# FedVSE: A Privacy-Preserving and Efficient Vector Search Engine for Federated Databases

Zeheng Fan
SKLCCSE Lab,
Beihang University
fanzh@buaa.edu.cn

Yuxiang Zeng
SKLCCSE Lab,
Beihang University
yxzeng@buaa.edu.cn

Zhuanglin Zheng
SKLCCSE Lab,
Beihang University
zzlin@buaa.edu.cn

Yongxin Tong
SKLCCSE Lab,
Beihang University
yxtong@buaa.edu.cn

## ABSTRACT

Efficient vector search is a foundational capability of vector databases. However, most prior research overlooks its critical role in federated databases for applications like financial risk control and smart healthcare. In these privacy-sensitive scenarios, a vector search engine must not only deliver high performance but also guarantee privacy across federated databases. Current solutions, however, struggle with scalability for high-dimensional vectors, and offer limited query support. To bridge this gap, this paper introduces FedVSE, a privacy-preserving vector search engine for federated databases. FedVSE supports both KNN and hybrid queries, matching the versatility of modern vector databases. It leverages Intel SGX for hardware-enabled security and offers highly optimized query processing via indexing and pruning. Conference audiences can interact with FedVSE in real time and observe how it enables real-world services like cross-platform trajectory similarity search.

**PVLDB Reference Format:**
Zeheng Fan, Yuxiang Zeng, Zhuanglin Zheng and Yongxin Tong. FedVSE: A Privacy-Preserving and Efficient Vector Search Engine for Federated Databases. PVLDB, 18(12): 5371 - 5374, 2025. doi:10.14778/3750601.3750674

## 1 INTRODUCTION

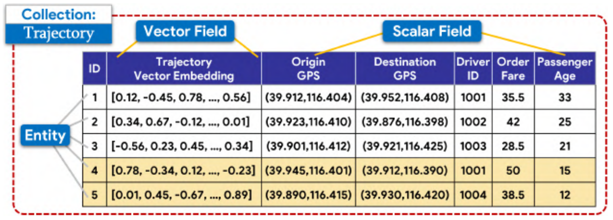
Vector databases have gained growing attention in industry and academia for their exceptional capabilities of managing data with high dimension. A core functionality of these systems is **vector search**, which enables fast retrieval of similar objects to a given query vector from large-scale datasets. Most research on vector search focuses on *single-source* vector data, overlooking the growing demand for *multi-source* vector data in applications like financial risk control, smart healthcare, and intelligent transportation.

In these privacy-sensitive scenarios, an effective vector search engine over federated databases ("**federated vector search**" as short) must provide both high query performance and privacy preservation across all databases. Prior work has explored federated vector search or federated KNN search, primarily relying on Homomorphic Encryption (HE) [13], Secure Multi-party Computation (SMC) [8, 11, 14], or Trusted Execution Environment (TEE) [12].

**Motivation.** However, these solutions suffer from two limitations:

(i) **Inefficiency for High-dimensional Vector**. HE based methods incur unavoidable latency during both data pre-processing (encryption) and query processing (decryption). SMC and TEE based methods, initially designed for low-dimensional data, become inefficient or inaccurate when applied to high-dimensional vectors.

Proceedings of the VLDB Endowment, Vol. 18, No. 12 ISSN 2150-8097.
doi:10.14778/3750601.3750674



| ID | Trajectory Vector Embedding | Origin GPS | Destination GPS | Driver ID | Order Fare | Passenger Age |
|---|---|---|---|---|---|---|
| 1 | [0.12, -0.45, 0.78, ..., 0.56] | (39.912,116.404) | (39.952,116.408) | 1001 | 35.5 | 33 |
| 2 | [0.34, 0.67, -0.12, ..., 0.01] | (39.923,116.410) | (39.876,116.398) | 1002 | 42 | 25 |
| 3 | [-0.56, 0.23, 0.45, ..., 0.34] | (39.901,116.412) | (39.921,116.425) | 1003 | 28.5 | 21 |
| 4 | [0.78, -0.34, 0.12, ..., -0.23] | (39.945,116.401) | (39.912,116.390) | 1001 | 50 | 15 |
| 5 | [0.01, 0.45, -0.67, ..., 0.89] | (39.890,116.415) | (39.930,116.420) | 1004 | 38.5 | 12 |

**Figure 1: Data model in vector search engine FedVSE**

(ii) **Limited Query Support**. These works lack support for *hybrid queries*, which integrate a structured attribute filter with KNN search and are prevalent in modern vector databases [2].

