Efficient and Secure Skyline Queries over Vertical Data Federation

Yuanyuan Zhang, Yexuan Shi, Zimu Zhou, *Member, IEEE,* Chunbo Xue, Yi Xu, Ke Xu, and Junping Du

Abstract—Skyline is a primitive operation in multi-objective decision applications and there is a growing demand to support such operations over a data federation, where the entire dataset is separately held by multiple data providers (a.k.a., silos). Data federations notably increase the amount of data available for data-intensive applications such as commercial recommendation and location based services. Yet they also challenge the conventional implementation of skyline queries because the raw data cannot be shared within the federation and the secure computation cross silos can be two or three orders of magnitude slower than plaintext computation. These constraints render existing solutions inefficient on data federation. In this work, we propose a novel local dominance based framework for efficient skyline queries over a vertical data federation. We decompose the skyline query into plaintext local dominance computations and secure result aggregations, which can perform as many computations in plaintext as possible without compromising security. We further propose a dedicate private set intersection based algorithm to accelerate the query processing. Extensive evaluations on both synthetic and real-world datasets show that compared with general-purpose secure multi-party computation techniques, our solutions reduce the time cost by up to 35.4× and communication cost by two orders of magnitude respectively.

Index Terms—Skyline, Data Federation, Secure Multi-party Computation, Private Set Intersection.

1 Introduction

Skyline queries are widely used to obtain preferred answers from the database by providing the orderings of attribute values [1], [2], [3]. The result of a skyline query consists of input tuples for which there is no input tuple having better or equal values in all the attributes and a better value in at least one attribute. Such queries are crucial for various big data applications such as location based services [3], multi-objective decisions [3], commercial recommendation [4], *etc.*

Recently, there is an emerging trend to scale these applications from a single service provider (*a.k.a.*, silo) to a federation of multiple silos for better quality of service [5], [6], [7]. Each silo in the federation holds one part of the entire data (*i.e.*, columns) for the same samples, and interact with other silos without revealing its own raw data, and we call such a federation as a *vertical data federation*. For example, a bank and an insurance company want to make customer recommendation collaboratively. The bank holds the deposit information about customers, and the insurance company holds the insurance records of these customers. Without the data federation, the bank and insurance company can only explore potential customers with their own data. For example, the insurance company cannot learn the deposits of their customers without the data federation. Thus, it can only find potential customers based on its own insurance data. With a skyline query on a federation between them, the insurance company can find customers that have large deposits. Similarly, the bank can find potential customers that have larger insurance orders but still not have many deposits. Data federation creates this win-win situation. However, since the information about customers is often considered as commercial secrets, both the bank and the insurance company will not share their raw data. They needs to perform secure skyline queries over their own data, without revealing extra information of their own dataset.

Providing efficient responses to skyline queries in these applications faces two main challenges. *(i)* To perform skyline queries without revealing the information about silos' individual data, the federation should utilize the secure multi-party computation (SMC) techniques, which can be two or three order of magnitude slower than plaintext computations [6], [8]. Thus, an efficient skyline algorithm over the data federation needs to reduce the number of secure operations and improve the efficiency of them. *(ii)* Existing secure skyline query processing methods mainly focus on horizontal partitioned data, which allows to utilize the additive property of skyline query to prune the dataset beforehand [9], [10]. However, in case of vertical partitioned data, all silos only hold a part of attributes of the entire dataset. Such pruning is inapplicable without other silos' information.

In this paper, we define the Vertical Federated Skyline (VFS) problem and investigate efficient solutions to skyline queries over a vertical data federation. We aim to find an efficient and secure solution to execute a VFS query, which holds potential for large-scale commercial recommendation applications without revealing the individual information about silos. At a high level, we optimize VFS query processing from two aspects. *(i)* We design a local dominance based framework which can decompose as plaintext local

- *Y. Zhang, Y. Shi, C. Xue, Y. Xu and K. Xu are with the State Key Laboratory of Software Development Environment and Beijing Advanced Innovation Center for Future Blockchain and Privacy Computing, Beihang University, China. E-mail: {zyy-buaa, skyxuan, xuechunbo, xuy, kexu}@buaa.edu.cn.*
- *Z. Zhou is with Singapore Management University, Singapore. E-mail: zimuzhou@smu.edu.sg.*
- *J. Du is with School of Computer Science, Beijing University of Posts and Telecommunications, China. E-mail: junpingd@bupt.edu.cn.*
- *Yexuan Shi and Yi Xu are the corresponding authors in this paper.*

TABLE 1: Summary of major notations

Notation	Description
S	a data federation
D	entire dataset of federation
A	relation schema of federation
s_i	i-th data silo of federation
D_{s_i}	local dataset of s_i
A_{s_i}	relation schema held by data silo s_i
p	a sample in federation S
$p[A_j]$	value of the j-th attribute of sample p

dominance computations and secure result aggregations. This framework can reduce the number of secure operations without compromising security. (ii) We propose a specialized private set intersection (PSI) emptiness protocol to accelerate our local dominance based VFS query framework.

Our main contributions and results are as follows.

> We identify a novel skyline query processing problem, the Vertical Federated Skyline (VFS) problem, and devise a local dominance based framework that can perform as many computations in plaintext as possible without compromising security, which leads to a higher query efficiency.
>
> By reformulate the secure operations in our framework as private set operations, we propose a new private set intersection based algorithm to further improve the query efficiency, which are implemented with specialized high-efficiency secure primitives rather than the general-purposed SMC libraries.
>
> Experiments on both synthetic and real-world datasets show that compared with generic SMC based solutions, our PSI based algorithm reduces the time cost of a single VFS query by up to 35.4 and communication cost by two orders of magnitude.

In the rest of this paper, we formulate the VFS problem in Sec. 2, present our local dominance based framework in Sec. 3, propose a baseline solution by generic SMC techniques in Sec. 4, and devise an accelerated VFS solution by PSI in Sec. 5. We present the evaluations in Sec. 6, review related work in Sec. 7, and finally conclude in Sec. 8.

2 Problem Statement

This section introduces the relevant concepts and the formal definition of the Vertical Federated Skyline (VFS) query. Tab. 1 summarizes the major notations.

Definition 1 (Silo). A silo, denoted by s, is an autonomous institution holding its own dataset D_s. Each row of the dataset corresponds to a sample, which is a d_s-dimension vector. That is, silo s holds d_s attributes for each sample.

The attribute set of silo s can be regarded as the relation schema of dataset D_s, denoted by $A_s = \{A_1, \ldots, A_{d_s}\}$. Each sample p possessed by silo s is a relation instance with schema A_s.

Definition 2 (Vertical Data Federation). A vertical data federation consists of m silos, i.e., $S = \{s_1, \ldots, s_m\}$. Each silo s_i has its own dataset about the same n samples on d_{s_i} attributes, where the corresponding attribute set A_{s_i} differs across silos, i.e., the attributes held by m silos are non-overlapping.

Let D be the dataset of the vertical data federation S, which contains n samples on $d = \sum_{i=1}^{m} d_{s_i}$ attributes. The schema of D is denoted by $A = [\bigcup_{i=1}^{m} A_{s_i}$. We consider applications such as commercial recommendation [4] on a vertical data federation. For convenience, we assume a coordinator to receive a query, distribute its execution, and return the query result to the requester. In practice, the coordinator can be either one of the silos or a dedicated entity.

