed in each silo to identify the k nearest neighbors to q'. However, we cannot send the kth nearest distances to the user's client as the upper bound, since it leaks information about locations in the data silos. Instead, each silo obfuscates the kth nearest neighbor $o_i \in NN_i$ into a noised spatial object o_i' (lines 3-4). In line 5, each silo locally computes its own upper bound U_i and sends it to the user's client. Finally, the upper bound can be safely set as the minimum value among $\{U_i\}$ (line 6).

Example 5 Figure 5 illustrates Alg. 3. In line 1, the query object q located at $(2.5, 2.5)$ is perturbed into the object q' located at $(4, 4)$. Each silo then performs a *plaintext kNN query* with $k = 2$ on q'. The top-2 nearest neighbors in three silos are denoted by NN_1-NN_3, where o_i is the farthest object to q' within NN_i. For example, the object o_1 at $(2, 4)$ has a distance of 2 to q'. However, revealing this distance directly may leak spatial information about o_1. Instead, Alg. 3 leverages a *secure location perturbation* operator to obfuscate o_1 to o_1' located at $(0, 6)$. Similarly, o_2 and o_3 are perturbed to o_2' and o_3', respectively. Based on the Euclidean distances, we have $U_1 = 7.30$, $U_2 = 5.16$, and $U_3 = 5.24$ (line 5). Finally, we pick the minimum from $\{U_i\}$ and derive the tight upper bound $U = 5.16 + 2.12 = 7.28$.

Algorithm 3: Compute tight upper bound for optimizing symmetric federated kNN query

```
Input: federation $F$, query object $q$, integer $k$
Output: the upper bound $U$ for binary-search radius
1 $q' \leftarrow$ secure location perturbation $\mathsf{GeoI}(q)$;
2 foreach silo $F_i \in F$ do // perform in parallel
3     $NN_i \leftarrow$ plaintext kNN query $\mathsf{kNN}(F_i, q', k)$
4 $o_i \leftarrow \arg\max_{o \in NN_i}\{\mathsf{dist}(l_o, l_{q'})\}$;
5 $o_i' \leftarrow$ secure location perturbation $\mathsf{GeoI}(o_i)$;
6 $U_i \leftarrow \mathsf{dist}(l_{q'}, l_{o_i'}) + \mathsf{dist}(l_{o_i'}, l_{o_i})$;
7 return $U \leftarrow dist(l_q, l_{q'}) + \min\{U_i \mid i = 1, \cdots, n\}$;
```

The correctness of Alg. 3 is proved in Lemma 1.

Lemma 1 *The upper bound U in Alg. 3 is no shorter than the kth nearest distance to the query object q in the data federation F.*

Proof Let d^* denote the kth nearest distance to q, so d^* should satisfy the following inequality:

$$d^* \leq \max\{\mathsf{dist}(l_q, l_o) \mid o \in NN_i\}, \forall i \in [1, n] \tag{2}$$

According to the triangle inequality, we have

$$\mathsf{dist}(l_q, l_o) \leq \mathsf{dist}(l_q, l_{q'}) + \mathsf{dist}(l_{q'}, l_o) \tag{3}$$

Based on (2) and (3), we can derive that

$$d^* \leq \mathsf{dist}(l_q, l_{q'}) + \max\{\mathsf{dist}(l_{q'}, l_o) \mid o \in NN_i\}, \forall i \in [1, n]$$

Since $\max\{\mathsf{dist}(l_{q'}, l_o) \mid o \in NN_i\} = \mathsf{dist}(l_{q'}, l_{o_i})$ based on the line 3 of Alg. 3, we have

$$d^* \leq \mathsf{dist}(l_q, l_{q'}) + \mathsf{dist}(l_{q'}, l_{o_i}), \forall i \in [1, n]$$

Based on the triangle inequality for $\mathsf{dist}(l_{q'}, l_{o_i})$ and the definition of U_i in line 5 of Alg. 3, we have

$$\begin{aligned} d^* &\leq \mathsf{dist}(l_q, l_{q'}) + \big(\mathsf{dist}(l_{q'}, l_{o'_i}) + \mathsf{dist}(l_{o'_i}, l_{o_i})\big) \\ &\leq \mathsf{dist}(l_q, l_{q'}) + U_i \end{aligned} \tag{4}$$

According to the inequality in (4) and the definition of U in line 6, we can now prove $d^* \leq U$.

Remark. In Alg. 3, the k nearest neighbors NN_i of the perturbed location q' are used to derive the upper bound. Notice that $\{NN_i\}$ do not necessarily encompass all the query results of the original location q. By contrast, with Lemma 1, the federated range query/counting during the binary-search can ensure that the candidate set includes the exact kNN of q.

4.5 Discussion

We provide further discussions on the query rewriter. Due to the page limitation, please refer to our full paper [7] for the security proof, algorithm details, or evaluation results related to the following discussions.

Security of Query Rewriter. We prove the security of our query rewriter based on the composition lemma in [29]. 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.

Handling Ties in kNN Queries. In federated kNN queries, we may encounter ties where multiple spatial objects share the same distance (*i.e.*, the kth nearest distance r^*) to the query object q. Here, we must resolve *two technical issues*: (1) identifying the presence of ties, and (2) retrieving exactly k nearest neighbors.

(1) Line 8 ($sign = 0$) of Alg. 1 indicates the current search radius covers exactly k objects (*i.e.*, no ties). If $sign \neq 0$ during the binary-search, then there are ties.

(2) Once ties are identified, we proceed to retrieve exactly k spatial objects. Let $[l, u]$ denote the lower and upper bounds of r^*. First, we use a federated range query with the circular range circle(q, l) to cache the nearest neighbors that are not part of the ties. We denote the number of these objects as k_l. Next, we select $k - k_l$ objects from the tied ones by sequentially requesting objects from all data silos until we reach the desired count. Finally, we use a secure union operator to collect the cached partial answers from all data silos.

Extension to Rectangular Query Range. The decomposition plan for radius-known queries can be extended to accommodate rectangular-shaped of query ranges. For *asymmetric queries*, the extension can be seamlessly implemented by using plaintext range query/counting operators for rectangular query ranges, which are typically supported by spatial database systems. For *symmetric queries*, where the query object (*i.e.*, the rectangle center) is private, our extension proceeds as follows. We first compute the rectangle's circumscribed circle. Next, by querying the circumscribed circle with lines 1-4 of Alg. 2, we identify potential candidates. Finally, we securely verify if a candidate (x, y) lies within the rectangle $[(x_L, x_R), (y_L, y_R)]$ using the Yao's garbled circuit (GC) protocol [24] to check the inequalities $x \geq x_L$, $x \leq x_R$, $y \geq y_L$, and $y \leq y_R$. The Yao's GC protocol here can be implemented using ObliVM [38].

Beyond Mainstream Spatial Queries. The query rewriter also supports *aggregation queries*, *e.g.*, the aggregate attribute on the result of kNN query or range query. For example, the range aggregate query can be decomposed similarly to a federated range counting. Our solution can be also extended to support *approximate spatial queries* by replacing the exact methods for the plaintext operators with approximate ones. While this is easy to implement, it may be hard to achieve a good balance between efficiency and accuracy.

5 Drivers

In Hu-Fu, a driver is deployed on each data silo, consisting of both *plaintext primitives* (Sec. 5.1) and *secure primitives* (Sec. 5.2). Here, plaintext primitives refer to the implementations of plaintext operators that leverage the local spatial database at each silo. Secure primitives, on the other hand, indicate our secure protocols tailored for the secure operators defined in Sec. 4.1.3.

