A Provenance-Aware Vector Database for Explainable Retrieval-Augmented Generation Systems

Huy Truong
Dept. of Computer Science
University of Illinois, Chicago

12/2/2024

1 Introduction

Provenance tracking within vector databases addresses the importance of explainability in Retrieval-Augmented Generation (RAG) systems. We develop a provenance-aware vector database system implemented in PostgreSQL and pgvector, a novel approach to capturing token-level influence for Why-provenance. The presented system is not intended to be a full-fledged database system meant for general use, but rather a proof-of-concept to demonstrate the ease and importance of incorporating provenance into a vector database. Therefore, features like data lineage tracking and concurrent query processing are not explored but could serve as extensions of the work presented in this paper.

2 Background

Here, we provide a comprehensive overview of RAG and its significance in advancing Large-Language-Model-based chatbot systems. RAG is an ingenious design pattern that allows the chatbot system to search for externally sourced documents, which contrast the LLM's internal knowledge, and add those documents to the LLM's context. For example, the user may query an LLM about **the hometown of the 46th president of the United States**. However, the LLM is trained on the entire internet before the year 2010 so it does not know the 46th US president, but it knows the hometown of a politician named Joe Biden. A good chatbot system should have a sub-process to determine if the given user query requires the LLM to expensively consult external knowledge bases, like a database or the internet, depending on the meta-data of the LLM powering the application and certain attributes of the user query. **We will not consider and record this sub-process in our provenance tracking**. Let's assume any user query triggers a retrieval operation from a database for our purpose. **This is where we begin our provenance tracking**. Using a pre-trained embedding model ϕ, the chatbot system embeds the user query sentence into a numerical vector.

This numerical vector is sent to the vector database, containing thousands of textual data paired with their numerical vector, produced by the same model ϕ so they live in the same embedding space as the user query's vector. You can think of the user query's vector as a key and those numerical vectors in the database as locks on a vault containing a large chunk of text snipped out of some document. The locks that the key can open **best** will allow the text data in that vault to be sent back to the LLM. How we determine the similarity between 2 vectors is typically the cosine distance between them. Let's say the top chunk of text that matches the user query writes the current US president is Joe Biden. Therefore, this chunk of text is retrieved. **This marks the end of our provenance tracking**. Of course, all these retrieved text chunks must be formatted systematically for the LLM to consume. Combined with the LLM's internal knowledge, the LLM can confidently answer Joe's hometown, **all of which we will not cover in our project's scope**. This also means that our provenance tracking will be architecture-agnostic, meaning the specific implementations of the LLM and the sentence embedding model ϕ will not matter.

As discussed, the role of a vector database is to enable an efficient semantic search for RAG. However, we emphasize the limitations of current RAG systems in terms of explainability. In this project, we focus on formalizing and developing the Why-provenance for RAG because our primary goal is to explain why the user query triggers certain chunk's retrieval. We propose a fast-approximate way to do this explanation but highlight that existing explanation techniques (e.g., LIME, SHAP) can be applied directly to vector-based retrieval [2, 1]. Finally, we emphasize that this provenance framework is not a general-purpose provenance system. We are only interested in capturing how the token-level provenance influences search results from database queries, not how the data was generated, transformed, and fed to the LLM.

3 System Architecture and Design

This section details the architecture of our provenance-aware vector database. We choose PostgreSQL as the underlying database system, emphasizing its extensibility and support for vector data types through the pgvector extension. We could have built a specialized database system from scratch to implement the provenance tracking, but doing so requires dealing with the more general database problems such as indexing and scalability. Building from existing systems, such as Postgres, will allow us to focus more on provenance itself. The all-mpnet-base-v2 embedding model and its sentence-embedding space are well-studied, semantically parsed, and open-source to public use so it is our choice for ϕ. We also highlight the importance of consistently tokenizing text chunks and user queries using this model's tokenizer to maintain the integrity of the semantically rich embedding space. The embedding model is a crucial component in our provenance framework, as we rely on it to determine the level of semantic influence of the provenance-tracked data.