**System Scope.** To address these limitations, we propose a Federated Vector Search Engine called FedVSE. We focus on the vector data model adopted in industrial vector databases [2, 4], where each entity contains both a vector embedding and structured attributes (see Fig. 1). This model necessitates support for *two query types*:

- **KNN Queries** identify the top-$k$ most similar entities.
- **Hybrid Queries** combine KNN search with attribute filters.

During the query processing, FedVSE ensures *two privacy requirements*: (i) clients learn nothing beyond their query answers, and (ii) no local vector database can infer private data from others.

**System Overview.** FedVSE comprises two key modules: *local vector databases* and *a central server*. Specifically, in each local vector database, FedVSE offers a dual-indexing scheme: built-in vector indexes and learned indexes for structured attributes. The central server coordinates query processing across all vector databases and is equipped with the TEE (*i.e.*, Intel SGX) to ensure privacy. FedVSE leverages the TEE to verify data access policies for each vector search request and securely processes private data from local databases. Moreover, we devise an optimized query processing framework that reduces both communication overhead and query latency through our indexing and pruning strategies.

**Contribution.** Overall, our main contributions are as follows:

(i) FedVSE is the first federated vector search engine of its kind.

(ii) FedVSE offers a privacy-preserving and efficient query processing framework that supports mainstream vector queries.

(iii) FedVSE is built upon an industrial vector database system, Milvus [2], and provides intuitive query interfaces for clients. To demonstrate this, FedVSE is deployed across six cloud servers and delivers a faster retrieval service than existing baselines [8, 11, 12] in applications like cross-platform trajectory similarity search.

## 2 SYSTEM OVERVIEW

This section introduces the basic concepts, system architecture, and query processing framework in our vector search engine.

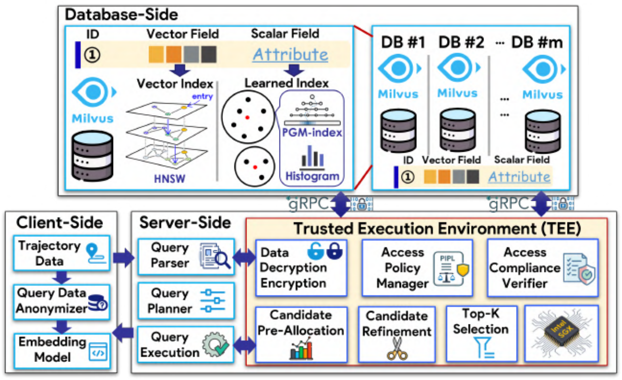


**Figure 2: System architecture of FedVSE**

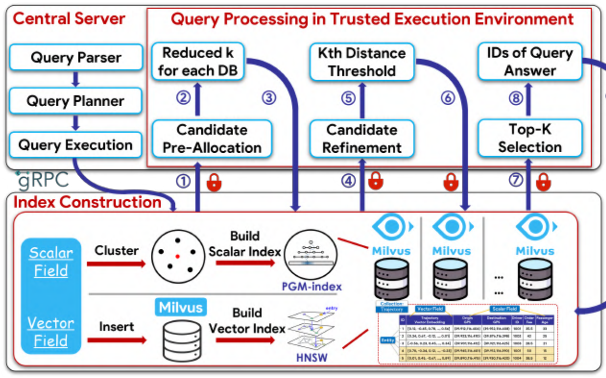


**Figure 3: Query processing framework in FedVSE**

## 2.1 Basic Concepts in Vector Databases

**Data Model.** We adopt the prevalent data model among modern vector database systems [4]. Formally, for a data **collection** $\mathcal{D}$ ("collection" as short), each (data) **entity** $o_i \in \mathcal{D}$ is denoted by a tuple $(id_i, v_i, \mathcal{M}_i)$, where $v_i \in \mathbb{R}^d$ is the embedding vector and $\mathcal{M}_i$ denotes the structured attributes associated with the entity. Data **schema** is composed of both the vector data field (*i.e.*, $v_i$) and scalar data fields (*i.e.*, $\mathcal{M}_i$). A database may contain multiple collections.