Definition 3 (Dominance). In dataset D with schema A, the dominance relationship between two samples p_a and p_b is denoted by $\prec_A$. p_a dominates p_b (denoted by $p_a \prec_A p_b$), if (1) for all $A_j \in A$, $p_a[A_j] \preceq p_b[A_j]$, and (2) for at least one attribute $A_j \in A$, $p_a[A_j] < p_b[A_j]$, where $p_a[A_j]$ is the value of the j-th attribute of sample p_a $(1 \le j \le d)$. Formally,

$$p_a \prec_A p_b, \bigwedge_{A_j \in A} p_a[A_j] \preceq p_b[A_j] \wedge \bigvee_{A_j \in A} p_a[A_j] < p_b[A_j] \quad (1)$$

Such dominance relationship is useful for instance pruning in applications such as recommendation systems. If p_a dominates p_b, it means p_a outperforms p_b in all the attributes. Therefore, we can safely omit p_b and only consider p_a for subsequent processing e.g, recommendation.

Definition 4 (Semi-honest Adversary Model). The semi-honest (a.k.a., honest-but-curious) adversary model assume each silo will honestly execute the required queries, but may attempt to infer other silos' data during query execution. This adversary model allows up to $m-1$ silos to collaboratively infer the data of the remaining silo.

The semi-honest adversary model is widely used in research on data federation [5], [6], [11]. In our context, a silo s_i will not share its own dataset D_{s_i} with other silos or the coordinator. It may attempt to infer the dataset of other silos, but will honestly execute the operations assigned by the coordinator.

We now formally define the Vertical Federated Skyline (VFS) query as follows.

Definition 5 (VFS Query). Given a data federation S with dataset D and schema A, the vertical federated skyline query aims to securely find the IDs of all samples $p_b \in D$ that are not dominated by any other samples in D, i.e.,

$$VFS(D) = \{p_b.id \mid p_b \in D, \nexists p_a \in D \text{ s.t. } p_a \prec_A p_b\}; \quad (2)$$

against the semi-honest adversary model. That is, each silo s_i can only know the sample IDs of the query results without any other information on the attributes of other silos.

VFS queries find samples that outperform others among all attributes, which facilitate silos to provide more accurate and effective recommendation without leaking information about their own data.

Example 1. Tab. 2 shows a vertical data federation holding 4 samples across 3 silos. Each silo holds one attribute of the samples, denoted by A_1, A_2, and A_3, respectively. A VFS query asks the sample IDs that cannot be dominated by any other samples. In this example, sample $p_1 = (6, 3, 8)$ is dominated by $p_0 = (4, 3, 6)$, while both sample $p_0 = (4, 3, 6)$ and $p_2 = (2, 7, 7)$ dominate $p_3 = (7, 8, 7)$. Thus, sample ID #0 and #2 are the result of the VFS query on this federation. Each silo can only get the sample

TABLE 2: An example of VFS query and result (the result IDs are marked in bold).

Silo #1		Silo #2		Silo #3	
ID	A_1	ID	A_2	ID	A_3
0	4	0	3	0	6
1	6	1	3	1	8
2	2	2	7	2	7
3	7	3	8	3	7

Fig. 1: Overview of local dominance based execution framework for VFS queries.

IDs of the query result, without knowing any other information about these samples from other silos, e.g., the attribute values of the returned sample IDs in other silos.

In this work, we aim at efficient and secure execution of VFS queries, which enables large-scale user recommendation with security guarantees against the semi-honest adversary model.

3 Local Dominance based Framework

This section introduces an efficient and secure VFS query execution framework. The principle is to decompose a VFS query into plaintext operations within silos and secure operations across silos. Prior studies [6], [8] have shown that plaintext computation can be two or three orders of magnitude faster than secure multiparty computation (SMC). Therefore, our objective is to decompose as many plaintext operations as possible for high query efficiency without compromising security.

Specifically, our idea is to decompose secure computation of dominance across silos, known as global dominance, into plaintext local dominance checking within each silo and then secure aggregation across silos, as shown in Fig. 1. The key insight is to decompose the global dominance defined on the entire schema $A = [_{i=1}^{m} A_{s_i}$, into the aggregation of local dominance defined on individual schemas A_{s_i}. Such decomposition involves two types of local dominance, as defined below.

> Strict Local Dominance. It is defined the same as Def. 3 expect that the scope is on schema A_{s_i} of silo s_i rather than the entire schema A of federation D. Specifically, sample p_a strictly dominates sample p_b in silo s_i iff p_a dominates p_b on schema A_{s_i}, which is denoted by:
>
> $$p_a \succ_{A_{s_i}} p_b \tag{3}$$
>
> Relaxed Local Dominance. It satisfies condition (1) in Def. 3 and is defined on each schema A_{s_i} of silo s_i. Specifically, the relaxed local dominance in silo s_i, denoted as $\succeq_{A_{s_i}}$, is defined as follows.
>
> $$p_a \succeq_{A_{s_i}} p_b, \bigwedge_{A_j \in A_i} p_a[A_j] \geq p_b[A_j] \tag{4}$$

Given the two types of local dominance defined above, the global dominance over the entire schema $A = [_{i=1}^{m} A_{s_i}$ can be reformulated as:

$$p_a \succ_A p_b, \bigwedge_{s_j \in S} p_a \succeq_{A_{s_j}} p_b \wedge \overline{\bigwedge_{s_i \in S} p_a \succeq_{A_{s_i}} p_b} \tag{5}$$

Remarks. We make two notes on our local dominance based framework for VFS query execution.

> The decomposition in Eq. (5) allows plaintext computation i.e., Eq. (3)-(4), which results in a higher query efficiency than the original definition of the VFS query, which can only be executed with slow SMC operations.
>
> The local dominance relationships only involves the attributes held by the same silo. Hence plaintext computation of local dominance does not leak data across silos, as long as these local dominance relationships are securely aggregated to generate the final results.

As next, we first introduce a baseline VFS algorithm that directly implements the framework with generic SMC techniques (Sec. 4), and then propose an accelerated algorithm leveraging dedicated SMC protocols (Sec. 5).

4 Baseline Algorithm with Generic SMC

This section presents a VFS query algorithm that directly implements the local dominance based framework i.e., Eq. (5), with general-purposed secure multiparty computation. We first explain the secure aggregation protocol with existing generic SMC operations, i.e., secret sharing (SS) in our case (Sec. 4.1), and then introduce the end-to-end VFS query execution algorithm (Sec. 4.2).

4.1 SS-based Secure Aggregation Protocol

Secret sharing [12] is a general-purposed SMC technique to distribute a secret among multiple parties while the secrete can only be reconstructed by aggregation among all parties. Assume m silos and a number v held by silo s_i is a secret to be sent to the coordinator. Silo s_i divides v into m shares, subject to $\sum_{j=1}^{m} v_j = v$. Since silo s_i will not disclose its own share to other silos, the remaining $m - 1$ silos cannot reconstruct v even if they collaborate to infer the secret.

We exploit secret sharing to enable secure aggregation at the coordinator without each silo cannot infer the operand values of other silos. Take the secure addition of two secrets u and v as an example. The coordinator first requests the two silos who hold u and v to distribute the secrets among the silos. That is, silo s_j will receive two shares u_j and v_j. The coordinator will then ask each silo to perform addition locally, i.e., $u_j + v_j$, and returns the results for aggregation. In our context, the coordinator can be one of the silos. Hence the silos can be aware of the result i.e., $u + v$. However, since

the silo holding u or v does not disclose its share of u or v, the remaining $m-1$ silos cannot infer the value of u or v.

Such secure computation between two secrets u and v involves the local computation of shares u_j, v_j at silo s_j and the communications between any two silos s_i, s_j with their shares u_i, v_j. Thus, a single secure computation takes $O(m^2)$ time and communication cost [13].

4.2 Baseline VFS Query Execution Algorithm

We now explain how to execute a VFS query by implementing the local dominance framework with secret sharing.