Unlike existing systems [14, 50], Hu-Fu aims to support *heterogeneous* databases through drivers. In this way, Hu-Fu can *enhance usability* and *avoid costly data migration* compared to these solutions that assume local databases are *homogeneous*. To achieve this, the *main difficulties* include

(1) drivers must integrate with the query rewriter to convert plaintext operators into diverse query formats used by data silos, and (2) drivers must offer default implementations of plaintext operators for local databases that lack support.

5.1 Plaintext primitives

Plaintext primitives implement *plaintext range query, range counting, and kNN query*. 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 plaintext primitives are implemented by the underlying spatial databases.

- For databases that support these plaintext queries, *e.g.*, Simba [57] and PostGIS [10], we utilize the built-in functions for these queries or generate the corresponding SQL request. For example, in PostGIS [10], a plaintext range counting on silo F_i with the center p and radius r of a circular range can be implemented by requesting the SQL below.

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

- When databases lack native support for any query, the drivers offer a default implementation based on their supported queries and indexes. For example, GeoMesa [26] does not inherently support range counting, so we extend range counting by calling a range query and subsequently counting the result size.

Time Complexity. In modern spatial databases, plaintext range query, range counting, and kNN query can take $O(\log m + |S|)$, $O(\log m)$, and $O(\log m)$ time [40], where m is the data size and $|S|$ is the output size.

Remark. In practice, the actual performance of plaintext primitives depends on the native implementation of the local spatial database at each silo. Thus, when silos utilize heterogeneous spatial databases, the efficiency of federated spatial queries can be limited by the slowest plaintext primitive (see Sec. 7.5).

5.2 Secure primitives

The secure primitives, including secure summation, count comparison, set union, distance comparison, and location perturbation, are independent of local databases.

Primitive Implementation. Each secure primitive is optimized with a tailored secure protocol as follows.

Secure Summation. This primitive is based on [23]. Initially, each silo F_i holds a private value v_i and all n silos agree on n distinct public parameters $\{u_i\}$. Each silo F_i then selects a random polynomial of degree $n-1$ in the form $t_i(x) = (\sum_{k=1}^{n-1} a_{ik}x^k) + v_i$, where a_{ik} is the random coefficient independently generated by silo F_i, and v_i denotes the private value (*i.e.*, local count) of silo F_i. These variables are kept secret from others by silo F_i. Next, each silo F_i evaluates its polynomial at the public parameters $\{u_1, \cdots, u_n\}$ and sends the resulting value $t_i(u_j)$ to every other silo F_j. Once the silo F_j receives all values $\{t_i(u_j)|i \neq j\}$ from the other silos, we have $S(u_j) = \sum_{i=1}^{n} t_i(u_j) = (\sum_{k=1}^{n-1}(u_j)^k \sum_{i=1}^{n} a_{ik}) + \sum_{i=1}^{n} v_i$. Afterward, this silo sends $S(u_j)$ to the query user. The user can interpret each $S(u_j)$ as a linear equation $S(u_j) = \sum_{k=1}^{n-1}(u_j)^k z_k + z_n$ in n unknown variables z_k, where $z_k = \sum_{i=1}^{n} a_{ik}$ (for $k < n$) and $z_n = \sum_{i=1}^{n} v_i$. Now, the user can solve the system of linear equations using the received coefficients $\{u_j\}$ and constants $\{S(u_j)\}$ via Gauss elimination, and obtain the unknown variable z_n (*i.e.*, the sum of v_i).

Secure Count Comparison. The primitive compares the constant k with the sum of each silo F_i's private range count v_i and prevents the leakage of either v_i or $\sum_{i=1}^{n} v_i$ to the silo F_j and the query user. The *main idea* is evaluating $X(\sum_{i=1}^{n} v_i - k)$ rather than directly computing $\sum_{i=1}^{n} v_i - k$ to avoid disclosing the actual sum of v_i, where X is a positive random number. Next, we implement this secure primitive by using existing secure multiplication protocol [18]. Specifically, this protocol [18] assumes that two multiplicands, X and Y, are partitioned into n shares x_i and y_i, where $X = \sum_{i=1}^{n} x_i$ and $Y = \sum_{i=1}^{n} y_i$. Each silo F_i holds the corresponding shares x_i and y_i, where x_i is randomly generated by this silo and $y_i = v_i - \frac{k}{n}$. Together, the multiplication XY happens to be $(\sum_{i=1}^{n} x_i)(\sum_{i=1}^{n} v_i - k)$. Finally, the comparison result is inferred from the sign of XY.

Secure Set Union. We implement this primitive based on the two-phase union method in [32] with additional optimizations. In the first phase, each silo appends its results into a global set, along with some fake records. Then, in the second phase, these fake records are removed from the set. To reduce the communication cost, the number of fake records should be as few as possible. Thus, we use the Laplace mechanism [36] in differential privacy to control the number of fake records. Moreover, by splitting the global set into batches, parallel executions are enabled for each silo to independently append and remove fake records from each batch, thereby resulting in a shorter latency.

Secure Distance Comparison. We leverage fully homomorphic encryption (FHE), the BGV scheme [12], to implement this primitive in three key steps.

(1) *Encrypt User's Data*: the query user encrypts their location (x_q, y_q) and the threshold r using the public key, *i.e.*, $E(x_q)$, $E(y_q)$, $E(r)$, where $E(\cdot)$ is the encryption function.

The encrypted data and public key are then sent to the data silo.

(2) *Compute Garbled Distance Difference*: by using FHE, the data silo computes the encrypted difference $E(\Delta)$ between the square of distance $\mathsf{dist}(l_q, l_o)$ and square of threshold r as follows:

$$\Delta = \mathsf{dist}(l_q, l_o)^2 - r^2 = (x_q - x_o)^2 + (y_q - y_o)^2 - r^2$$

$$E(\Delta) = (E(x_q) - E(x_o))^2 + (E(y_q) - E(y_o))^2 - E(r)^2$$

To further obfuscate the value of $E(\Delta)$, the silo applies a random polynomial function $f(\cdot)$ that only has odd powers and positive coefficients in each term, The obfuscation here prevents the user from inferring the exact distance $\mathsf{dist}(l_q, l_o)$ after decryption, thereby protecting the silo's location privacy.

(3) *Decrypt*: upon receiving $f(E(\Delta))$ from the silo, the user decrypts it with the secret key and obtains $f(\Delta)$. The final result is derived based on the sign of $f(\Delta)$ without knowing the exact value of Δ.

Secure Location Perturbation. We implement this primitive based on the BPL mechanism in [54]. This mechanism obfuscates the original location (x, y) in the polar coordinate system. The polar angle θ is uniformly sampled from $[0, 2\pi]$. The polar radius r is sampled based on the -1 branch of the Lambert W function. If the sampled radius r is too long, it will be truncated into a random value in $[0, R]$, where R is a safe upper bound of radius based on the privacy parameters ϵ, δ. The resulting perturbed locations are $(x+r\cos\theta, y+r\sin\theta)$. *Complexity Analysis.* The *secure summation* takes $O(n^3)$ time and $O(n^2)$ communication cost. The *secure count comparison* requires $O(n)$ time and $O(n^2)$ communication cost. The time complexity and communication cost of *secure set union* are $O(n+|S|)$, where $|S|$ is the output size. The *secure distance comparison* (between two parties) takes $O(1)$ time and communication cost, since the complexity of the BGV scheme [12] used for this primitive primarily depends on constant security parameters. The time complexity and communication cost of *secure location perturbation* are also $O(1)$ due to the usage of differential privacy mechanism.

6 Query interface

For easy usability, the query interface of Hu-Fu 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 users, while the detailed information of silos is hidden. This not only enables users to send queries without caring about the silo organization, but also protects the data privacy of individual silos.

