FedSQ: A Secure System for Federated Vector Similarity Queries

Zeqi Zhu
SKLCCSE Lab,
Beihang University
zhuzeqi@buaa.edu.cn

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

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

Yexuan Shi
SKLCCSE Lab,
Beihang University
skyxuan@buaa.edu.cn

Yi Xu
SKLCCSE Lab,
Beihang University
xuy@buaa.edu.cn

Mengmeng Zhou
Beijing Academy of
Blockchain and Edge
Computing
zhoumm@baec.org.cn

Jin Dong*
Beijing Academy of
Blockchain and Edge
Computing
dongjin@baec.org.cn

ABSTRACT

Vector databases have emerged as crucial tools for managing and retrieving representation embeddings of unstructured data. Given the explosive growth of data, vector data is often distributed and stored across multiple organizations. However, privacy concerns and regulations like GDPR present new challenges in collaborative and secure queries, also known as federated queries, over those vector data distributed across various data owners. Although existing research has attempted to enable such query services for low-dimensional data, such as relational and spatial data, these solutions can be inefficient in answering vector similarity queries involving high-dimensional data. Therefore, we are motivated to develop a new prototype system called FedSQ that (1) ensures privacy protection across data owners and (2) balances query efficiency and result accuracy when processing federated vector similarity queries. To achieve these goals, FedSQ utilizes advanced secure multi-party computation techniques to prevent information leakage during query processing and incorporates indexing and sampling based optimizations to strike a proper performance balance.

PVLDB Reference Format:
Zeqi Zhu, Zeheng Fan, Yuxiang Zeng, Yexuan Shi, Yi Xu, Mengmeng Zhou, and Jin Dong. FedSQ: A Secure System for Federated Vector Similarity Queries. PVLDB, 17(12): 4441 - 4444, 2024.
doi:10.14778/3685800.3685895

1 INTRODUCTION

Vector databases [7] have become powerful tools for managing and retrieving embedding vectors, which effectively capture the semantic meaning of unstructured data. By executing vector similarity queries, such as k-Nearest Neighbor (kNN) search, these databases can quickly retrieve relevant information based on the similarity of the embedding vectors. Recently, vector databases have emerged as "retrieval plugins" seamlessly integrated into the generation process of Large Language Models (LLMs) [7]. By harnessing their retrieval capabilities, we can provide relevant external data to enhance the accuracy of LLMs like ChatGPT when responding to queries that extend beyond the training data.

Up until now, vector databases have primarily been designed to handle data that do not contain private information. Conversely, in private domains, data is often highly sensitive and difficult to integrate into a single database due to regulations like GDPR. Therefore, deploying a vector database in such application scenarios becomes even more challenging. A typical example is as follows.

Example 1 (LLM-based Medical Question Answering)**.** In the realm of medical informatics, LLMs are revolutionizing medical question answering systems [9]. Consider a scenario where an LLM receives a complex medical query that requires accessing external data like clinical cases or records related to the queried disease. In practice, these records contain highly sensitive patient information and are often distributed across multiple hospitals. Due to the privacy regulations like GDPR, each hospital must independently manage its own database, often in the form of a vector database (*e.g.*, Milvus [13]) containing embedding vectors of medical records. In this scenario, it is imperative to *provide joint and secure query processing services* across these hospitals' local vector databases.

Recently, data federation systems [11] have been proposed to securely process queries across multiple data owners, such as SMCQL [4], Conclave [12], and Hu-Fu [8, 10]. Existing work is usually designed to answer exact queries on low-dimensional data, like relational data [4, 12] and spatial data [8, 10, 14]. However, due to the high-dimensional nature of vector data, existing solutions can be inefficient in processing federated vector similarity queries across local vector databases, which typically require approximate answers. More specifically, the following technical challenges need to be addressed in a vector data federation system:

- **Privacy protection across data owners during query processing.** Each data owner could potentially act as a semi-honest attacker, aiming to infer sensitive information during the specified query processing protocol. Meanwhile, the query requester can only access information pertaining to the retrieved data, remaining unaware of the other data.
- **Balancing query efficiency with result accuracy.** Such queries need to be processed rapidly to prevent reduction in the efficiency (*e.g.*, inference latency of LLMs). Simultaneously, the query answer should be accurate and reliable enough to prevent misguidance, especially in model reasoning. Given the aforementioned

*Jin Dong is the corresponding author.