Alg. 1 illustrates the overall procedure. The VFS query result are initialized as an empty set in line 1. For each sample p_b (line 2), each silo first computes the strict and relaxed local dominance defined in Eq. (3) and Eq. (4) in plaintext. Specifically, $\text{st-dom}_{s_i}(a,b)$ indicates whether sample p_a strict dominates p_b in silo s_i (line 4), and $\text{re-dom}_{s_i}(a,b)$ indicates whether sample p_a relaxed dominates p_b in silo s_i (line 5). Then the coordinator will ask the m silos to share their local dominance results via secret sharing, i.e., make shares for $\text{st-dom}_{s_i}(a,b)$ and $\text{re-dom}_{s_i}(a,b)$ $(1 \le a \le n)$ and distribute $m-1$ shares to the other $m-1$ silos (lines 6-7). Afterwards, the coordinator will ask the m silos to securely computes the global dominance $\text{dom}(a,b)$ according to Eq. (5) (lines 8-9). Finally, sample p_b is appended to the skyline result if there are no sample p_a that can dominate p_b (lines 10-11). Notice that only the final condition $\bigvee_{a=1}^{n} \text{dom}(a,b)$ can be revealed to the coordinator, all intermediate results $\text{dom}(a,b)$ should be computed and stored in secret.

Algorithm 1: Baseline algorithm with generic SMC

```
Input: vertical data federation S, the entire dataset
       D and its schema A
Output: the skyline set of D
1  res ← ∅;
2  for p_b ∈ D do
       /* plaintext within silo                          */
3      for s_i ∈ S do
4          st-dom_{s_i}(·, b) ← strict local dominance of p_b
             according to Eq. (3)
5          re-dom_{s_i}(·, b) ← relaxed local dominance of
             p_b according to Eq. (4)
       /* secure cross silos                             */
6      for s_i ∈ S do
7          share st-dom_{s_i}(·, b) and re-dom_{s_i}(·, b) to all
             silos in secret
8      for a ← 1 to n do
9          securely compute dom(a, b) according to
             Eq. (5)
10     if ⋁_{a=1}^{n} dom(a, b) is false then
11         res ← res ∪ {p_b}
12 return res
```

Example 2. Back to Example 1, in the local dominance based framework, to check whether p_b is a skyline sample, each silo will first compute the local dominance relationships of p_b. For example, when processing p_1, the strict local dominance in s_1 is $\text{st-dom}_{s_1}(\cdot,1) = (1,0,1,0)$, which means sample p_0 and p_2 are in strict local dominance to p_1 in s_1. The relaxed local dominance $\text{st-dom}_{s_1}(\cdot,1) = (1,1,1,0)$, i.e., except the strict dominating sample p_1 is in relaxed dominance to itself. Similarly, the local dominance relationships of s_2 are $\text{st-dom}_{s_2}(\cdot,1) = (0,0,0,0); \text{st-dom}_{s_2}(\cdot,1) = (1,1,0,0)$. For silo s_3, $\text{st-dom}_{s_3}(\cdot,1) = (1,0,1,1); \text{st-dom}_{s_2}(\cdot,1) = (1,1,1,1)$. With these local dominance relationships, we can apply secure sharing to securely compute each $\text{dom}(a,b)$ and finally get the result of $\bigvee_{a=1}^{n} \text{dom}(a,b) = 1$, which means p_b is dominated by another samples in the federation. The whole skyline result can be securely computed by this way.

Complexity Analysis. We only account for the time complexity and communication cost of secure computations in Alg. 1, because the plaintext computations (lines 3-5) are more efficient and do not involve any communication cost. Specifically, for each sample p_b, as shown in Eq. (5), the computation of $\text{dom}(a,b)$ needs $O(m)$ secure computations (line 9), which takes $O(m^3)$ time and communication cost. Similarly, both the time and communication cost of secure computations in line 8 are also $O(m^3)$. Also, the time complexity of line 9 in Alg. 1 is $O(m^3)$. Thus, both the total time complexity and communication cost of Alg. 1 are $O(n^2 m^3)$.

5 Accelerated Algorithm with PSI

This section presents an accelerated VFS query algorithm via a novel reformulation of the local dominance based framework. The key idea is to convert the secure aggregation of local dominance from sample-level to set-level, by designing a multi-party private set intersection (PSI) based secure aggregation protocol. We first explain how to reformulate the secure aggregation (Sec. 5.1), introduce a PSI-based aggregation protocol (Sec. 5.2), and finally describe the overall accelerated algorithm (Sec. 5.3).

5.1 Reformulation of Secure Aggregation

The ultimate objective of a VFS query is to check for each sample p_b whether there is another sample p_a can dominate p_b over the entire schema A, i.e., Eq. (1). The local dominance based framework in Sec. 3 decomposes the process into plaintext local check at each silo with schema A_{s_i} followed by secure aggregation of the local dominance results, i.e., Eq. (5). The baseline algorithm in Alg. 1 naively aggregates the local results on a per sample basis, i.e., securely aggregate all values of $\text{dom}(a,b)$ for a from 1 to n (lines 8-9). We argue that an efficient alternative is to securely compute the global dominance at set-level, i.e., by securely checking whether the intersection of local dominating sets is empty.

Specifically, we aggregate the per-sample local dominance relationships into two local dominating sets: strict dominating set and relaxed dominating set, which are defined as follows.

$$\text{Strict}_{s_i}(b) = \{a \mid a \in \{1,\dots,n\}, p_a \prec_{A_{s_i}} p_b\} \tag{6}$$

$$\text{Relaxed}_{s_i}(b) = \{a \mid a \in \{1,\dots,n\}, p_a \preceq_{A_{s_i}} p_b\} \tag{7}$$

Recall from Eq. (5) that a sample p_a dominating p_b requires relaxed dominance relationships for all silos $s_j \in S$, and at least one strict dominance in certain silo $s_i \in S$. Note

that $p_a \succ_{A_{s_i}} p_b$ implies $p_a \succ_{A_{s_i}} p_b$. Hence a sample set that (1) dominates p_b and (2) strictly dominates p_b at silo s_i can be represented by a set intersection among strict dominating sets in silo s_i and relaxed dominating sets in other silos s_j ($j \neq i$). Thus, we can reformulate line 9 Alg. 1 as:

$$Dom_{s_i}(b) = \bigcap_{j \neq i} Relaxed_{s_j}(b) \cap Strict_{s_i}(b) \quad (8)$$

Afterwards, the total sample set that dominates p_b (line 10 in Alg. 1) can be computed by the union of $Dom_{s_i}(b)$:

$$Dom(b) = \bigcup_{i=1}^{m} Dom_{s_i}(b) \quad (9)$$

This way, we transform the n per-sample secure operations (lines 8-11 in Alg. 1) into check whether all the set intersections $Dom_{s_i}(b)$ are empty, which can be implemented more efficiently with a multi-party PSI emptiness protocol, as presented next.

5.2 PSI-based Secure Aggregation Protocol

In this subsection, we first introduce two basic secure primitives in Sec. 5.2.1, based on which we design a dedicated protocol for VFS query: multi-party PSI emptiness (Sec. 5.2.2). Finally, Sec. 5.3 illustrates how to use the proposed protocol to execute a VFS query.

5.2.1 Secure Primitives

Our dedicated protocol use two secure primitives as building blocks, ElGamal Encryption and Garbled Bloom Filter. The former is for secure computations. The latter achieves secure storage and communication.