We implement this unified federation view by extending the schema manager of Calcite [19], a popular query processing framework. In Calcite's schema manager, each table is independent and indivisible. We treat silos as an abstraction layer below the table of schema manager. This means each table comprises multiple silo objects, and each object records the identity information of its silo. The silo identities are used when executing secure primitives. Specifically, the query rewriter will attach the identity 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 with four keywords: `DWithin`, `kNN`, `Private_DWithin`, and `Private_kNN`. The first two keywords are used in asymmetric queries, and the last two are used in symmetric queries.

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

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

The `WHERE clause checks` whether the distance from p to an object in F is shorter than r. Similarly, an asymmetric federated kNN join on a relation R and federation F can be written in SQL as

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

The `WHERE` clause indicates whether a spatial object in F belongs to the kNN set of the query point $o \in R$.

In contrast, when locations in both R and F need protection, a symmetric federated kNN join in SQL is

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

s queries can be written as SQL similarly with these four keywords.

7 Evaluation

In this section, we first introduce the experimental setup (Sec. 7.1), and then present the overall performances of asymmetric queries (Sec. 7.2) and symmetric queries (Sec. 7.3), scalability tests (Sec. 7.4), and results with heterogeneous spatial databases across silos (Sec. 7.5).

7.1 Experimental setup

Datasets. Experiments are conducted on two datasets, with each object having a location and unique ID.

- *Multi-company Spatial Data in Beijing (BJ).* This dataset[1] 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 $\sim$ 42.0°N and 115.5°E $\sim$ 117.2°E. We use the dataset to simulate a real-world federation, where each company can be naturally regarded as a silo. During the evaluation, we vary the silo number n and queries without altering the spatial object distributions across silos.
- *OpenStreetMap (OSM).* This is a popular open dataset to evaluate spatial queries. 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 [9]. 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.

General-Purpose Baselines. As a data federation system, the evaluation first aims to compare Hu-Fu with existing general-purpose data federation systems: the GIS extensions of SMCQL [14] and Conclave [50].

- *SMCQL-GIS.* It adopts the principles of SMCQL [14], a garbled circuit (GC) based solution for relational data, to support spatial queries. We implement it with ObliVM [38], which is used in SMCQL for GC protocols across two silos (only) [50, 53]. Thus, it is only evaluated over two data silos.
- *Conclave-GIS.* It adopts the principles of Conclave [50], the secret sharing (SS) based solution for relational data, to support spatial queries. It is implemented with a different SS based library, MP-SPDZ [33], rather than Sharemind [18] in the original Conclave, since Sharemind is devised for only three silos [24] and it is a commercial library. In contrast, MP-SPDZ is a popular open-source library that supports more than three silos based on SS.
- *SMCQL-GISext & Conclave-GISext* are their variants 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.

Specialized Baselines. Beyond these general-purpose data federation systems, the evaluation also compares Hu-Fu with the following specialized baselines.

- *Additional Baselines for Asymmetric Queries*. The plaintext baseline Public directly collects local results from each silo without any secure operation, and serves as the upper bound of query efficiency.
- *Additional Baselines for Symmetric Queries*. We consider two more baselines for symmetric queries: LFHE [34] and PINED-RQ++ [43]. LFHE [34] is an industrial solution that utilizes Leveled Fully Homomorphic Encryption (LFHE) and two mutually untrusted servers to securely answer (exact) kNN queries over multiple data silos. This solution can be easily extended to support secure range query and counting over a spatial data federation. By contrast, PINED-RQ++ [43] leverages a differentially private index (*e.g.*, grid index for spatial data) and AES encryption [29] to approximately answer range queries with small errors. However, this method assumes that the user has access to data objects outside the query answer, potentially violating the data privacy requirement. Nevertheless, we select PINED-RQ++ [43] as a baseline for comparison, since it also utilizes differential privacy for filtering before verifying each candidate through encryption.

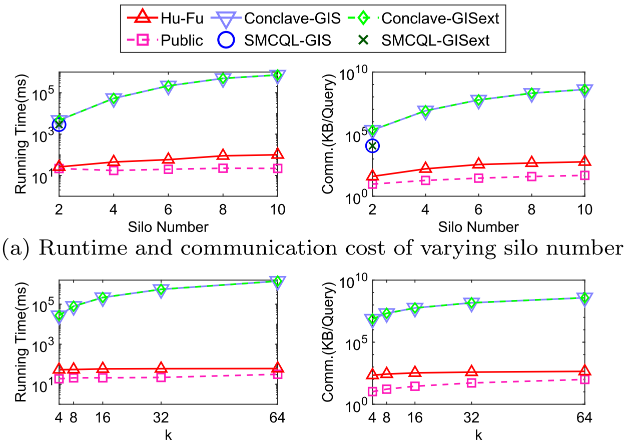
Metrics. We assess the query efficiency by two metrics:

(1) *Running time* is the time cost from receiving the query to returning the query answer to the user.

(2) *Communication cost* is the total network communication among the user and all data silos.

Implementation. We use PostgreSQL 10.15 with PostGIS extension as the default spatial database for all silos. To show the support of heterogeneous spatial data systems by Hu-Fu, we also use MySQL 5.7 [8], SpatiaLite [11], GeoMesa 3.0.0 [26], Simba 1.0 [57], and SpatialHadoop 2.4.3 [21] as

[1] https://share.weiyun.com/z4QfVhVv



(a) Runtime and communication cost of varying silo number

(b) Runtime and communication cost of varying k

Fig. 6 Performance of federated kNN query

different silos, as will be explained in Sec. 7.5. They all use spatial indexes (R-Tree in PostGIS, Simba, SpatialHadoop, and MySQL, and R*-Tree in SpatiaLite, and Z-Curve in GeoMesa) to speed up plaintext primitives by up to 2042× (see our full paper [7]). Among the compared solutions, LFHE and PINED-RQ++ are implemented in C++, while the others are implemented in Java. The reason for using C++ for LFHE and PINED-RQ++ is due to the lack of robust and open-source libraries in Java for the encryption methods (*e.g.*, CKKS [12]) they utilize.

7.2 Experiments on asymmetric queries

Parameter Setting. In this experiment, we 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 [57]. When evaluating the query-specific parameters, we use 6 silos by default.

Environment. We run this experiment 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.

Table 4 Improvement with DP in federated kNN

Silo number	2	4	6	8	10
Running time	2.9%	10.3%	19.6%	16.2%	14.0%
Communication	32.6%	31.5%	27.4%	39.4%	47.7%

7.2.1 Performance of federated kNN query

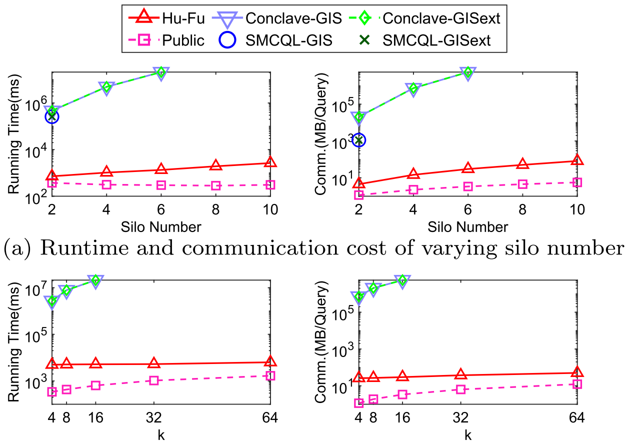
Figure 2 shows the runtime and communication cost of (asymmetric) federated kNN query. Hu-Fu is 109.6× to 7, 198.8× faster than SMCQL-GIS and Conclave-GIS, and has 2 to 5 orders of magnitude lower communication cost. When the number of silos increases from 2 to 10, the runtime and communication cost 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 since the secure comparison and set union used in this query grow linearly with the silo number. Compare with Conclave-GIS and SMCQL-GIS, the runtime and communication cost of Conclave-GISext and SMCQL-GISext marginally increase (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. 6b. 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 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.