Fig. 1 depicts a collection of trajectory vector data. Here, each entity comprises: (1) an embedding vector generated by deep learning models, encoding the trajectory's spatio-temporal patterns, and (2) associated structured attributes (*e.g.*, origin and destination).

**Vector Queries.** FedVSE supports two primary vector query types:

(i) **KNN Queries**: Given a query vector $q$ and a distance metric $dis$, this query type retrieves the top-$k$ most similar vectors from a collection $\mathcal{D}$, including exact/approximate KNN and KNN join.

(ii) **Hybrid Queries**: This query type integrates KNN search with an attribute filter $\mathcal{F}(\mathcal{M})$ and restricts vectors in the query answer to satisfy the filter, such as "finding similar trajectories to $q$ with order fare < 20" from the dataset in Fig. 1.

**Index Scheme.** FedVSE adopts a dual-indexing scheme:

(i) **Vector Index**: three vector index families are adopted for accelerating KNN queries: FLAT (for exact KNN), IVF and HNSW (for approximate KNN with a high recall).

(ii) **Learned Index**: FedVSE also builds multi-dimensional learned indexes for structured attributes using PGM-index [3].

## 2.2 System Architecture

As shown in Fig. 2, the system architecture of FedVSE consists of two core modules: *local vector database* and *central server*. We also outline how clients interact with FedVSE in this subsection.

**Local Vector Database.** FedVSE operates on a federation of autonomous vector database systems. To enable efficient vector search, vector indexes (*e.g.*, HNSW [7]) are pre-built within each local database. We also build auxiliary learned indexes over structured attributes, which leverage machine learning models to reduce space overhead. Together, these indexes improve the system throughput.

**Central Server.** To satisfy the privacy requirements, the central server needs to be equipped with a TEE [6], Intel SGX. It processes each query request through the following core components:

(i) **Query Parser**. This component extracts query parameters from an incoming request, including the query vector, integer $k$, and attribute filter. The attribute filter is then transmitted to the TEE for access control verification. Access policies are initially expressed as structured attribute predicates by administrators and securely loaded into the TEE to prevent tampering. Access is only granted when the attribute filter in the query request fully complies with these access policies. For example, under China's Personal Information Protection Law (PIPL), personally identifiable data (*e.g.*, trajectories) for minors under 16 years old requires strict protection. In this context, an attribute filter $age \geq 12$ would conflict with this policy when applied to the dataset in Fig. 1.

(ii) **Query Planner**. Once access is granted, this component decomposes each federated vector search into vector queries at local databases and secure operations exclusively within the TEE. KNN join is decomposed into a series of independent KNN searches.

(iii) **Query Execution**. Following the query plan, this component coordinates the federated search process by (1) initiating concurrent searches at local vector databases to obtain local candidates, and (2) use the TEE to select the final answer from candidates.

**Client.** To use FedVSE, clients first anonymize queries by removing sensitive data. Then, they generate embedding vectors of anonymized queries, submit vector search requests, and await answers.

## 2.3 Query Processing Framework

Fig. 3 illustrates our query processing framework [5] as follows.

**(i) Index Construction.** In this phase, vector indexes, such as HNSW [7], are built inside each local database. For hybrid queries, we also build learned indexes over structured attributes. Specifically, we first partition vectors into multiple shards using balanced clustering, and then construct a PGM-index [3] for each shard. Finally, we also generate a distance histogram $\mathcal{H}_j$ for each shard to track the distances between contained vectors and their centroid $c_j$.