ElGamal Encryption. ElGamal is a partial homomorphic encryption scheme [14], which allows computations on encrypted data. It is defined over a cyclic group G_p of order p with a generator g, where p is often a large prime number and g is a primitive element of the group. It consists of three components: key generation $KGen(p; g)$, encryption $Enc(t; h)$ and decryption $Dec((c1; c2); x)$.

> $KGen(p; g)$: Generate a public key h and a private key x. The private key x is a random integer uniformly drawn from $Z_p = \{1; \ldots; p-1\}$, and the public key is $h = g^x$.
> $Enc(t; h)$: It encrypts a secret value t by public key h. With a random integer r uniformly drawn from Z_p, the ciphertext of t is represented as $E(t) = (c1; c2)$, where $c_1 = g^r$ and $c_2 = t \cdot h^r$.
> $Dec((c1; c2); x)$: Given a ciphertext tuple $(c1; c2)$, the corresponding secret value t can be decrypted by private key x, i.e., $t = c2 \cdot (c1^x)^{-1}$.

The ElGamal encryption is multiplicative homomorphic, i.e., $E(t_1 \cdot t_2) = E(t_1) \cdot E(t_2)$. We adopt a threshold variant [15] of ElGamal for encryption and decryption cross m silos. It allows the private key to be separated into m part $x = \sum_{i=1}^{m} x_i$ and each silo s_i holds x_i. The ciphertext $(c1; c2)$ can then be decrypted by $Dec((c1; c2); x_1; \ldots; x_m) = c2 \cdot (c1^{x_1} \cdots c1^{x_m})^{-1}$, where each silo s_i only provides $c1^{x_i}$.

Garbled Bloom Filter. The garbled bloom filter is a probabilistic data structure that can implement secure key-value store and communications cross silos [16]. With the garbled bloom filter, other silos can only extract values from specific keys, but cannot learn the values of other keys held by the garbled bloom filter. It is defined by an array $GBF[1::N]$ and a collection of hash functions $H_1; \ldots; H_k$. For a given key u, the value associated with u can be calculated by:

$$v = \bigoplus_{j=1}^{k} GBF[H_j(u)] \quad (10)$$

where $\bigoplus$ is the exclusive-or operator. Given a set of key-value pairs $\{(u; v)\}$, the garbled bloom filter can be built with the following steps.

> Initialize all elements in array $GBF[1::N]$ as $?$.
> For each key-value pair $(u; v)$, find the positions $H_j(u)$ in GBF that have not been set, i.e.,
>
> $$J = \{H_j(u) | GBF[H_j(u)] = ?\} \quad (11)$$
>
> The procedure aborts if $J = \emptyset$. Otherwise, choose random values for $GBF[J]$ subject to the lookup equation in Eq. (10) equalling the desired value v.
> For any remaining $GBF[j] = ?$, replace $GBF[j]$ with a randomly chosen value.

Unless this procedure aborts, it produces the desired key-value mapping. The probability of aborting is the same as the probability of false positive in plaintext bloom filter [16]. It has been proven [17] that by setting $N = nk \log_2 e$, the probability of aborting is bounded by 2^{-k}, where n is the number of key-value pairs. The silos can achieve secure communications by sending the mapping array $GBF[1::N]$.

5.2.2 Multi-party PSI Emptiness

Based on above secure primitives, we design a novel protocol that securely tests whether the multi-party set intersection is empty.

Basic Idea. Given m sets $O_1; \ldots; O_m$ distributed among m silos, we aim to check whether the intersection of these sets is empty. For each silo s_i, it can associate each sample u a value $v_{s_i}(u)$, where $v_{s_i}(u) = 1$ for samples in O_i, and $v_{s_i}(u)$ can be a random value r otherwise. Thus, the intersection contains sample u if and only if $\prod_{s_i \in S} v_{s_i}(u) = 1$. It can be securely computed by first encrypting each $v_{s_i}(u)$ with ElGamal and then multiplying them consecutively. The next question is how a silo identifies the sample ID of a ciphertext from other silos, which can be achieved by the garbled bloom filter. It can transfer the ciphertext with corresponding keys securely. Because all values stored in the garbled bloom filter are ciphertext, such transmissions will not reveal extra information about silos.

Algorithm Details. Alg. 2 illustrates our protocol to securely check the emptiness of multi-party set intersection. It contains five steps:

> **Init.** Firstly, all silos generate a public key h with the threshold variant of ElGamal, where the secret key $x = \prod_{i=1}^{m} x_i$ is divided into m parts for m silos (line 1). They share a common collection of hash functions $H_1; \ldots; H_m$ (line 2).
> **Encode.** Without loss of generality, let silo s_1 be the coordinator. For any other silo s_i ($2 \leq i \leq m$) (line 3),

Algorithm 2: Multi-party PSI Emptiness

Input: vertical data federation S, the sets $O_1, \ldots, O_m$, a prime p and a generator g
Output: the indicator that whether the intersection is empty

```
/* Step 1: Init */
1  (h, x) ← KGen(p, g), where x is divided into m parts x_1, ..., x_m for m silos
2  {H_1, ..., H_k}  a collection of hash functions
/* Step 2: Encode */
3  for data silos s_i ∈ {s_2, ..., s_m} do
4      build garbled bloom filter GBF_{s_i}[1::N] based on {(u, E(1)) | u ∈ O_i}
5      send GBF_{s_i}[1::N] to s_1
/* Step 3: Product */
6  for data silos s_1 do
7      for u ∈ O_1 do
8          v(u) ← ∏_{i=2}^{m} ∑_{j=1}^{k} GBF_{s_i}[H_j(u)]
9      send V ← {v(u)} to s_2
/* Step 4: Shuffle */
10 for data silos s_i ∈ {s_2, ..., s_m} do
11     receive V from s_{i-1}
12     compute v'(u) = v(u) ⊗ E(1)
13     V' ← random shuffle {v'(u)} and send it to s_{i+1}
/* Step 5: Decode */
14 for v'(u) ∈ V' do
15     t ← Dec(v'(u); x_1, ..., x_m)
16     if t = 1 then
17         return false
18 return true
```

it builds a garbled bloom filter $GBF_{s_i}[1::N]$ where the key-value pairs are $\{(u, E(1)) \mid u \in O_i\}$ (line 4). All silos send their own GBF_{s_i} to silo s_1 (line 5).

Product. In silo s_1 (line 6), for each sample $u \in O_1$, it computes $v(u) = \prod_{i=2}^{m} v_{s_i}(u)$, where $v_{s_i}(u)$ is obtained by Eq. (10) on the garbled bloom filter GBF_{s_i} (lines 7-8).

Shuffle. The encoded dataset $\{v(u)\}$ is shuffled by all m silos. When silo s_i receives $\{v(u)\}$ (line 11), it calculates $v'(u) = v(u) \otimes E(1)$ (line 12) and sends the randomly shuffled $\{v'(u)\}$ to the next silo (line 13).

Decode. Finally, given the shuffled $\{v'(u)\}$, all silos unite to decode them one by one (line 15). If there are some $Dec(v'(u); x) = 1$, it means the intersection is not empty (lines 16-17). Otherwise, the intersection is empty (line 18).

$GBF_{s_2}[H_1(0)]$ as $(1; 2)$, and $GBF_{s_2}[H_2(0)]$ as $(8; 7)$, subject to $(1; 2) \otimes (8; 7) = (9; 5)$. The other values in GBF_{s_2} are set randomly. GBF_{s_3} can be obtained similarly and both these two arrays are sent to s_1. In the product step, s_1 receives GBF_{s_2} and GBF_{s_3}. It computes $v(u) = \prod_{i=2}^{3} \otimes_{j=1}^{2} GBF_{s_i}[H_j(u)]$ for $u_1 = 0$ and $u_2 = 2$. The results are $v(u_1) = (3; 4)$ and $v(u_2) = (2; 8)$. Then, it sends these two ciphertexts to s_2 and s_3 for the shuffle step (Fig. 2d) and gets the shuffled ciphertext $(9; 3)$ and $(8; 7)$. Finally, all silos united to decrypt these two ciphertexts, which yields $Dec((8; 7); 2; 3; 4) = 1$. It means the set intersection is not empty.