The schema for the documents table

```
CREATE TABLE IF NOT EXISTS documents (
id SERIAL PRIMARY KEY, – unique id for each chunk
filename TEXT, – the doc file name
chunk_text TEXT, – the chunk in this doc
start_index INT, – absolute start token index in the doc
end_index INT, – absolute end token index in the doc
embed_model TEXT, – name of the embedding model
num_tokens INT, – number of tokens in the chunk
embedding VECTOR(768) – result of running the embedding model on the chunk text
);
```

Finally, we introduce the provenance table, defining its structure and connecting it to the documents table through the chunk_id foreign key.

The schema for the provenance table

```
CREATE TABLE IF NOT EXISTS provenance (
chunk_id INT REFERENCES documents (id), – reference which chunk in the documents table is being tracked
query_text TEXT, – the user query that triggers this provenance record
embed_model TEXT, – name of the embedding model, should match that of this chunk
top_k INT, – the top number of retrieved chunk the user used
time TIMESTAMP, – the time of the user query
query_influence JSONB – token-level influence of the query on this chunk being retrieved
);
```

4 Provenance Model and Calculation

This section dives deeper into our provenance model, explaining our theoretical motivation for token-level influence. We use an approximation based on the cosine similarity between the user query token and chunk embeddings to estimate token influence scores. This assumes that the embedding space induced by the sentence-embedding model captures meaningful token-level semantics, which is reasonable because most sentence-embedding models perform some specialized pooling of each token embedding in the sentence to arrive at the sentence embedding. We could have implemented a full gradient-based calculation on the embedding model parameters; however, this often requires significantly more computational resources. We argue that our approximation provides an adequate trade-off between efficiency and accuracy for capturing provenance information in the context of interactive RAG.

5 Implementation

This code base comes with the file main.py that implements the provenance-aware vector database in Python using the psycopg2 library for PostgreSQL server interaction. We present the get_embedding_data function, serving an important role in reading document files, chunking documents, tokenizing, and embedding chunks. This data is inserted into the documents table. We present the rag_with_provenance function, which not only retrieves the top-k similar chunks but also calculates and stores their provenance information in the provenance table. It is important to note that our RAG provenance tracking happens for the same database query - table accessing followed by sorting by vector similarity scores - across different user queries to the chatbot system. Additionally, we use efficient methods like execute_values for bulk insertion into the database, which addresses scalability concerns by minimizing database round trips. The database server used is at localhost and has port 5432 on setup. The database name and username are specified by the database admin and used as credentials to establish the connection.

6 Use cases

We explore 2 common use cases of a client interacting with this provenance model:

1. The client is curious why many of their queries lately keep retrieving particular chunks of text about music from the database. Essentially, they want to see the user's queries, retrieved chunks, and their token-level influences so the following query would work fast and effectively: SELECT p.query_text, d.chunk_text, p.query_influence FROM provenance p JOIN documents d ON p.chunk_id = d.id WHERE p.query_text LIKE '%blues%';

2. The client prompts the chatbot about a research body, likes the bot's responses, and wants to cite sources for their paper. Essentially, they want to see which source documents were used and where the retrieved chunk of text is located in those documents. All this information is saved in our database so we can use the following query: SELECT d.filename, d.start_index, d.end_index, d.chunk_text FROM provenance p JOIN documents d ON p.chunk_id = d.id WHERE p.time = (SELECT MAX(time) FROM provenance);

7 Discussion

Our provenance model could be evaluated through user studies to assess its effectiveness in enhancing explainability and trust in RAG. We could have performed more quantitative evaluations; however, we feel qualitative demonstrations are more appropriate due to the subjective nature of judging explainability. Prospective future directions include exploring alternative why-provenance and other provenance models, incorporating uncertainty quantification and possible attribute dependency models, and scaling the system to handle massive datasets. Additionally, a natural extension to this system includes implementing efficient temporal data procedures, such as inserting

and embedding new documents without rebuilding the whole database. This necessitates more sophisticated provenance storing and querying pipelines. Finally, we emphasize the potential of our approach to enhance transparency and trust in AI systems, opening up new avenues for research in data provenance and explainable AI.

References

[1] Scott Lundberg. “A unified approach to interpreting model predictions”. In: *arXiv preprint arXiv:1705.07874* (2017).

[2] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. “” Why should i trust you?” Explaining the predictions of any classifier”. In: *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*. 2016, pp. 1135–1144.