Proceedings of the VLDB Endowment, Vol. 17, No. 12 ISSN 2150-8097.
doi:10.14778/3685800.3685895

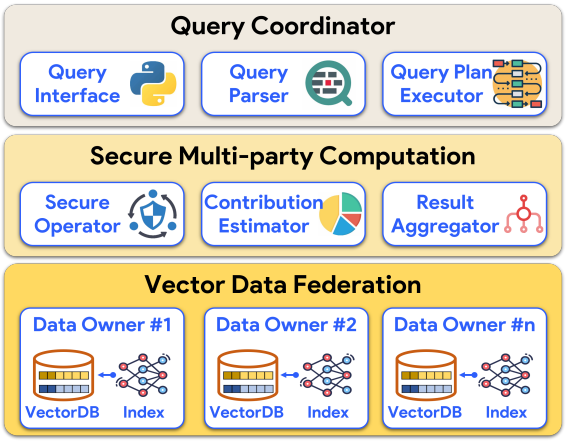

Figure 1: The architecture of FedSQ system

security requirement, it is very challenging to achieve a balanced trade-off between query efficiency and result accuracy.

To fill this gap, we aim to propose a prototype system for providing secure and efficient query processing in a vector data federation system. We have made the following major contributions:

- We have proposed a secure and efficient framework that processes federated vector similarity queries across multiple vector databases. This framework leverages both local plaintext queries and collaborative secure computations. Local plaintext queries are executed in vector databases to obtain partial results by using high-dimensional vector data indexes like HNSW [6]. Collaborative secure computations, such as secure aggregations of these partial results, are implemented using secure multi-party computation techniques.
- We have developed a user-friendly web client that enables DBAs to easily configure and monitor the vector data federation system, thereby enhancing the system's usability. Additionally, we have created easy-to-integrate APIs in Python to further facilitate the incorporation of our solution into existing data retrieval and analytics workflow.
- By combining previous core components, we have built a prototype system named FedSQ. Moreover, by using the open-source framework LangChain [1], FedSQ is integrated into a chat-based LLM and demonstrated to be useful in the application scenario of medical question answering.

The rest of this paper is organized as follows. Section 2 presents the system architecture and workflow. Section 3 introduces the implementation. Section 4 illustrates our demonstration plan.

2 SYSTEM OVERVIEW

This section introduces the overall architecture and workflow.

2.1 Architecture of Our FedSQ System

FedSQ consists of three important layers, namely *vector data federation*, *secure multi-party computation*, and *query coordinator* layers, as shown in Fig. 1. More details of these layers are as follows:

Vector Data Federation Layer. It consists of n data owners. Data owners can execute local vector similarity queries in vector databases.

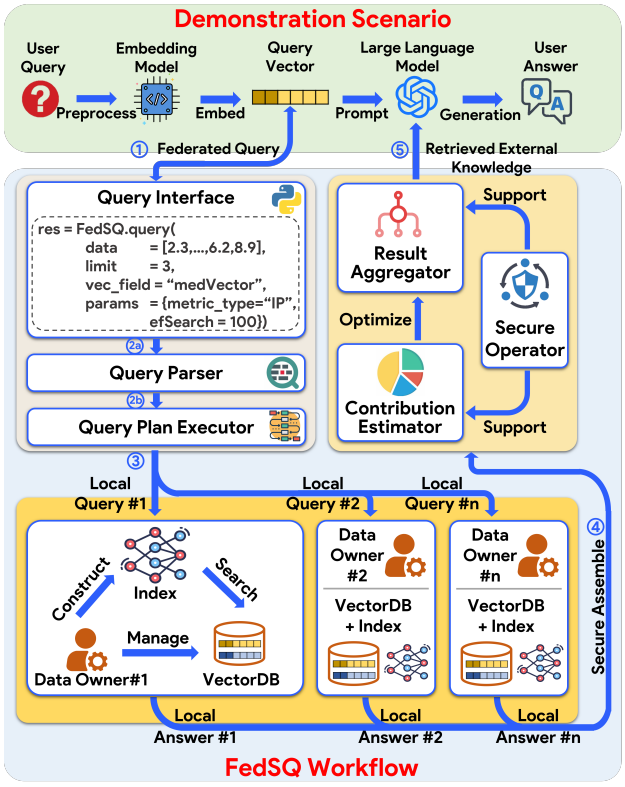

Figure 2: The workflow of FedSQ system

In addition, these local vector databases provide convenient and autonomous data management services for data owners.