To show the improvement of DP optimization in kNN queries, we list the percentage of running time and communication cost reduced by DP in Table 4. 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

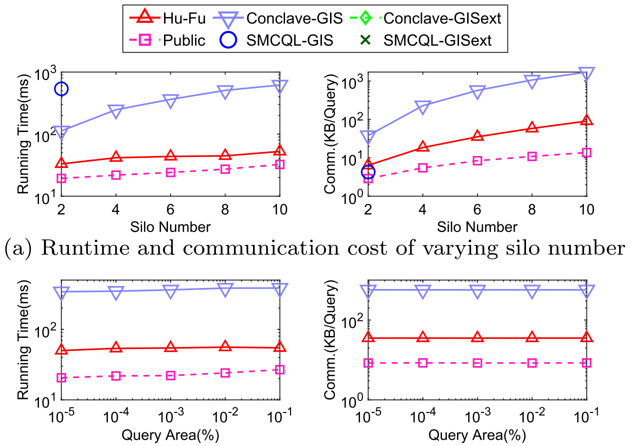
Figure 7a shows the results of (asymmetric) federated kNN join. Results of Conclave-GIS and Conclave-GISext with 8-



(a) Runtime and communication cost of varying silo number

(b) Runtime and communication cost of varying k

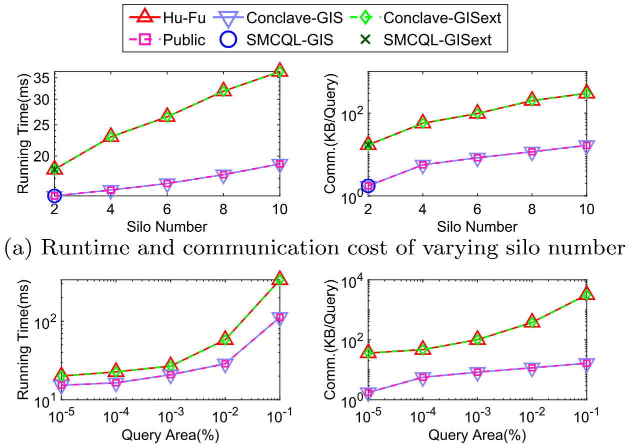
Fig. 7 Performance of federated kNN join



(a) Runtime and communication cost of varying silo number

(b) Runtime and communication cost of varying query area

Fig. 8 Performance of federated range counting



(a) Runtime and communication cost of varying silo number

(b) Runtime and communication cost of varying query area

Fig. 9 Performance of federated range query

10 silos are omitted since they incur over 6 hours for a single query. Hu-Fu is the most efficient, which is up to 360.2× and 15, 814.2× faster than SMCQL-GIS and Conclave-GIS with 247.8× and 185, 151.0× lower communication cost. The time and communication cost of SMCQL-GISext and Conclave-GISext slightly increase over SMCQL-GIS and Conclave-GIS.

Figure 7b illustrates the impact of k. As k increases above 32, Conclave-GIS and Conclave-GISext require longer than 6 hours to process a federated kNN query. Thus, we can only provide their partial results (when $k \leq 16$). In contrast, Hu-Fu demonstrates superior advantages in the efficiency, achieving at least 553× faster and 27,404× lower communication cost compared to Conclave-GIS. As k increases from 4 to 64, the running time and communication cost of Hu-Fu rise by 28% and 48% respectively. The experimental trends for the federated kNN join closely resemble those observed in the federated kNN query, since a federated kNN join is decomposed into multiple federated kNN queries for all the compared solutions.

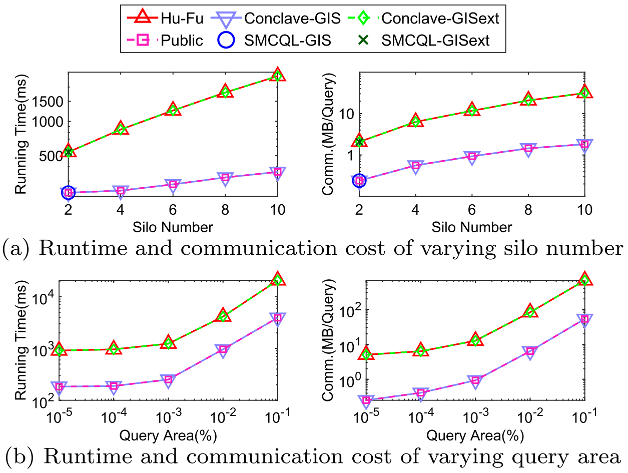
7.2.3 Performance of federated range counting

Figure 8 shows the results of (asymmetric) 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. 8b. 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

Figure 9 illustrates the results of (asymmetric) 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



(a) Runtime and communication cost of varying silo number

(b) Runtime and communication cost of varying query area

Fig. 10 Performance of federated distance join

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 our full paper [7]).

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 usage 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 size of data returned. As shown in Fig. 9b, 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

Figure 10 presents the performance of (asymmetric) 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. 9). 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.

7.2.6 Summary of major findings

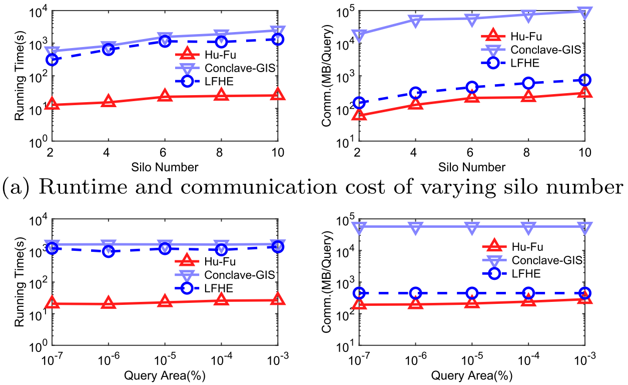
We have observed the following findings in the experiments of asymmetric queries.

- 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 time cost 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.
- The experimental trends of federated kNN join and distance join are similar to those of federated kNN query and range query for all compared solutions. This is reasonable since a federated kNN join or distance join is decomposed into a series of federated kNN queries or range queries.

7.3 Experiments on symmetric queries

Parameter Setting. The parameter configurations for the query workloads are identical to those introduced in Section 7.2. Beyond these parameters, the privacy budget ϵ also affects the query performance of Hu-Fu. In general, a smaller ϵ (*i.e.*, stricter privacy preservation) leads to lower efficiency than a larger ϵ. Please refer to our full paper [7] for the evaluation of varying ϵ in Hu-Fu. Moreover, due to page limitations, we have omitted reporting the results of the federated kNN join. However, the query efficiency of Hu-Fu's federated kNN join can be inferred from the results of its federated kNN query, as evidenced by previous experiments. We also omit the results of SMCQL-GIS, SMCQL-GISext, and Conclave-GISext, since their results are similar to Conclave-GIS when answering symmetric queries. Additional baselines, LFHE [34] and PINED-RQ++ [43], are involved in this experiment, where PINED-RQ++ is limited to federated range query and distance join.

