International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 09 Issue: 03 | March - 2025 SJIF Rating: 8.586 ISSN: 2582-3930

Secure Natural Language Querying Across Distributed Data with RAG and Reverse Engineering

Author: Chitresh Goyal
Independent Researcher | AI, NLP & Secure Systems

Abstract

As enterprises store growing volumes of data across disparate systems and silos, enabling intuitive and secure querying becomes a major challenge—especially when metadata is sparse or user queries are context-driven. This whitepaper proposes a Retrieval-Augmented Generation (RAG)-based approach combined with reverse-engineering techniques to securely handle natural language queries. By exposing only metadata to LLMs and dynamically mapping user queries to actual file locations and data columns, the system preserves data security while delivering accurate, context-aware responses (Ni et al., 2025).

Executive Summary

Natural language querying is emerging as the default interface for analytics. Traditional systems rely on rigid schemas and structured queries, which present usability challenges for business users. This whitepaper introduces a solution that integrates RAG and NLP-based reverse engineering to provide secure, schema-agnostic querying capabilities. The model ensures that actual data never leaves the local environment and leverages only metadata for language model interaction. This architecture is informed by previous works on secure data visualization, embedded schema matching, and sandboxed natural language interfaces. Our architecture uses metadata-aware RAG and reverse-engineering NLP to allow users to query distributed datasets securely, even without knowing schema details. It maintains security by preventing raw data exposure and dynamically maps data values to the correct files and schema columns using embeddings and local logic.

1. Introduction

In real-world enterprise settings, data is distributed across multiple files and formats, often without standardized metadata. Users want to interact using plain natural language, asking business questions like 'What were Q2 sales for Acme Corp?'. This paper presents a system designed to support such interactions while maintaining data privacy, using a RAG-based pipeline augmented by reverse engineering and local NLP matching. The methodology builds upon techniques I previously developed in secure GenAI-assisted querying, visual analytics, and Streamlit-based enterprise tools (Goyal, C. (2024)).

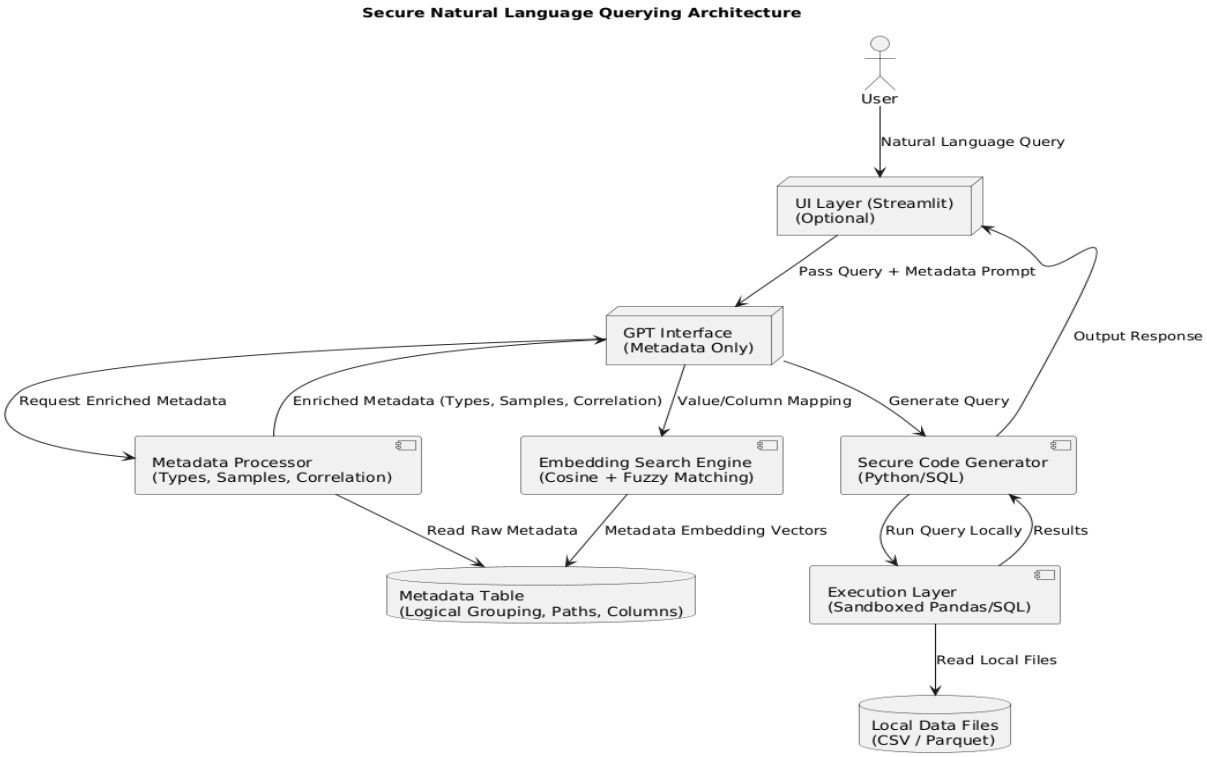
2. Challenges in Secure Distributed Querying