Secure Multi-Party Computation Layer. This layer consists of three components: *secure operator*, *contribution estimator*, and *result aggregator*. When processing a federated vector similarity query, the contribution estimator quantifies the proportion of partial answers provided by each data owner. Then, the result aggregator assembles the partial answers from all data owners. To protect the privacy of each data owner, these two components are implemented via secure multi-party computation [5].

Query Coordinator Layer. This layer also includes three components: *query interface*, *query parser*, and *query plan executor*. The query interface enables users to directly interact with the system and submit their queries. The query parser performs syntax analysis on users' query requests and extracts query conditions. The query plan executor generates query plans according to the conditions and distribute them to the vector data federation layer for subsequent query executions.

2.2 Workflow of Our FedSQ System

Fig. 2 presents the main workflow of our FedSQ system. It starts with a federated vector similarity query and ends with the query answer. The entire processing procedure has the following steps:

Step 1. A user provides his federated vector similarity query in Python based on the retrieval function in vector databases [13]. For example, Fig. 2 shows a federated vector kNN query, where *data* denotes the query object and *limit* denotes the parameter k.

Step 2. This step involves both (a) query parser and (b) query plan executor. The query parser decomposes the Python code into query parameters and predicates. Based on the query conditions, the query plan executor decomposes the code into a series of plaintext vector similarity queries over local vector databases and secure aggregation protocols across the data owners, which are subsequently distributed to the other layers for query executions.

Step 3. Each data owner now receives the plaintext vector similarity queries. Local answers are obtained by executing these queries within their vector databases, accelerated by vector data index.

Step 4. Based on predefined secure aggregation protocols, the contribution estimator determines the proportion of partial results in each local answer. Subsequently, the result aggregator collects the estimated quantity from local answers via secure operations.

Step 5. The final answer will be returned to the user by the result aggregator component in the secure multi-party computation layer.

3 PROTOTYPE IMPLEMENTATION

This section introduces the implementation details of FedSQ.

Vector Data Federation Layer. In this layer, a vector database (Milvus [13]) is assumed to have been deployed by each data owner, which facilitates local vector similarity queries via efficient index like HNSW [6]. To balance between the index construction overhead and query efficiency, the maximum number of connections per node is set to 16 in HNSW, and 200 candidate neighbors are saved per node. Furthermore, the number of candidate neighbors accessed during the search phase of HNSW is dynamically adjusted, thereby providing an appropriate query latency across varying data sizes.

Secure Multi-Party Computation Layer. The secure multi-party computation layer is integrated into an Django backend, serving as a middleware that connects the other two layers. The CrypTen library [2] is employed to implement secure operation protocols, whereas the Protobuf library [3] enables efficient communication among the data owners. Moreover, the contribution estimator is designed by following the idea of "sample-and-seek":

(1) **Sample.** The estimator begins by uniformly sampling from all data owners to construct profiles for their local answers.

(2) **Seek.** According to the profiles, specific non-uniform sampling techniques are employed to seek the complete query result.

Take the federated vector kNN query as an example. It will be decomposed into several local kNN queries. After performing these queries, each data owner will hold his local kNN answers. Our contribution estimator can first **sample** the nearest neighbor (distance) from each data owner. Based on the similarity of these samples, the estimator can deduce which vector database potentially holds more relevant data to the query object. Thus, an iterative process is employed to **seek** more samples from this database until k objects have been collected. Finally, a secure set union operator collects the vectors and their raw data as the final answer.

Query Coordinator Layer. We develop a user-friendly front-end query interface utilizing the Vue.js framework, which enables users to conveniently manage the vector data federation, post federated vector similarity queries, and monitor the overall system status. The query parser and query plan executor are also integrated in the Django back-end. We also integrate the Celery task queue for efficient asynchronous execution of query tasks, and create easy-to-integrate APIs in Python to facilitate the incorporation of vector databases into existing data retrieval and analytics workflow.

4 DEMONSTRATION SCENARIO

In this section, we introduce our demonstration plan for the audiences. We will show the GUI interface of FedSQ and demonstrate how to utilize FedSQ in LLM-based medical question answering.

4.1 GUI Interface of FedSQ

FedSQ offers a unified interface for query users, providing a cohesive and user-friendly system for both usage and management purposes. As shown in Fig. 3, query users can obtain essential information about the vector data federation from the system, such as the input query interface and retrieved answers, the accessible data schema, and the participated data owners.