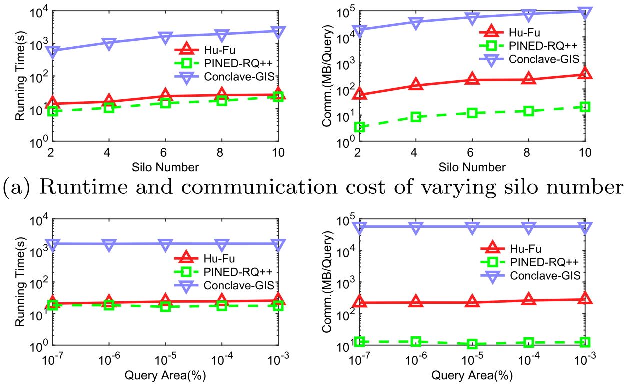
Environment. Due to the expired funding support, the hardware environment for testing asymmetric queries is no longer accessible, so all solutions to symmetric queries are tested in a new hardware environment. Specifically, this new environment is composed of 5 cloud servers. Each server has Intel Xeon(R) Platinum 8361HC CPU 2.60GHZ processors and 32GB memory with Ubuntu 18.04 LTS OS. The network



(a) Runtime and communication cost of varying silo number

(b) Runtime and communication cost of varying query area

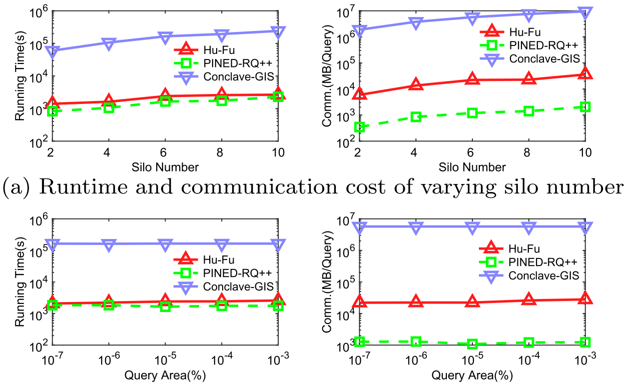
Fig. 11 Performance of federated range counting



(a) Runtime and communication cost of varying silo number

(b) Runtime and communication cost of varying query area

Fig. 12 Performance of federated range query



(a) Runtime and communication cost of varying silo number

(b) Runtime and communication cost of varying query area

Fig. 13 Performance of federated distance join

bandwidth between servers is up to 1.5 GB/s and may fluctuate slightly at different times. Four of the five servers act as data silos. Since this experiment requires up to 10 data silos, each of these four servers can host up to 3 data silos using different processes. The remaining server is only used as the user's client, facilitating parallel data transmission between data silos for compare solutions. For each query type, we generate 50 queries, repeat 10 times for each query, and report the average results.

7.3.1 Performance of federated range counting

Figure 11 shows the results of (symmetric) federated range counting. Hu-Fu always outperforms the compared baselines in both running time and communication cost. Hu-Fu takes up to 98.9× shorter runtime and up to 410.7× lower communication cost than Conclave-GIS. Hu-Fu is also up to 56.6× faster with up to 2.7× lower communication cost compared to LFHE.

We can also observe that as the silo number increases from 2 to 10, the running time and communication cost of all compared algorithms tend to increase. This pattern is reasonable, as more data silos require more secure computations among them. When the query area expands from small to large, the runtime and communication overhead of Hu-Fu increase slightly. This trend in Hu-Fu is attributable to the increase of candidates for secure distance comparisons.

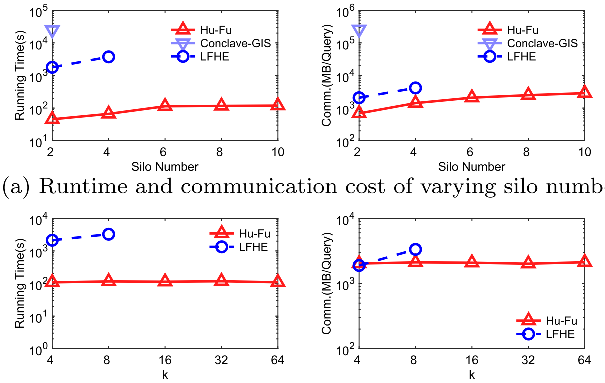
7.3.2 Performance of federated range query

Figure 12 illustrates the performance of (symmetric) federated range queries. The results of LFHE are not reported because the running time is over 1 hour and the memory consumption exceeds the server configuration (32 GB). In fact, LFHE can only handle a small-scale dataset. For example, LFHE already takes 13 minutes and 3 MB of communication cost when the data size is 10^4. Although PINED-RQ++ is more efficient than Hu-Fu and Conclave-GIS, it suffers from *two major drawbacks*: (1) it leaks locations of the candidate data objects that are not part of the true answer to the query user and (2) it is only capable of retrieving approximation results. For example, the recall of PINED-RQ++ is 72.8%-96.8% when varying the silo number and 72.2%-92.3% when varying the query area.

Aside from PINED-RQ++, Hu-Fu achieves the best performance in the efficiency and security. It takes at least 42.7× shorter time with at least 204.2× lower communication cost compared to Conclave-GIS. Figure 12 exhibits a similar trend in the efficiency variation of Hu-Fu when comparing to the results in Fig. 11 (for federated range counting). This similarity arises because the query decomposition plans for symmetric federated range queries and counting are quite identical. The main difference lies in the additional secure set union operator required for federated range queries.

7.3.3 Performance of federated distance join

Figure 13 presents the performance of (symmetric) federated distance joins. Here, a symmetric federated distance join



(a) Runtime and communication cost of varying silo number

(b) Runtime and communication cost of varying k

Fig. 14 Performance of federated kNN query

Table 5 Improvement with upper bound in Lemma 1

k	4	8	16	32	64
Running time	6.3×	4.2×	4.0×	3.6×	3.4×
Communication	5.0×	3.9×	3.6×	3.4×	3.1×

employs the same strategy as the asymmetric federated distance join, treating the join operation as multiple independent federated range queries, with $|R| = 100$. Therefore, the relative performance of all methods here is similar to that of symmetric federated range query (see Fig. 12). Compared with PINED-RQ++, although Hu-Fu incurs higher communication cost, it takes nearly the same amount of running time and offers a more secure solution that produces the exact query answer. Compared to Conclave-GIS, Hu-Fu is up to 90.7× faster and up to 332.5× lower in communication cost. By contrast, LFHE is the least efficient method and cannot respond to a join query within 6 hours, so we cannot report its result in Fig. 13.

7.3.4 Performance of federated kNN query

Figure 14 shows the running time and communication cost of (symmetric) federated kNN queries. We cannot report some results of LFHE and Conclave-GIS because they cannot terminate within 1 hour to process a single query or the maximum memory usage is beyond the limitation (32 GB) of the cloud server. Additionally, PINED-RQ++ is unable to handle KNN queries, so it is excluded in this evaluation. According to the experimental results, Hu-Fu is the most efficient solution to federated kNN queries. For instance, Hu-Fu is up to 551.8× faster than Conclave-GIS and 56.2× faster than LFHE, with up to 380.0× and 3.0× lower communication overhead than them, respectively. Similar to previous results, the time cost of Hu-Fu gradually increases as the silo number increases. By contrast, the efficiency of Hu-Fu does not notably change as k increases.

We also evaluate the improvement of using the upper bound in Lemma 1. As shown in Table 5, this optimization improves the running time by up to 6.3× and reduces the communication cost by up to 5.0×.

7.3.5 Summary of major findings

The major findings in the experiments of symmetric queries are summarized as follows.

- Hu-Fu is at least 42.7× faster than Conclave-GIS, with at least 204.2× lower communication overhead. Compared to LFHE, Hu-Fu is up to 56.6× faster with up to 3.0× lower communication overhead.
- Although PINED-RQ++ is more efficient than the others, it suffers from three significant drawbacks: (1) violations on the data privacy, (2) inability to provide exact answers, and (3) limited support to only federated range queries. By contrast, our Hu-Fu can address all these drawbacks effectively.
- When comparing with the evaluations of asymmetric queries in Sec. 7.2, it is evident that running time and communication overhead both escalate when Hu-Fu or Conclave-GIS processes symmetric queries. Consequently, symmetric queries pose a greater challenge than asymmetric queries, primarily due to the additional concern for query privacy.