**(ii) Query Processing.** When executing a federated vector query, our algorithm operates via 3 sequential stages with 9 total steps:

**Steps 1–3: Candidate Pre-Allocation.** This stage estimates each local database's contribution to the final answer and pre-allocates the initial candidate size before performing local search.

At Step 1, each local database receives a vector query from the server and a subset of shards $C^*$ is selected based on Eq. (1).

$$C^* = \left\{ j \mid dis(c_j, q) \leq (1+\alpha) \cdot \min_j dis(c_j, q) \right\},\ \alpha \in [0,1] \quad (1)$$

Step 2 converts attribute filter into a multi-dimensional search window and derives the filter selectivity $sel_j$ with range counting over PGM-indexes. Consequently, the top $\lceil k/sel_j \rceil$ nearest neighbors to the query vector $q$ likely contain $k$ entities satisfying the filter. Since each histogram bin maintains vector counts with specific distance intervals, we can use the triangle inequality to derive the distance upper bound for each vector $v_i$ and query vector $q$:

$$dis^{\uparrow}(q, v_i) = dis(q, c_j) + dis^{\uparrow}(c_j, v_i) \tag{2}$$

where $dis^{\uparrow}(c_j, v_i)$ denotes the distance upper bound of histogram bin containing $v_i$. The minimal threshold $\gamma$ is computed through histograms, where the cumulative count of vectors across clusters $C^*$, satisfying $dis^{\uparrow}(q, v_i) \leq \gamma$, meets or exceeds $\lceil k/sel_j \rceil$.

At Step 3, the TEE receives the estimated $k$th nearest distance $\gamma_i$ to $q$ from each local database $\mathcal{D}_i$, and determines the initial candidate size as $k_i = k \cdot (\min_i \gamma_i)/\gamma_i$.

**Steps 4–6: Candidate Refinement.** This stage performs initial local candidate generation and refinement using the server's TEE.

At step 4, each local database executes a local vector search with reduced integer $k_i$ and discretizes candidates' distances into $\sqrt{k_i}$ intervals, which are then encrypted and sent to the TEE.

At Steps 5–6, the TEE decrypts the received distance intervals and establishes an upper bound for the global $k$th nearest distance to $q$. Then local databases are notified about refined intervals.

**Steps 7–9: Top-K Selection.** Local databases encrypt the refined candidates' distances to $q$ and send them to the TEE. After collecting the distances, the TEE performs a secure top-$k$ selection over these distances and requests the corresponding entities from each local database to assemble the final query answer.

**Privacy Guarantee.** FedVSE satisfies the two privacy requirements in Sec. 2.1. First, a client obtains at most $k$ entities as the query answer, learning nothing about other entities. Second, local vector databases operate in complete isolation from each other, eliminating any possibility of inter-database privacy leakage. Moreover, all sensitive data processing is isolated within the TEE, which offers strong security guarantee based on dedicated hardware [6, 12].

## 3 PROTOTYPE IMPLEMENTATION

This section introduces the detailed implementation of FedVSE, which leverages the open-source industrial vector database system Milvus [2] as its foundation for local databases. Client implementations are excluded as they are typically application-driven.

### 3.1 Local Vector Databases in Federation

FedVSE employs the high-performance vector database system Milvus [2] to manage each local vector dataset. Milvus supports diversified built-in vector indexes, such as FLAT, IVF, and HNSW. By using these indexes, Milvus enables efficient processing of both KNN queries and hybrid queries at scale. To balance query latency and result recall, we choose HNSW [7] as the default vector index.

For structured attributes associated with vectors, we also build learned indexes (*i.e.*, PGM-indexes [3]) and maintain the distance histograms outside Milvus. The number of clusters is dynamically configured between 10 and 100 depending on the data scalability. These auxiliary indexes incur minor memory footprints, so they remain memory-resident during the query processing.

To facilitate collaborative search, we encapsulate Milvus's native query interface with a gRPC [1] service, and enable seamless remote procedure calls from the central server. Besides, each database maintains an in-memory cache to store local candidates.

### 3.2 Central Server for Query Coordination

A central server coordinates privacy-preserving query processing using Intel SGX, and implements the following core components.