Complexity Analysis. The init step (lines 1-2) can be done in $O(1)$. In step 2, each silo builds a garbled bloom filter $GBF_{s_i}[1::N]$ locally, which takes $O(n)$ time (line 4). In step 3, for each sample u in silo s_1, the ciphertext $v(u)$ is calculated in $O(m)$ time (line 8). Thus, the time complexity of step 3 is $O(nm)$. In step 4, the ciphertext set $\{v(u)\}$ is shuffled by all silos, which needs $O(nm)$ time in total. Finally, in the decode step, each ciphertext $v'(u)$ needs $O(m)$ time to decrypt. Thus, this step also takes $O(nm)$ time. Hence, the total time complexity of Alg. 2 is $O(nm)$. As for the communication cost, only step 2 (lines 3-5) and step 4 (lines 10-13) incur communication among all silos, which leads to $O(nm)$ communication cost.

5.3 Accelerated VFS Query Execution Algorithm

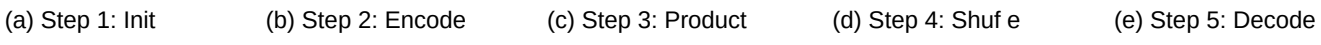
With the multi-party PSI protocol, we now explain how to execute a VFS query efficiently.

Intuitively, whether Eq. (9) is empty can be checked by independently running m multi-party PSI emptiness of $Dom_{s_i}(b)$ i.e., Eq. (8). However, this method may impose security risks. Specifically, silo s_1 will know whether attributes in silo s_j can strictly dominate p_b in the computation of $Dom_{s_i}(b)$. Hence, we need an alternative to get the final answer without revealing the internal results of set intersections. Our solution is to merge the first three steps of Alg. 2 for all $Dom_{s_i}(b)$, i.e., compute the whole ciphertext set of all $Dom_{s_i}(b)$, and then randomly shuffle these ciphertexts and decode them unitedly (step 4 $\rightarrow$ step 5 in Alg. 2).

Alg. 3 illustrates our PSI based VFS algorithm. In line 1, it first initializes the skyline result as an empty set. For each sample p_b, each silo first computes two local dominating sets in lines 4-5. Then, they unite to execute the multi-party PSI emptiness test. Specifically, they first execute the init and encode step of Alg. 2, to get the garbled bloom filters for two local dominating sets (lines 6-7). Then, silo s_1 executes the product step for each $Dom_{s_i}(b)$ and obtains the whole ciphertext set V (lines 8-10). The silos in the federation then shuffle V as V' and finally decrypt elements in V' to check whether p_b is a skyline sample (lines 11-13).

Example 3. Fig. 2 shows a example of our multi-party PSI emptiness protocol. Back to Example 2, assume we want to check whether the intersection of sets $Relaxed_{s_1}(1)$; $Relaxed_{s_2}(1)$ and $Strict_{s_3}(1)$ is empty. In the init step, all silos agree on $p = 11$; $g = 8$; $h = 7$ and two hash functions H_1, H_2. Each silo s_i holds a private key x_i, as shown in Fig. 2a. Next, s_2 and s_3 build their garbled bloom filters. For example, the set of s_2 only has one element 0. Thus, it generates a ciphertext $E(1) = (g^2; 1 \otimes h^2) = (9; 5)$ for this element. Ant it sets

Example 4. Back to Example 2, when checking whether p_0 is a skyline sample, each silo first builds two garbled bloom filters for strict and relaxed dominating sets, as shown in Tab. 3. The building of garbled bloom filter is similar to Example 3. Next, s_1 executes the product step, as shown in Fig. 3. Tab. 3 shows the execution of the 3rd intersection. All silos shuffle the ciphertext and decrypt them one by one. The decrypted value $(4; 3)$ is 1, which means the set intersection is not empty. Then, the procedure returns false immediately, which means p_0 is not a skyline sample.



(a) Step 1: Init (b) Step 2: Encode (c) Step 3: Product (d) Step 4: Shuf e (e) Step 5: Decode

Fig. 2: Example of multi-party PSI emptiness.

TABLE 3: Example of garbled bloom lters in s_2 and s_3.

	GBF_{s_2}			GBF_{s_3}	
pos	Relaxed	Strict	pos	Relaxed	Strict
0	(1,2)	(3,6)	0	(10,2)	(2,7)
1	(8,7)	(2,5)	1	(2,5)	(6,4)
2	(3,9)	(5,5)	2	(5,7)	(5,2)
3	(9,2)	(4,6)	3	(6,6)	(4,2)
4	(7,6)	(1,8)	4	(3,5)	(2,10)

Fig. 3: Example of PSI based VFS algorithm.

Algorithm 3: PSI based VFS algorithm

```
   Input: vertical data federation S, the entire dataset
          D and its schema A
   Output: the skyline set of D
1  res ;
2  for p_b 2 D do
       /* plaintext within silo                          */
3      for s_i 2 S do
4          Strict_{s_i}(b)  strict local dominating set of p_b
             according to Eq. (6)
5          Relaxed_{s_i}(b)  relaxed local dominating set
             of p_b according to Eq. (7)
       /* secure cross silos                             */
6      all silos initialize the public key h and private
         key x_1;  ; x_m
7      each silo s_i builds garbled bloom lter for its
         strict and relaxed dominating set respectively
         and sends them to silo s_1
8      V ;
9      for i  1 to m do
10         data silo s_1 compute V  V [ the ciphertext
             set for Dom_{s_i}(b)
11     V'  random shuf ed V by all data silos
12     if there is no element in V' decrepyted as 1 then
13         res  res [ f bg
14 return res
```

There is no need to decrypt the remaining ciphertext. Finally, Alg. 3 can securely get the whole skyline set.

Complexity Analysis. We focus on the complexity of secure computations in Alg. 3. Lines 6-7 correspond to step 1 and step 2 of Alg. 2, which take $O(n)$ time and communication cost. Lines 8-10 execute $O(m)$ product step of Alg. 2, which takes $O(nm^2)$ time. Similarly, both the time complexity and the communication cost of line 11 are $O(nm^2)$ because it needs to transfer $O(nm)$ ciphertext across all silos. Lines 12-13 decrypt all ciphertext in V', each of which takes $O(m)$ time. Thus, their time cost is also $O(nm^2)$. To summarize, both the total time complexity and communication cost of Alg. 3 are $O(n^2m^2)$.