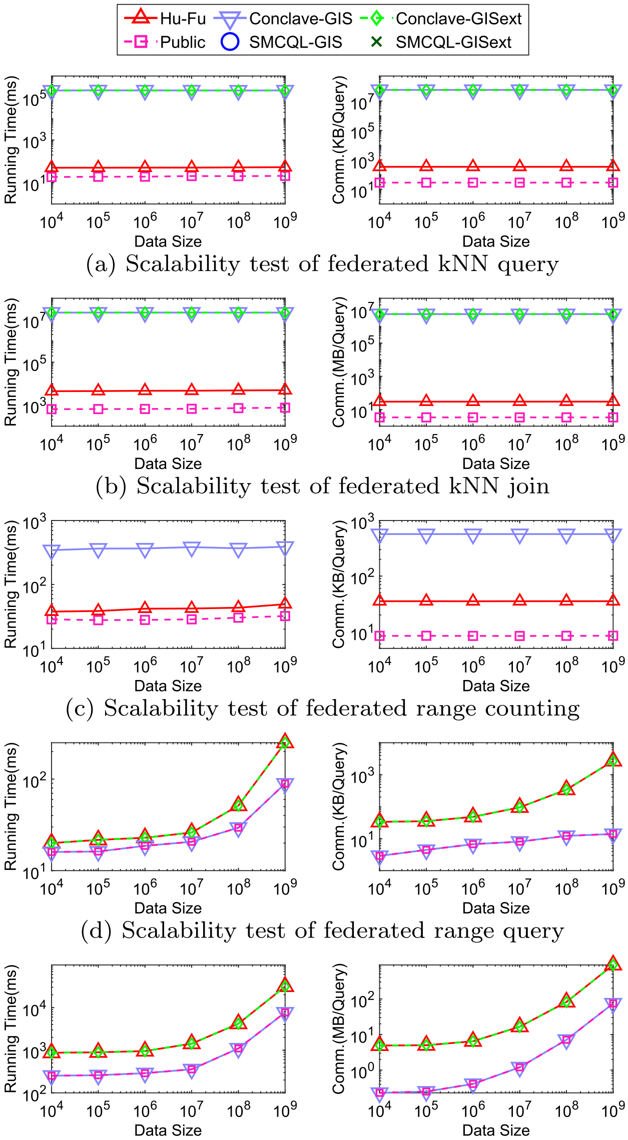
Remark. To assess the impact of query privacy on time efficiency, we can compare the running time of asymmetric and symmetric queries in the new hardware environment. Our evaluation shows that the baseline Conclave-GIS takes 4, 054× longer to protect query privacy in federated range queries, while our solution Hu-Fu reduces this gap to 59×. Due to page limitations, please see our full paper [7] for detailed results.

7.4 Experiments on scalability tests

In the following, we report the results of scalability tests on asymmetric queries and symmetric queries in Sec. 7.4.1 and Sec. 7.4.2, respectively.

7.4.1 Scalability test on asymmetric queries

Parameter Setting. In the following, 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%. Since SMCQL-GIS and SMCQL-GISext only support two silos, they are excluded



(a) Scalability test of federated kNN query

(b) Scalability test of federated kNN join

(c) Scalability test of federated range counting

(d) Scalability test of federated range query

(e) Scalability test of federated distance join

Fig. 15 Scalability test on asymmetric queries

in the scalability test. The running time and communication cost on all five spatial queries are shown in Fig. 15.

Result and Analysis. 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. 15a-15c). 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.

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, an asymmetric federated range query takes 250 ms running time and 2.6 MB communication cost on the data size of 10^9.

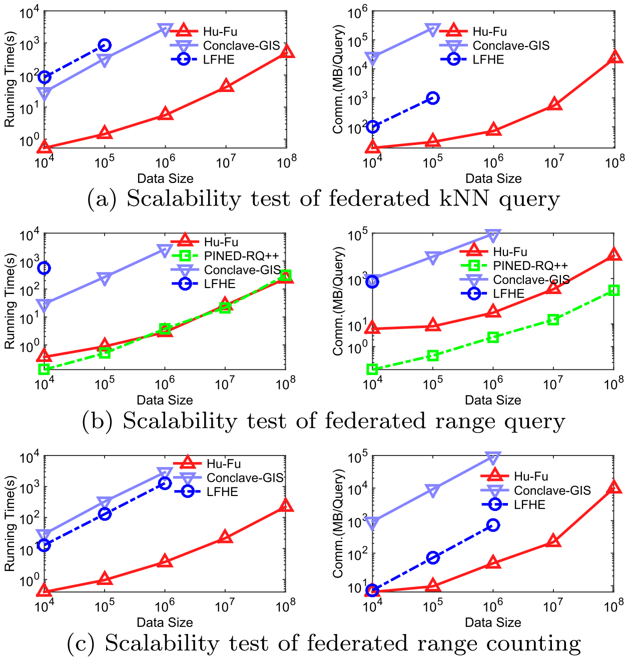
7.4.2 Scalability test on symmetric queries

Parameter Setting. As for symmetric queries, we evaluate the scalability of Hu-Fu by scaling the OSM dataset, gradually increasing the total number of spatial objects from 10^4 to 10^8. The baseline selection here is identical to that in Sec. 7.3. Although we omit the results of spatial joins due to page limitations, the scalability of the federated distance join and kNN join can be inferred from the scalability of the federated range query and kNN query, as demonstrated in the previous experiments in Sec. 7.2 and Sec. 7.3.

Result and Analysis. As shown in Fig. 16, for any data size, Hu-Fu significantly outperforms Conclave-GIS and LFHE in both running time and communication overhead for all the tested queries. For example, in Fig. 16a, Hu-Fu is at least 54.8× and 1375.7× faster than Conclave-GIS and LFHE, respectively. Neither Conclave-GIS nor LFHE can efficiently process these queries over large-scale datasets (*e.g.*, when the data size is over 10^6). After running for more than 6 hours, neither of them have terminated, so we are unable to obtain their full results. Moreover, when processing symmetric federated range queries, both Conclave-GIS and LFHE take at least 2 orders of magnitude higher communication cost than Hu-Fu.

By contrast, PINED-RQ++ is slightly faster than Hu-Fu (no more than 2.8×) and has lower communication overhead in Fig. 16b. However, since PINED-RQ++ may violate the data privacy during the query processing and can only obtain approximate results, the performance gap is acceptable. For instance, the recall of PINED-RQ++ can be lower than 80%, and such a low recall may lead to an unsatisfying service experience in our motivation scenarios like contact tracing, where accurate results are crucial.

Takeaways. Unlike the scalability tests for asymmetric queries, the efficiency of symmetric federated range query, range counting, and kNN query is highly sensitive to the data size. Specifically, when processing *asymmetric queries*, all



(a) Scalability test of federated kNN query

(b) Scalability test of federated range query

(c) Scalability test of federated range counting

Fig. 16 Scalability test on symmetric queries

compared solutions leverage plaintext operators to filter most of the data objects. For example, a plaintext range query can ensure no false positive candidate for asymmetric federated range queries, while a plaintext kNN query results in only k false positive candidates for asymmetric federated kNN queries. Consequently, the number of false positive candidates is (almost) independent of data size.

However, when processing *symmetric queries* that require additional protections for the query location, although plaintext operators are still used to reduce the candidate size, no solution can ensure a constant number of false positive candidates. Even in our Hu-Fu, the number of candidate objects awaiting secure verification after filtering by plaintext operators is proportional to the data size. This contributes to the increased time and communication overhead required to process symmetric queries compared to asymmetric ones.