**Components Outside TEE.** There are three core components outside the TEE: *query parser*, *query planner*, and *query execution*. First, our FedVSE follows the format of query interfaces in Milvus to leverage its existing parser for extracting query parameters from client requests. Second, we use fixed strategies to generate query plans for KNN and hybrid queries. When access policies are specified, KNN queries transform into hybrid queries and are processed by detailed steps in Sec. 2.3. Otherwise, KNN query plans skip the candidate pre-allocation, since they have no attribute filter. Finally, the query execution processes each query plan sequentially.

**Components Within TEE.** There are two types of components within the TEE: *access policy verification* and *secure operations*.

**Access Policy Verification.** Access policies are described through constraints on structured attributes, offering entity-level access control. The access policy manager initially loads these policies from a configuration document. The access compliance verifier sequentially evaluates whether the attribute filter in query conditions overlaps with the preserved attribute range specified in policies.

**Secure Operations.** To ensure end-to-end privacy, local vector databases apply AES-128 encryption for all data transmitted to the TEE. Three operations are isolated within the secure enclave of Intel SGX: *candidate pre-allocation* (with hyper-parameter $\alpha = 0.5$), *candidate refinement*, and *top-k selection*. The latter two operations utilize a min-heap data structure to compute the distance threshold. Thus, oblivious sorting is used to maintain the min-heap.

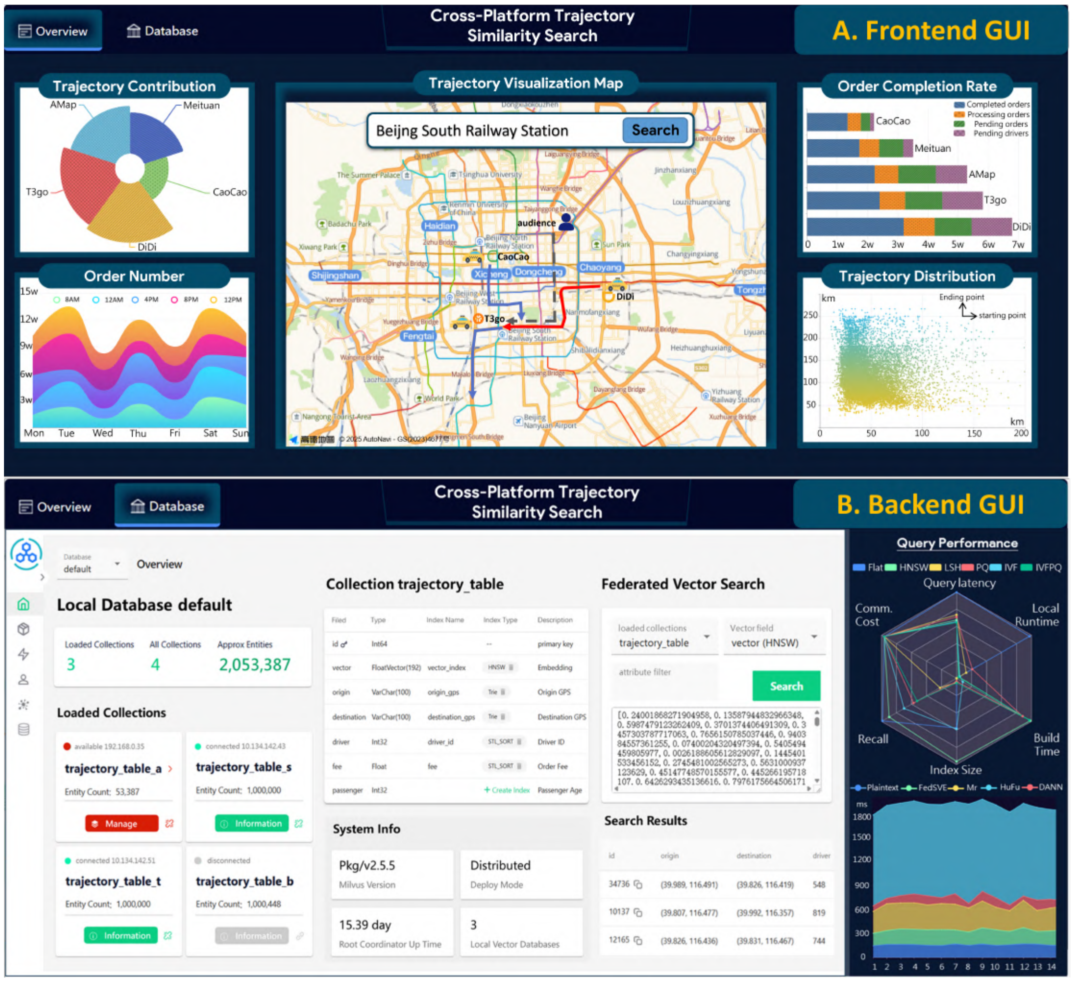
**Remark.** We also implement three state-of-the-art baseline methods in our system FedVSE: HuFu [8], Mr [11], and DANN* [12].