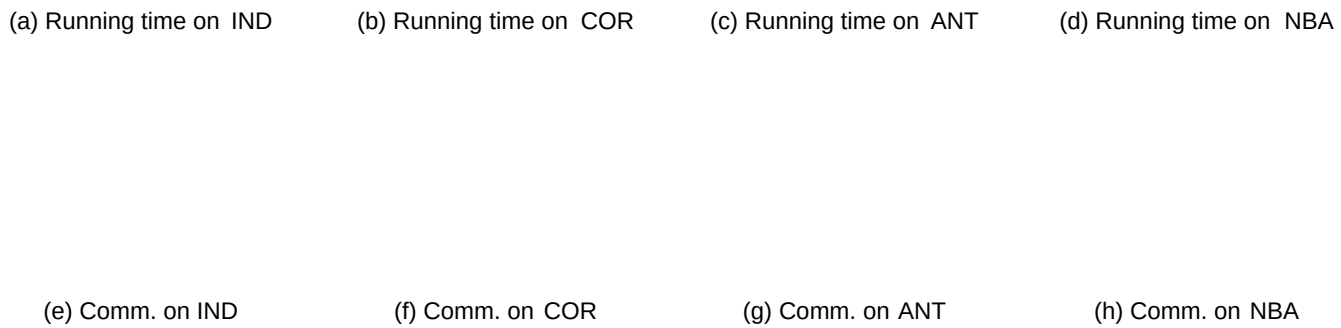
Security Analysis. The security of our PSI based accelerated algorithm depends on the underlying secure primitives. Assume a subset I S bands together to infer the other silos' information. Let the simulator SIM have access to the colluded silos' input sets $f D_{s_i} g_{s_i 2 I}$. As Alg. 3 executes, the simulator SIM can only get the ciphertexts about other silos $f CP_{s_i} g_{s_i 2 I}$ (garbled bloom lters from other silos). Since the encryption scheme ElGamal is semantically secure, the ciphertexts $f CP_{s_i} g_{s_i 2 I}$ and their raw data $f D_{s_i} g_{s_i 2 I}$ are indistinguishable. Thus, the simulated view of SIM is indistinguishable from the real world algorithm output. The silos in I cannot learn any information about other silos.

6 Experiment Study

This section presents the evaluations of our methods.

6.1 Experimental Setup

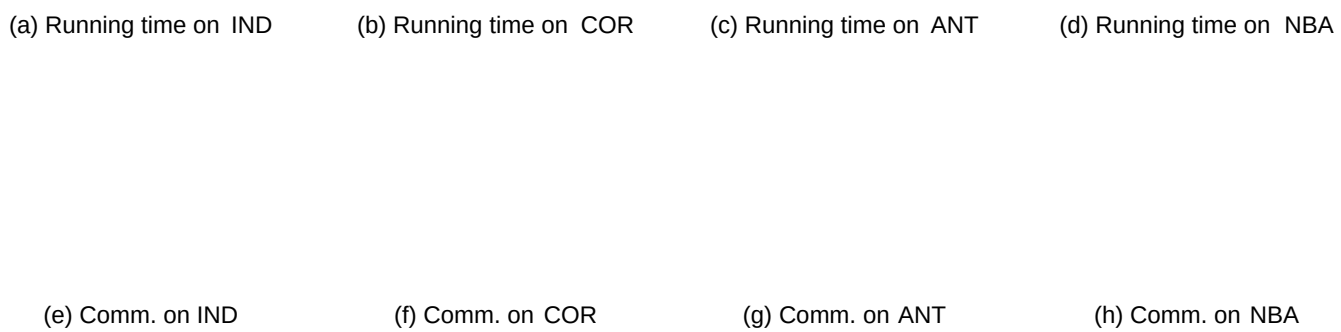
Datasets. Following previous research [1], [9], [10], we experiment with three synthetic datasets with different distri-



(a) Running time on IND (b) Running time on COR (c) Running time on ANT (d) Running time on NBA

(e) Comm. on IND (f) Comm. on COR (g) Comm. on ANT (h) Comm. on NBA

Fig. 4: Performance of varying the silo number.



(a) Running time on IND (b) Running time on COR (c) Running time on ANT (d) Running time on NBA

(e) Comm. on IND (f) Comm. on COR (g) Comm. on ANT (h) Comm. on NBA

Fig. 5: Performance of varying the data size.

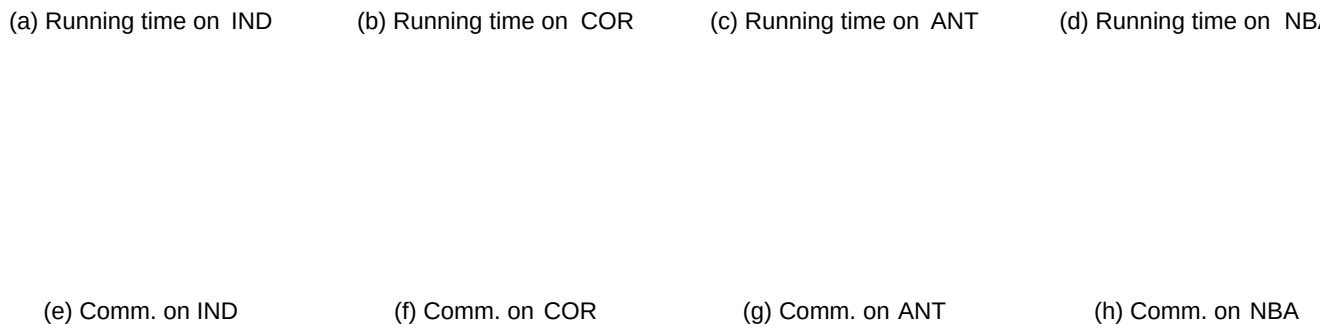
butions and a real-world dataset about NBA players.

Synthetic Datasets We generate sample sets with independent (IND), correlated (COR) and anti-correlated (ANT) distribution. Speci cally, the IND dataset generates all values independently with a uniform distribution. The COR dataset is generated by rst selecting a plane perpendicular to the line from (0; ; 0) to (1; ; 1) using a normal distribution (= 0:1) and then generating a point on such line with another normal distribution (= 0:05). In the ANT dataset, each sample is generated by a uniform distribution on the same plane perpendicular selected by a normal distribution with small variance (= 0:05).

Real-world Dataset To evaluate our skyline algorithms in real-world applications, we collect the statistics of 4000 NBA players from 2002 to 2022 [18], which we call as the the NBA dataset. Each player has 20 attributes, including game played (GP), points (PTS), eld-goal percentage (FG), rebound (REB), steals (STL),etc.

To simulate real-world skyline query applications, we vary the size of data federation n (i.e., the number of samples) from 100 to 4000. To vary the number of silos (m), we equally split the samples to form multiple silos as in previous studies [5], [6], [19]. By default, each silo holds one attribute. We vary the number of attributes held by one silo (dimensions of a silo) from 1 to 5 to further evaluate the ef ciency of proposed algorithms. The parameter setting are listed in Tab. 4. The default values are marked in bold.

Compared Algorithms. We compare the performance of the following algorithms.



(a) Running time on IND (b) Running time on COR (c) Running time on ANT (d) Running time on NBA

(e) Comm. on IND (f) Comm. on COR (g) Comm. on ANT (h) Comm. on NBA

Fig. 6: Performance of varying the number of dimensions in each data silo.

TABLE 4: Parameter settings (defaults are marked in bold)

Parameter	Setting
dataset	IND, COR, ANT, NBA
silo number m	2; 4; 6; 8; 10
data size n	100; 500; 1000; 2000; 4000
dimension of each data silo	1; 2; 3; 4; 5

MP-SPDZ. The baseline VFS query execution algorithm implemented by a state-of-the-art general-purpose SMC library MP-SPDZ [13] (Alg. 1).
MPSIC. Our accelerated VFS query execution algorithm implemented with a state-of-the-art PSI cardinality protocol [20].
MPSIE. Our accelerated VFS query execution algorithm implemented with our specialized multi-party PSI emptiness protocol (Alg. 3).

Implementation. The experiments are conducted on a cluster of up to 10 servers, each of which has four 3.10GHz Intel(R) Xeon(R) Platinum 8269CY processors and 32GB memory with Ubuntu 18.04 LTS. The network bandwidth between machines is up to 6 GB/s. All of the algorithms are implemented in GNU C++ with the GMP library [21] for large integer computation. The security parameters are fixed as follows: both the length of generator g in ElGamal and the length of prime p are 1024 bits. The garbled bloom filter has $k = 20$ hash functions, and the length of the garbled bloom filter is set as $N = 20n \log_2 e$.