7.5 Experiments on heterogeneous data 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 [10], MySQL [8], SpatiaLite [11], Simba [57], GeoMesa [26], and SpatialHadoop [21]. Other parameters are set as the default values as in Sec. 7.2.

Figure 17 plots the running time breakdown *i.e.*, plaintext vs. secure primitives for radius-unknown (*i.e.*, asymmetric federated kNN query) and radius-known (*i.e.*, asymmetric federated range counting) queries. We make the following observations.

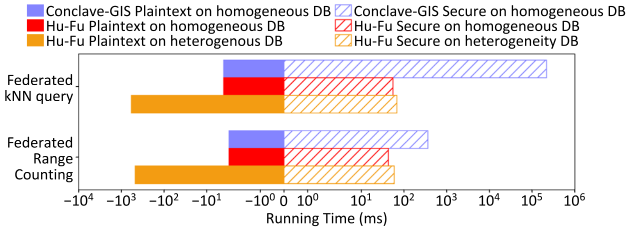


Fig. 17 Running time breakdown

- Given homogeneous underlying spatial databases (*i.e.*, PostGIS), our Hu-Fu significantly reduces the running time of secure primitives *e.g.*, 3, 935.4× 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). It takes even longer time 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 data 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 [40]) in Hadoop (*e.g.*, SpatialHadoop [21]) or Spark (*e.g.*, Simba [57]). 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 [30], where data is encrypted and stored in a third-party platform (*e.g.*, a cloud platform) to process queries securely. For example, existing work [22, 34, 55, 58] study the secure kNN query on encrypted databases and prior studies [43, 48, 52, 56] 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 and SEAL [12]) 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. Besides, our experiments demonstrate that extending these solutions [34, 43] to the scenario of the data federation can be either insecure or inefficient.

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. Early research on federated databases focused on finding solutions to access data in autonomous databases [41], while recent studies on federated databases support diverse data types, *e.g.*, on federated graph databases [51]. Note that the autonomous database here means that data can be only managed by its held silo.

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 [14] is probably the first secure query processing solution over a data federation and Conclave [50] is the state-of-the-art solution. More recent studies explored efficient solutions to join queries [31, 37, 53] in a data federation. All these studies adopt SMC techniques to achieve secure query processing for *relational* data with *exact* results.

Exact federated queries have been explored for various data types. Our preliminary work [46] and its accompanying demonstration system [42] focus on exact federated queries over spatial data federation. Zhang *et al.* [62] propose an efficient method that leverages the Intel SGX to securely perform similarity searches over a data federation under metric spaces. For example, the metric distance can be the graph edit distance for graph data or the edit distance for sequence data.

Existing studies also investigate *approximate* query processing over a data federation. SAQE [16], Cryptϵ [20], and Shrinkwrap [15] use differential privacy to trade off between accuracy and efficiency in processing relational queries. Others study approximate kNN queries [61] and range counting [35, 44] over a spatial data federation. In contrast, we focus on *exact* query processing, since accurate results can be crucial for spatial applications like contact tracing [25].

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 [14] and Conclave [50], the state-of-the-arts for relational data. Moreover, unlike most existing studies that solely focus on protecting data privacy under this emerging scenario, Hu-Fu also considers the application need for preserving the query privacy. For example, in applications like contact tracing, spatial queries often contain sensitive location data of patients, thereby necessitating the protection on the query privacy.

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 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, *e.g.*, spatial keyword search.

Acknowledgements This work is partially supported by National Key Research and Development Program of China under Grant No. 2023YFF0725103, National Science Foundation of China (NSFC) (Grant Nos. 62425202, U21A20516, 62336003), Beijing Natural Science Foundation (Z230001), the Fundamental Research Funds for the Central Universities No. JK2024-03, and Didi Collaborative Research Program. Zimu Zhou's research is supported by Chow Sang Sang Group Research Fund No. 9229139. Lei Chen's work is partially supported by National Key Research and Development Program of China Grant No. 2023YFF0725100, National Science Foundation of China (NSFC) under Grant No. U22B2060, Guangdong-Hong Kong Technology Innovation Joint Funding Scheme Project No. 2024A0505040012, the Hong Kong RGC GRF Project 16213620, RIF Project R6020-19, AOE Project AoE/E-603/18, Theme-based project TRS T41-603/20R, CRF Project C2004-21G, Guangdong Province Science and Technology Plan Project 2023A0505030011, Guangzhou municipality big data intelligence key lab, 2023A03J0012, Hong Kong ITC ITF grants MHX/078/21 and PRP/004/22FX, Zhujiang scholar program 2021JC02X170, Microsoft Research Asia Collaborative Research Grant, HKUST-Webank joint research lab, and 2023 HKUST Shenzhen-Hong Kong Collaborative Innovation Institute Green Sustainability Special Fund, from Shui On Xintiandi and the InnoSpace GBA. Yuxiang Zeng is the corresponding author.