Query Interface. In this module, query users can write and submit their vector similarity queries. They have the option to submit queries as a single Python statement or by uploading an executable file. Below the input area, users can view both the current and historical query results. The query user can choose to download the retrieved vector embeddings and the corresponding raw data.

Data Schema. The data schema displays the available attributes of the vector data federation, including the raw data, vector embedding, and other meta-data. For ease of vector data management, FedSQ supports a series of data types. For example, in Fig. 3, "FLOAT_VECTOR" represents the vector data, and "TEXT" represents the raw data type. The DBAs can manage the schema through the "Delete" button and the "Add New Attribute" button.

Participated Data Owners. This module provides a management board about the current status of data owners, which includes their names, IPs, update times, etc. Moreover, the DBAs of this vector data federation system can remove or add data owners flexibly.

Status of Vector Data Federation. FedSQ further provides a graphical module to show the current status of the vector data federation. The connected and disconnected data owners are depicted in different colors, allowing users to easily identify which vector database is available through the visualization.

4.2 Medical Question Answering by FedSQ

We also show audiences how our FedSQ system can be deployed into the scenario of LLM-based medical question answering.

LLM Configuration. We develop a user-friendly web interface that integrates our FedSQ with LLMs. The interface allows users to input queries, adjust LLM parameters, and obtain results. The configuration menu on the left enables users to adjust settings related to LLM. For example, users can select the underlying LLM model or set the temperature parameter, among other options. On the right side of the interface, there is a conversation window where users can enter their medical questions in the text box, and the generated responses are displayed directly on the same interface.

Medical Query without FedSQ. When users ask questions without FedSQ, the LLM relies solely on its internal knowledge embedded within the model. As a result, the responses may lack immediate