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

Running time. It is the time cost from receiving the VFS query to returning the query result.
Communication cost. It is the total network communication among all silos for a VFS query.

All the experimental results are the average over 10 repetitions.

6.2 Experimental Results

Impact of # silos m. Fig. 4 presents the results of varying the number of silos m on the four datasets. The running time of all skyline algorithms increases with the increase of silo number (see Fig. 4a-4d). This is because a large silo number involves more dimensions of samples and needs more computations for a skyline query. MPSIE achieves the smallest running time followed by MPSIC. MP-SPDZ performs the worst. When there are more than 6 silos, it fails to get results in 48 hours. MPSIE is up to 35.4 and 4.1 faster than MP-SPDZ and MPSIC, respectively. As for the communication cost, the results of MPSIE and MPSIC increase slightly as the silo number increases, since their communication cost is linear to silo number. Compare with MP-SPDZ, the PSI based algorithms significantly reduce the communication cost. For example, in the NBA dataset (Fig. 4h), the communication cost of MPSIE is up to 1224 smaller than MP-SPDZ. The communication costs of MPSIE and MPSIC are almost the same. It is because MPSIC is implemented by plaintext bloom filters, while MPSIE uses garbled bloom filters as building blocks, which achieves more efficient key-value storage than the plaintext version at the same communication cost.

Impact of # samples n. Fig. 5 illustrates the results of varying the number of samples n on the four datasets. Again, all skyline algorithms take more time and communication cost to perform a VFS query as the number of samples increases. From Fig. 5a-5d, MP-SPDZ also performs the worst. MPSIE is still the most efficient and MPSIC is the runner-up. In terms of running time, MPSIE takes less than 21 hours to perform a skyline query over 4000 samples. But MP-SPDZ takes more than 2 days even for 2000 samples. Thus, we omit the results of MP-SPDZ when there are 4000 samples. Again, MPSIE is up to 13.7 and 2.5 faster than MP-SPDZ and MPSIC, respectively. The communication cost of all skyline algorithms increases linearly. But the communication cost of MP-SPDZ can be up to 295.8 higher than MPSIE

and MPSIC.

Impact of # dimensions d_S. Fig. 6 shows the results of varying the number of dimensions in each silo d_S. Due to our local dominance based framework, the number of dimensions only has little impact on the results. For example, in all four datasets, the running time and communication cost of MP-SPDZ are relatively stable. Among the compared algorithms, MPSIE has the smallest running time, which is up to 73.5 and 2.5 lower than MP-SPDZ and MPSIC, respectively. In terms of communication cost, MP-SPDZ needs up to 293.6 higher cost than PSI based algorithms.

Impact of data distribution. We further study the impact of different data distributions. In Fig. 4 and Fig. 5, all algorithms have the similar trends on four datasets. However, in Fig. 6, the performance of MPSIE and MPSIC differs across the four datasets. Specically, as shown in Fig. 6a and Fig. 6c, for the independent and anti-correlated data distribution, the running time of the two PSI based algorithms MPSIE and MPSIC decreases as the number of dimensions increases. On the contrary, for the correlated distribution and the real-world NBA dataset, the running time of all algorithms is relatively stable. This is because in the correlated distribution, given a $p_b \in D$, the size of the set $Strict_{S_1}(b)$ and $Relaxed_{S_1}(b)$ (defined in Alg. 3) will be stable as the dimension increases. And the NBA dataset is a nearly correlated distribution. For the independent or even anti-correlated distribution, as the number of dimensions increases, the size of the set $Strict_{S_1}(b)$ and $Relaxed_{S_1}(b)$ would become smaller, leading to a smaller size of $Dom(b)$. For example, in the anti-correlated distribution, when each silo holds one dimension of samples, the average size of $Dom(b)$ is 1901. However, when each silo holds five dimension of samples, the size of $Dom(b)$ is only 136 on average. Thus, the two accelerated algorithms implemented with PSI will take lower time cost to execute the VFS queries as the number of dimension increases.

Summary. Our experimental findings are as follows.

- The running time our accelerated VFS algorithm implemented by specialized PSI emptiness protocol is significantly more efficient than the baselines. Specifically, MPSIE can be up to 35.4 and 4.1 faster than MP-SPDZ and MPSIC, respectively.
- The communication cost of PSI based algorithms is low. For example, the communication cost of MPSIE and MPSIC is 70.1GB on average. However, the communication cost of MP-SPDZ can be over 40TB for a single VFS query. The communication cost of our PSI based VFS algorithms can reduce the communication cost by two orders of magnitude.
- The PSI based algorithms are sensitive to the size of dominating set in a VFS query. However, this is often not an issue because there are the number of samples is typically orders more than the size of the dominating set in many real-world applications.

7 Related Work

Our work is related to three categories of research: skyline query processing, querying and analysis on data federation, and multi-party private set intersection.

7.1 Skyline Query Processing

The notion of skyline operator was first introduced in [1] with a well-known solution: block nested loop (BNL). After that, a series of efficient skyline query processing techniques are presented, including building indices [22], [23], utilizing topological order [2], [24], splitting into buckets [25], etc. More details can be found in [26], [27]. In [28], the authors studied the skyline query processing on vertical partitioned data and propose an algorithm based on the Fagin's top-k ranking framework [29]. However, it may leak the priorities of tuples to all data owners.

Many variants of skyline queries have been studied, such as group skyline [30], reverse skyline [31], k-dominant skyline [32], etc. Recently, there are some studies about secure skyline querying on the untrusted cloud platforms. For example, Chen et al. [33] studied the verification of skyline query result returned by malicious cloud servers. Liu et al. [34] proposed a semantically secure method based on a partial homomorphic encryption scheme named Paillier. However, the method has low efficiency. By utilizing a new order-revealing encryption scheme, Wang et al. [35] can securely build indices on cloud platforms and answer skyline queries more efficiently. Ding et al. [9] and Liu et al. [36] further studied the secure skyline querying over multiple horizontal partitioned data. All the aforementioned methods require two non-colluding cloud platforms, and the data owners need to upload their encrypted data to cloud. However, such two cloud platforms are hard to find in practice.

7.2 Querying and Analysis on Data Federation

As a new solution for data sharing, data federation can facilitate multiple data owners to collaboratively process queries with SMC techniques [5], [6], [37]. SMCQL [5] is the first data federation system based on a general garbled circuit library ObliVM [38]. It can support secure SQL queries over a federation with two data silos. Conclave [8] extend the secure query processing to spark and can support up to three data silos with a general secret sharing library Sharemind [39]. However, because of the dependence of general SMC libraries, these systems are still significantly slower than plaintext querying methods. To improve the system efficiency, SAQE [40] studies the secure approximate query processing over data federation, and Hu-Fu [6] designs several dedicate secure operators to accelerate querying on spatial data federation. There are also some studies apply differential privacy to data federation analysis [41], [42], [43].

As an efficient dedicated SMC techniques, private set intersection (PSI) has been used in certain query and analysis on data federation. Ge et al. [11] studied the functional dependence discovery problem in data federation. It uses PSI cardinality as a building block. In [44], the authors proposed an efficient solution with linear complexity by applying a circuit based PSI protocol to classic Yannakakis join algorithm, to handle join-aggregate queries on a data federation.

Overall, querying and analysis on data federation has become a hot topic but there has been no study about secure skyline querying on data federation.