References

1. Facebook scandal: Who is selling your personal data? (2022) https://www.bbc.com/news/technology-44793247
2. Acxiom (2024) https://www.acxiom.com
3. AMAP (2024) https://mobility.amap.com
4. China Mobile (2024a) https://www.chinamobileltd.com
5. China Telecom (2024b) http://www.chinatelecom-h.com
6. Communication travel card (2024) https://xc.caict.ac.cn/
7. Hu-Fu: Efficient and secure spatial queries over data federation (2024) https://github.com/BUAA-BDA/OpenHuFu/
8. MySQL (2024) https://www.mysql.com
9. OpenStreetMap (2024) https://www.openstreetmap.org
10. PostGIS (2024) https://www.postgis.org
11. SpatiaLite (2024) https://www.gaia-gis.it/fossil/libspatialite/index
12. Acar, A., Aksu, H., Uluagac, A.S., Conti, M.: A survey on homomorphic encryption schemes: theory and implementation. ACM Comput. Surv. **51**(4), 791–7935 (2018)
13. Andrés, M.E., Bordenabe, N.E., Chatzikokolakis, K., Palamidessi, C.: Geo-indistinguishability: differential privacy for location-based systems. CCS (2013). https://doi.org/10.1145/2508859.2516735
14. Bater, J., Elliott, G., Eggen, C., Goel, S., Kho, A.N., Rogers, J.: SMCQL: secure query processing for private data networks. PVLDB **10**(6), 673–684 (2017)
15. Bater, J., He, X., Ehrich, W., Machanavajjhala, A., Rogers, J.: ShrinkWrap: efficient SQL query processing in differentially private data federations. PVLDB **12**(3), 307–320 (2018)
16. Bater, J., Park, Y., He, X., Wang, X., Rogers, J.: SAQE: practical privacy-preserving approximate query processing for data federations. PVLDB **13**(11), 2691–2705 (2020)
17. Bettini, C., Jajodia, S., Samarati, P., Wang, X.S. (eds.): Privacy in Location-Based Applications. Springer (2009)
18. Bogdanov, D., Laur, S., Willemson, J.: Sharemind: A framework for fast privacy-preserving computations. ESORICS (2008). https://doi.org/10.1007/978-3-540-88313-5_13
19. Calcite (2024) https://calcite.apache.org/
20. Chowdhury, A.R., Wang, C., He, X., Machanavajjhala, A., Jha, S.: Cryptϵ: crypto-assisted differential privacy on untrusted servers. SIGMOD (2020). https://doi.org/10.1145/3318464.3380596
21. Eldawy, A., Mokbel, M.F.: SpatialHadoop: a mapreduce framework for spatial data. ICDE, Seoul, Korea (2015)
22. Elmehdwi, Y., Samanthula, B.K., Jiang, W. Secure k-nearest neighbor query over encrypted data in outsourced environments. In: ICDE, pp 664–675 (2014)
23. Emekçi, F., Sahin, O.D., Agrawal, D., Abbadi, A.E.: Privacy preserving decision tree learning over multiple parties. Data Knowl. Eng. **63**(2), 348–361 (2007)
24. Evans, D., Kolesnikov, V., Rosulek, M.: A pragmatic introduction to secure multi-party computation. Found. Trend. Priv. Secur. **2**(2–3), 70–246 (2018)
25. Ferretti, L., Wymant, C., Kendall, M., Zhao, L., Nurtay, A., Abeler-Dörner, L., Parker, M., Bonsall, D., Fraser, C.: Quantifying sars-cov-2 transmission suggests epidemic control with digital contact tracing. Science **368**(6491), eabb6936 (2020)
26. GeoMesa (2024) https://www.geomesa.org/
27. Gertz, M., Jajodia, S. (eds.): Handbook of database security - applications and trends. Springer (2008)
28. Gkoulalas-Divanis, A., Bettini, C. (eds.): Handbook of mobile data privacy. Springer (2018)
29. Goldreich, O.: Foundations of cryptography: volume 2, basic applications. Cambridge university press (2009)
30. Hacigümüs, H., Iyer, B.R., Li, C., Mehrotra, S.: Executing SQL over encrypted data in the database-service-provider model. In: SIGMOD, pp 216–227 (2002)
31. Han, F., Zhang, L., Feng, H., Liu, W., Li, X.: Scape: Scalable collaborative analytics system on private database with malicious security. In: ICDE, pp 1740–1753 (2022)
32. Jurczyk, P., Xiong, L.: Information sharing across private databases: Secure union revisited. In: PASSAT/SocialCom, pp 996–1003 (2011)
33. Keller, M.: MP-SPDZ: A versatile framework for multi-party computation. In: CCS, pp 1575–1590 (2020)
34. Kesarwani, M., Kaul, A., Naldurg, P., Patranabis, S., Singh, G., Mehta S., Mukhopadhyay, D.: Efficient secure k-nearest neighbours over encrypted data. In: EDBT, pp 564–575 (2018)
35. Li, M., Zeng, Y., Chen, L. Efficient and accurate range counting on privacy-preserving spatial data federation. In: DASFAA, pp 317–333 (2023)
36. Li, N., Lyu, M., Su, D., Yang, W.: Differential privacy: from theory to practice. Morgan Claypool Publish. (2016). https://doi.org/10.1007/978-3-031-02350-7
37. Li, S., Zeng, Y., Wang, Y., Zhong, Y., Zhou, Z., Tong, Y.: An experimental study on federated equi-joins. IEEE Trans. Knowl. Data Eng. **36**(9), 4443–4457 (2024)
38. Liu, C., Wang, X.S., Nayak, K., Huang, Y., Shi, E. ObliVM: A programming framework for secure computation. In: S&P, pp 359–376 (2015)
39. Liu, F., Zheng, Z., Shi, Y., Tong, Y., Zhang, Y.: A survey on federated learning: a perspective from multi-party computation. Frontiers Comput Sci **18**(3), 181336 (2024)
40. Mamoulis, N.: Spatial Data Management. Synthesis Lectures on Data Management, Morgan & Claypool Publishers (2011)
41. Özsu, M.T., Valduriez, P.: Principles of Distributed Database Systems, 4th edn. Springer (2020)
42. Pan, X., Tong, Y., Xue, C., Zhou, Z., Du, J., Zeng, Y., Shi, Y., Zhang, X., Chen, L., Xu, Y., Xu, K., Lv, W.: Hu-fu: A data federation system for secure spatial queries. PVLDB **15**(12), 3582–3585 (2022)
43. Sahin, C., Allard, T., Akbarinia, R., Abbadi, A.E., Pacitti, E. A differentially private index for range query processing in clouds. In: ICDE, pp 857–868 (2018)
44. Shi, Y., Tong, Y., Zeng, Y., Zhou, Z., Ding, B., Chen, L.: Efficient approximate range aggregation over large-scale spatial data federation. IEEE Trans. Knowl. Data Eng. **35**(1), 418–430 (2023)
45. Sun, N., Wang, W., Tong, Y., Liu, K.: Blockchain based federated learning for intrusion detection for internet of things. Frontiers Comput Sci **18**(5), 185328 (2024)
46. Tong, Y., Pan, X., Zeng, Y., Shi, Y., Xue, C., Zhou, Z., Zhang, X., Chen, L., Xu, Y., Xu, K., Lv, W.: Hu-Fu: efficient and secure spatial queries over data federation. PVLDB **15**(6), 1159–1172 (2022)
47. Tong, Y., Zeng, Y., Zhou, Z., Liu, B., Shi, Y., Li, S., Xu, K., Lv, W.: Federated computing: query, learning, and beyond. IEEE Data Eng Bull **46**(1), 9–26 (2023)
48. Tran, H.V., Allard, T., d'Orazio, L., Abbadi, A.E.: Range query processing for monitoring applications over untrustworthy clouds. In: EDBT, pp 666–669 (2019)
49. Voigt, P., Von dem Bussche, A.: The EU General Data Protection Regulation (GDPR): A Practical Guide, vol. 10. Springer (2017)
50. Volgushev, N., Schwarzkopf, M., Getchell, B., Varia, M., Lapets, A., Bestavros, A.: Conclave: secure multi-party computation on big data. In: EuroSys, pp 3:1–3:18 (2019)
51. Vu, X., Ait-Mlouk, A., Elmroth, E., Jiang, L.: Graph-based interactive data federation system for heterogeneous data retrieval and analytics. In: WWW, pp 3595–3599 (2019)
52. Wang, P., Ravishankar, C.V.: Secure and efficient range queries on outsourced databases using rp-trees. In: ICDE, pp 314–325 (2013)
53. Wang, Y., Yi, K. Secure Yannakakis: Join-aggregate queries over private data. In: SIGMOD, pp 1969–1981 (2021)
54. Wang, Y., Zeng, Y., Xu, Y., Zhou, Z., Tong, Y. Efficient and private federated trajectory matching. CoRR abs/2312.12012 (2024)

55. Wong, W.K., Cheung, D.W., Kao, B., Mamoulis, N. Secure knn computation on encrypted databases. In: SIGMOD, pp 139–152 (2009)
56. Wu, S., Li, Q., Li, G., Yuan, D., Yuan, X., Wang, C.: ServeDB: Secure, verifiable, and efficient range queries on outsourced database. In: ICDE, pp 626–637 (2019)
57. Xie, D., Li, F., Yao, B., Li, G., Zhou, L., Guo, M.: Simba: Efficient in-memory spatial analytics. In: SIGMOD, pp 1071–1085 (2016)
58. Yao, B., Li, F., Xiao, X.: Secure nearest neighbor revisited. In: ICDE, pp 733–744 (2013)
59. Ye, J.: Transportation: A data driven approach. In: KDD, p 3183 (2019)
60. Zeng, Y., Tong, Y., Chen, L.: Last-mile delivery made practical: an efficient route planning framework with theoretical guarantees. PVLDB **13**(3), 320–333 (2019)
61. Zhang, K., Tong, Y., Shi, Y., Zeng, Y., Xu, Y., Chen, L., Zhou, Z., Xu, K., Lv, W., Zheng, Z. Approximate k-nearest neighbor query over spatial data federation. In: DASFAA, pp 351–368 (2023)
62. Zhang, X., Wang, Q., Xu, C., Peng, Y., Xu, J.: FedKNN: secure federated k-nearest neighbor search. Proc. ACM Manage. Data. **2**(1), 1–26 (2024)

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.