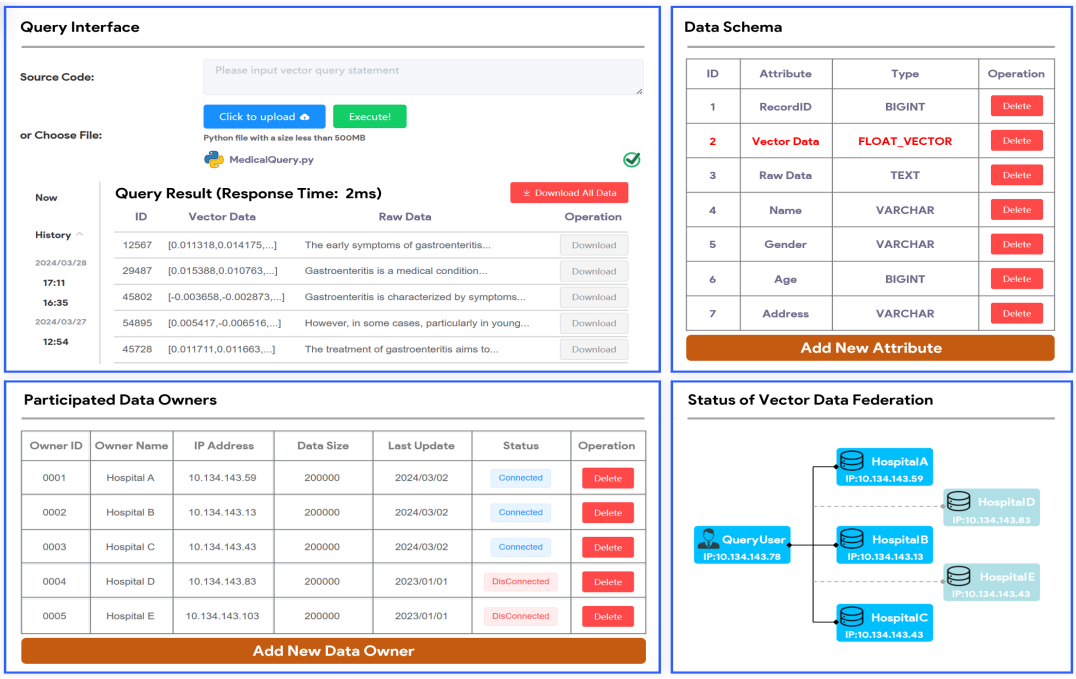

Figure 3: The GUI interface of our FedSQ system

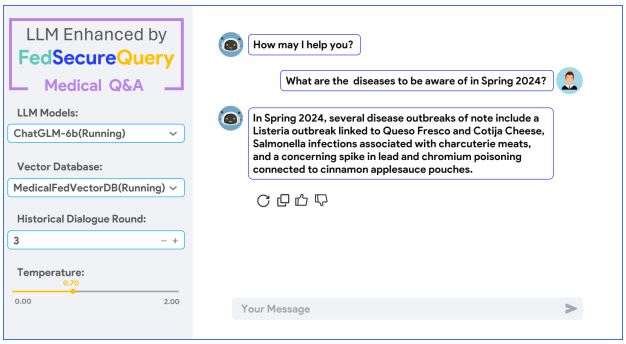

Figure 4: Medical question answering enhanced by FedSQ

relevance due to the absence of real-time updates. For example, if a user asks about "new diseases to be cautious of in Spring 2024", a LLM (*e.g.*, ChatGLM) often acknowledges its inability to provide this information for 2024. Instead, it offers general advice on common seasonal ailments such as pollen allergies and influenza.

Medical Query with FedSQ. By using LangChain [1] to integrate our FedSQ system into the inference procedure, the LLM collaborates with the external vector databases to generate answers. This integration allows the model to access fresh data, ensuring that the outputs are based on the latest and most relevant information. In our demonstration, the vector data federation consists of several data repositories from different hospitals. As shown in Fig. 4, when presented with the same query about new diseases to be cautious of in Spring 2024, the LLM informs the user about recent disease outbreaks. For instance, it can mention a Listeria outbreak linked to certain dairy products or a surge in lead and chromium poisoning associated with specific food items due to the retrieved external data by federated vector similarity queries over the data federation.

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. U21A20516, 62336003, 62076017) and Beijing Natural Science Foundation (Z230001), Beijing Advanced Innovation Center for Future Blockchain and Privacy Computing(GJJ-23-004), CCF-Huawei Populus Grove Fund, the Basic Research Funding in Beihang University No.YWF-22-L-531, and Didi Collaborative Research Program NO2231122-00047. Jin Dong is the corresponding author.

References

[1] 2022. LangChain. https://github.com/langchain-ai/langchain
[2] 2024. CrypTen. https://github.com/facebookresearch/CrypTen
[3] 2024. Protobuf. https://github.com/protocolbuffers/protobuf
[4] Johes Bater, Gregory Elliott, Craig Eggen, et al. 2017. SMCQL: Secure Query Processing for Private Data Networks. *PVLDB* 10, 6 (2017), 673–684.
[5] Marcel Keller. 2020. MP-SPDZ: A versatile framework for multi-party computation. In *CCS*. 1575–1590.
[6] Yury A. Malkov and Dmitry A. Yashunin. 2020. Efficient and Robust Approximate Nearest Neighbor Search Using Hierarchical Navigable Small World Graphs. *IEEE TPAMI* 42, 4 (2020), 824–836.
[7] James Jie Pan, Jianguo Wang, and Guoliang Li. 2024. Vector Database Management Techniques and Systems. In *SIGMOD*. 597–604.
[8] Xuchen Pan, Yongxin Tong, Chunbo Xue, et al. 2022. Hu-Fu: A Data Federation System for Secure Spatial Queries. *PVLDB* 15, 12 (2022), 3582–3585.
[9] Karan Singhal, Shekoofeh Azizi, Tao Tu, et al. 2023. Large Language Models Encode Clinical Knowledge. *Nature* 620, 7972 (2023), 172–180.
[10] 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.
[11] Yongxin Tong, Yuxiang Zeng, Zimu Zhou, et al. 2023. Federated Computing: Query, Learning, and Beyond. *IEEE Data Eng. Bull.* 46, 1 (2023), 9–26.
[12] Nikolaj Volgushev, Malte Schwarzkopf, Ben Getchell, et al. 2019. Conclave: secure multi-party computation on big data. In *EuroSys*. 1–18.
[13] Jianguo Wang, Xiaomeng Yi, Rentong Guo, et al. 2021. Milvus: A Purpose-built Vector Data Management System. In *SIGMOD*. 2614–2627.
[14] Xinyi Zhang, Qichen Wang, Cheng Xu, et al. 2024. FedKNN: Secure Federated k-Nearest Neighbor Search. *SIGMOD* 2, 1 (2024), V2mod011:1–V2mod011:26.