7.3 Multi-party Private Set Intersection

The multi-party PSI requires multiple parties to compute the intersection of their own datasets without revealing any additional information. The primary multi-party PSI protocols are based on oblivious polynomial evaluation (OPE) technique [45], [46]. The main idea is to represent a dataset as a polynomial with homomorhpic encryption scheme, *e.g.*, Paillier, which leads to a quadratic time complexity. Some studies [47], [48] reduce the complexity of PSI to linear under the honest majority assumption, where only less than m=2 parties can be corrupted. Hazay *et al.* [49] extends the idea of OPE [45] to a star network topology, and achieves linear complexity with the threshold variant of homomorphic encryption scheme. By introducing the oblivious programmable pseudo random function (OPPRF), Kolesnikov *et al.* [50] propose a protocol that does not rely on any homomorphic encryption scheme. It has been proved to be efficient, but it will reveal the items belonging to the intersection set. However, in our problem setting, only the emptiness of the intersection set should be revealed. In [20], the authors proposed a plaintext bloom filter based protocol to compute the cardinality of multi-party PSI, which involves large number of encryption operations.

8 Conclusion

In this paper, we formulate the VFS problem, which aims at skyline queries over a vertical data federation. To enable efficient processing of such queries, we propose a local dominance based framework. Specifically, we decompose the judgement condition of VFS query into two local dominance relationships which can be executed within each silo in plaintext. The federation then only needs execute secure aggregation over these local dominance values to get the final VFS query results. The framework allows a higher query efficiency without compromising security. Furthermore, an accelerated algorithm based on a dedicated PSI emptiness protocol is proposed to improve the query efficiency. We validate the efficiency of our solutions on both synthetic and real-world datasets. Experiments show that compared with the baseline algorithms, our PSI based VFS algorithm can reduce the time cost by up to 35.4 and communication cost by two orders of magnitude. The experimental results show that the proposed accelerated VFS algorithm implemented by specialized PSI emptiness protocol is fit for real-world applications with large numbers of samples yet a small dominating set.

Acknowledgements

We are grateful to anonymous reviewers for their constructive comments. This work is partially supported by the National Science Foundation of China (NSFC) under Grant No. 62076017, 62192784 and U1811463, and the Lee Kong Chian Fellowship awarded to Zimu Zhou by Singapore Management University.

References

[1] S. Börzsönyi, D. Kossmann, and K. Stocker, "The skyline operator," in *ICDE*, 2001, pp. 421–430.

[2] J. Chomicki, P. Godfrey, J. Gryz, and D. Liang, "Skyline with presorting," in *ICDE*, 2003, pp. 717–719.

[3] M. Sharifzadeh and C. Shahabi, "The spatial skyline queries," in *VLDB*, 2006, pp. 751–762.

[4] J. Yang, G. P. C. Fung, W. Lu, X. Zhou, H. Chen, and X. Du, "Finding superior skyline points for multidimensional recommendation applications," *World Wide Web*, vol. 15, no. 1, pp. 33–60, 2012.

[5] J. Bater, G. Elliott, C. Eggen, S. Goel, A. N. Kho, and J. Rogers, "SMCQL: secure query processing for private data networks," *Proceedings of the VLDB Endowment*, vol. 10, no. 6, pp. 673–684, 2017.

[6] Y. Tong, X. Pan, Y. Zeng, Y. Shi, C. Xue, Z. Zhou, X. Zhang, L. Chen, Y. Xu, K. Xu, and W. Lv, "Hu-fu: Efficient and secure spatial queries over data federation," *Proceedings of the VLDB Endowment*, vol. 15, no. 6, pp. 1159–1172, 2022.

[7] Y. Shi, Y. Tong, Y. Zeng, Z. Zhou, B. Ding, and L. Chen, "Efficient approximate range aggregation over large-scale spatial data federation (extended abstract)," in *ICDE*, 2022, pp. 1559–1560.

[8] N. Volgushev, M. Schwarzkopf, B. Getchell, M. Varia, A. Lapets, and A. Bestavros, "Conclave: secure multi-party computation on big data," in *EuroSys*, 2019, pp. 3:1–3:18.

[9] X. Ding, Z. Wang, P. Zhou, K. R. Choo, and H. Jin, "Efficient and privacy-preserving multi-party skyline queries over encrypted data," *IEEE Transactions on Information Forensics and Security*, vol. 16, pp. 4589–4604, 2021.

[10] J. Liu, J. Yang, L. Xiong, and J. Pei, "Secure and efficient skyline queries on encrypted data," *IEEE Transactions on Knowledge and Data Engineering*, vol. 31, no. 7, pp. 1397–1411, 2019.

[11] C. Ge, I. F. Ilyas, and F. Kerschbaum, "Secure multi-party functional dependency discovery," *Proceedings of the VLDB Endowment*, vol. 13, no. 2, pp. 184–196, 2019.

[12] A. Beimel, "Secret-sharing schemes: A survey," in *IWCC*, vol. 6639, 2011, pp. 11–46.

[13] M. Keller, "MP-SPDZ: A versatile framework for multi-party computation," in *CCS*, 2020, pp. 1575–1590.

[14] T. E. Gamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," in *CRYPTO*, vol. 196, 1984, pp. 10–18.

[15] Y. Desmedt and Y. Frankel, "Threshold cryptosystems," in *CRYPTO*, vol. 435, 1989, pp. 307–315.

[16] C. Dong, L. Chen, and Z. Wen, "When private set intersection meets big data: an efficient and scalable protocol," in *CCS*, 2013, pp. 789–800.

[17] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Communications of the ACM*, vol. 13, no. 7, pp. 422–426, 1970.

[18] "NBA Advanced Stats," *https://www.nba.com/stats/leaders/*, 2022.

[19] Y. Shi, Y. Tong, Y. Zeng, Z. Zhou, B. Ding, and L. Chen, "Efficient approximate range aggregation over large-scale spatial data federation," *IEEE Transactions on Knowledge and Data Engineering*, 2022.

[20] S. K. Debnath, P. Stanica, N. Kundu, and T. Choudhury, "Secure and efficient multiparty private set intersection cardinality," *Advances in Mathematics of Communications*, vol. 15, no. 2, pp. 365–386, 2021.

[21] "The GNU Multiple Precision Arithmetic Library," *https://gmplib.org/*, 2022.

[22] D. Papadias, Y. Tao, G. Fu, and B. Seeger, "An optimal and progressive algorithm for skyline queries," in *SIGMOD*, 2003, pp. 467–478.

[23] K. C. K. Lee, B. Zheng, H. Li, and W. Lee, "Approaching the skyline in Z order," in *VLDB*, 2007, pp. 279–290.

[24] I. Bartolini, P. Ciaccia, and M. Patella, "Efficient sort-based skyline evaluation," *ACM Transactions on Database Systems*, vol. 33, no. 4, pp. 31:1–31:49, 2008.

[25] F. N. Afrati, P. Koutris, D. Suciu, and J. D. Ullman, "Parallel skyline queries," in *ICDT*, 2012, pp. 274–284.

[26] K. Hose and A. Vlachou, "A survey of skyline processing in highly distributed environments," *VLDB Journal*, vol. 21, no. 3, pp. 359–384, 2012.

[27] J. Chomicki, P. Ciaccia, and N. Meneghetti, "Skyline queries, front and back," in *SIGMOD*, vol. 42, no. 3, 2013, pp. 6–18.

[28] W. Balke, U. Güntzer, and J. X. Zheng, "Efficient distributed skylining for web information systems," in *EDBT*, vol. 2992, 2004, pp. 256–273.

[29] R. Fagin, "Combining fuzzy information from multiple systems," *Journal of Computer and System Sciences*, vol. 58, no. 1, pp. 83–99, 1999.