## 4 DEMONSTRATION SCENARIO

This section presents our demonstration plan in the application scenario of **cross-platform trajectory similarity search**. Trajectory similarity search is commonly used in intelligent transportation. Recently, multiple ride-hailing platforms (*e.g.*, Didi and T3go) collaboratively offer this trajectory analytic service to optimize dispatching between drivers and passengers [9, 10].

**Deploying FedVSE.** Five ride-hailing platforms generate the embedding vectors of their trajectories and upload the vector data into Milvus. As shown in Fig. 4, FedVSE is deployed across five cloud servers running Milvus, along with another server as central server. A client is implemented with anonymization methods using metric differential privacy. A stream of query workloads is generated from ride-sharing orders in a real platform, Didi. Moreover, we also implement frontend and backend GUI in this application scenario. The frontend GUI mainly shows the status of cross-platform trajectory similarity search, while the backend GUI primarily displays the status of our vector search FedVSE under this application scenario.

**Interaction with Audiences.** Fig. 4 (A) shows the frontend GUI. Four corners of this GUI illustrate the distribution of the ride-hailing orders along with each platform's contribution to query results. Audiences can also submit search requests via the central map



Database default — Overview

Local Database default

| Loaded Collections | All Collections | Approx Entities |
|---|---|---|
| 3 | 4 | 2,053,387 |

Loaded Collections

| | |
|---|---|
| available 192.168.0.35 | connected 10.134.142.43 |
| trajectory_table_a | trajectory_table_s |
| Entity Count: 53,387 | Entity Count: 1,000,000 |
| Manage | Information |
| connected 10.134.142.51 | disconnected |
| trajectory_table_t | trajectory_table_b |
| Entity Count: 1,000,000 | Entity Count: 1,000,448 |
| Information | Information |

Collection trajectory_table

| Filed | Type | Index Name | Index Type | Description |
|---|---|---|---|---|
| id | Int64 | | -- | primary key |
| vector | FloatVector(192) | vector_index | HNSW | Embedding |
| origin | VarChar(100) | origin_gps | Trie | Origin GPS |
| destination | VarChar(100) | destination_gps | Trie | Destination GPS |
| driver | Int32 | driver_id | STL_SORT | Driver ID |
| fee | Float | fee | STL_SORT | Order Fee |
| passenger | Int32 | | + Create Index | Passenger Age |

System Info

| | |
|---|---|
| Pkg/v2.5.5 | Distributed |
| Milvus Version | Deploy Mode |
| 15.39 day | 3 |
| Root Coordinator Up Time | Local Vector Databases |

Federated Vector Search

loaded collections: trajectory_table
Vector field: vector (HNSW)
attribute filter
Search

```
[0. 24001868271904958, 0. 13587944832966348,
0. 5987479123262409, 0. 3701374406491309, 0. 3
457303787717063, 0. 7656150785037446, 0. 9403
84557361255, 0. 07400204320497394, 0. 5405494
459805977, 0. 0026188605612829097, 0. 1445401
533456152, 0. 274548100256527З, 0. 5631000937
123629, 0. 45147748570155577, 0. 445266195718
107, 0. 6426293435136616, 0. 7976175664506171
```