- **Relevant Data Source Discovery**: Identify the correct files dynamically without user-specified filenames. Data is split by time, region, or business units. The system must identify the right files automatically from the query.

- **Metadata-Only Exposure**: GPT receives metadata only. For compliance, only metadata is accessible to the LLM. All raw data must remain local and protected
- **Value-to-Column Reverse Mapping** or **Schema Free Data Matching**: Users may reference values like "Facebook" without knowing the related schema (e.g., social_media). The system must reverse-map these values securely.

3. Solution Architecture

The system is composed of metadata tables, local embedding-based search, and secure execution modules for interpreting user prompts. The RAG pattern is used not for full text retrieval but as a way to bind file metadata and local schema to user intent.



3.1 Metadata Table

A metadata table acts as a secure lookup interface:

Logical Grouping	File Name	File Path
Q1 Sales	q1_sales.parquet	/data/files/q1_sales.parquet
Q2 Sales	q2_sales.parquet	/data/files/q2_sales.parquet
All Regions	all_data.parquet	/data/files/all_data.parquet

3.2 Query Scenarios

Scenario A: Metadata-Based Query
Query: "What is the revenue trend for Q1?"

- Identify Q1 file
- Find "revenue" in metadata
- Map "revenue" → total_revenue
- Run local query

```
python

df = load_file('q1_sales.parquet')
result = df['total_revenue'].resample('M').sum()
```

Scenario B: Data-Value-Based Query
Query: "What is Z'Oréal's total revenue in Q2?"

- Identify Q2 file
- Find "Z'Oréal" in local data using algorithms
- Retrieve "brand" column using relevance mapping.
- Map "revenue" → total_revenue
- Run local query

```
python

df = load_file("q2_sales.parquet")
result = df[(df['brand'] == "Z'Oréal")]['total_revenue'].sum()
```

3.3 Reverse Mapping via Embeddings & Local NLP Search

- Using prior research on embedding-based NLP interfaces (Goyal, 2024), we use cosine similarity to match keywords in queries to column values and names. This enables flexible value-to-schema matching.
- Perform fuzzy + vector similarity matching to determine the most probable mapping (Radeva et al., 2024)

3.4 Query Formation

Queries are generated in secure Python code or SQL and executed locally, isolating raw data from the LLM layer.

3.5 GPT Integration via RAG

GPT is used only for interpreting the user prompt and formulating explanations—not for executing code or viewing data. This follows secure practices as outlined in prior work by the author in enterprise-grade Streamlit-GPT systems.

4. Technical Implementation Overview

Core Components

- **LLM Integration**

GPT is used with metadata-only prompts to interpret user intent securely, ensuring no raw data is exposed.

- **Secure Data Loader**

Loads structured files (CSV/Parquet) based on metadata-indexed file references; supports secure temporary storage during execution.

- **Embedding Search Engine**

Performs semantic search using vector embeddings and cosine similarity to match query phrases to column names and data values.

- **Query Generator & Execution Layer**

Dynamically generates Python or SQL code for local execution, using filters mapped from user input and schema detection.

- **Auditing and Access Control (Planned)**

Includes design provisions for row-level data masking, access control, and audit trails for enterprise compliance.

5. Use Case Workflows

Case 1: Metadata-Based Query

This workflow illustrates how a user query leveraging known metadata is securely mapped to files and schema, enabling local query execution without exposing raw data.