Search Results

| id | origin | destination | driver |
|---|---|---|---|
| 34736 | (39.989, 116.491) | (39.826, 116.419) | 548 |
| 10137 | (39.807, 116.477) | (39.992, 116.357) | 819 |
| 12165 | (39.826, 116.436) | (39.831, 116.467) | 744 |

**Figure 4: Deploying our federated vector search engine FedVSE in cross-platform trajectory similarity search**

by specifying their destinations. Our system searches the most shareable (similar) trajectories and visualizes recommended routes.

Whenever an audience submits a trajectory search request, the backend GUI also visualizes the query vector (see Fig. 4 (B)). Additionally, audiences can change the default query parameters in the backend GUI, such as integer $k$ and attribute filters. The result entities for audiences' federated vector searches will be returned in real time. Besides, one additional chart reflects the comparisons of our FedVSE and existing baselines (HuFu [8], Mr [11], and DANN* [12]) in terms of query latency, which verifies our query processing framework's superiority. The other one depicts the comparisons between different vector indexes, such as Flat, HNSW, and IVF.

## Acknowledgments

This work was 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), the Beijing Natural Science Foundation (Z230001), the Fundamental Research Funds for the Central Universities No. JK2024-03, the Didi Collaborative Research Program and the State Key Laboratory of Complex & Critical Software Environment (SKLCCSE). Yuxiang Zeng and Yongxin Tong are the corresponding authors.

## References

[1] 2024. gRPC. https://grpc.io/
[2] 2024. Milvus. https://milvus.io
[3] 2024. The PGM-Index. https://pgm.di.unipi.it/
[4] Sebastian Bruch. 2024. *Foundations of Vector Retrieval.* Springer.
[5] Zeheng Fan, Yuxiang Zeng, Zhuanglin Zheng, et al. 2025. FedVS: Towards Federated Vector Similarity Search with Filters. In *SIGKDD.*
[6] Xiaoguo Li, Bowen Zhao, Guomin Yang, et al. 2023. A Survey of Secure Computation Using Trusted Execution Environments. *CoRR* abs/2302.12150 (2023).
[7] Yury A. Malkov and Dmitry A. Yashunin. 2020. Efficient and Robust Approximate Nearest Neighbor Search Using Hierarchical Navigable Small World Graphs. *IEEE Trans. Pattern Anal. Mach. Intell.* 42, 4 (2020), 824–836.
[8] Yongxin Tong, Xuchen Pan, Yuxiang Zeng, et al. 2022. Hu-Fu: Efficient and Secure Spatial Queries over Data Federation. *PVLDB* 15, 6 (2022), 1159–1172.
[9] Yongxin Tong, Jieying She, Bolin Ding, et al. 2016. Online mobile Micro-Task Allocation in spatial crowdsourcing. In *ICDE.* 49–60.
[10] Yuxiang Wang, Yuxiang Zeng, Shuyuan Li, et al. 2024. Efficient and Private Federated Trajectory Matching. *IEEE Trans. Knowl. Data Eng.* 36, 12 (2024), 8079–8092.
[11] Kaining Zhang, Yongxin Tong, Yexuan Shi, et al. 2023. Approximate k-Nearest Neighbor Query over Spatial Data Federation. In *DASFAA.* 351–368.
[12] Xinyi Zhang, Qichen Wang, Cheng Xu, et al. 2024. FedKNN: Secure Federated k-Nearest Neighbor Search. *SIGMOD* 2, 1 (2024), V2mod011:1–V2mod011:26.
[13] Dongfang Zhao. 2024. FRAG: Toward Federated Vector Database Management for Secure Retrieval-Augmented Generation. *CoRR* abs/2410.13272 (2024).
[14] Zeqi Zhu, Zeheng Fan, Yuxiang Zeng, et al. 2024. FedSQ: A Secure System for Federated Vector Similarity Queries. *PVLDB* 17, 12 (2024), 4441–4444.