User Query → Metadata Table Lookup → File Selection → Column Mapping → Code Generation → Local Execution

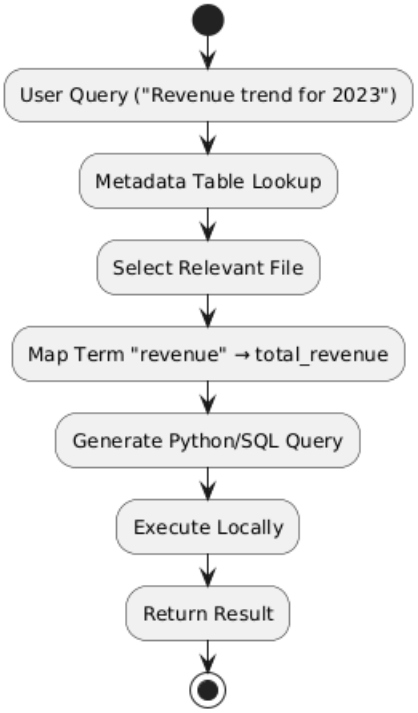


Figure 1: Metadata-Based Query Workflow

Case 2: Data-Value-Based Query

This workflow demonstrates how value-based queries are resolved by locating data references through embedding search, followed by secure local execution.

User Query → Metadata Lookup → File Match via Value Embeddings → Column Mapping → Code Generation → Local Execution

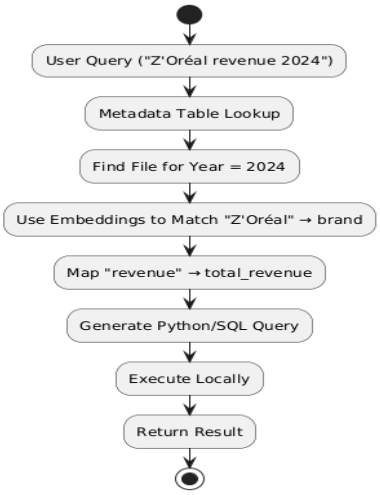


Figure 2: Data-Value-Based Query Workflow

These workflows demonstrate that both metadata-driven and value-based queries can be resolved without revealing data to external LLMs, using local embeddings and secure code execution.

6. Benefits

- **Data Security**: Data never leaves local context. No raw data is sent to GPT.
- **Flexibility**: Schema-agnostic query support. Works across schema- or value-driven queries
- **Scalability**: New domains can be added by extending metadata (CSV, Parquet, SQL etc.)
- **Compliance**: Built on proven OpenAI-secure interfaces and Metadata-only prompts satisfy governance constraints
- **User-Centric**: Designed and validated through prior deployments and allows conversational querying for non-technical users

7. Conclusion

This approach enables secure, intelligent querying of distributed datasets without compromising privacy. Combining Retrieval-Augmented Generation with NLP-based reverse engineering gives users a natural interface while maintaining enterprise-level control.

This architecture builds upon my prior work in secure AI-powered insight generation platforms, including:

- Democratizing CSV-based GenAI Analysis (2024)

- Secure Visualization with GPT in Python (2024)

- Streamlit-Snowflake Native GPT Apps (2024)

8. Acknowledgements

This work extends and integrates concepts developed in my earlier research on secure GenAI-assisted analytics and enterprise NLP workflows. Key foundations were laid through prior projects described in point 7.

Appreciation is also extended to the broader GenAI research community and open-source contributors whose work on RAG, vector search, and LLM tooling provided essential building blocks for this architecture.

9. References

- Goyal, C. (2024). *Democratizing Data Visualization with Pandas and GenAI*. IJSREM.
- Goyal, C. (2024). *Leveraging GenAI for Python-Based Visual Analytics*. IJSREM.
- Goyal, C. (2024). *Snowflake's Streamlit GenAI Strategy*. IJSREM.

10. Future Work

- Support for real-time data streams and API-based data ingestion.
- Addition of semi-structured formats like JSON, XML, or Excel.
- Feedback-driven learning loop to improve column and schema inference accuracy over time.
- Integration with enterprise IAM for role-based query access and policy enforcement.
- Incorporation of vector databases for faster and scalable semantic search.