LlamaIndex-based and LLM-powered Q&A Management System

Author: Ju Wu

1, Configuration and Requirement

The operation systems used is Microsoft Windows 11 Pro.

The virtual environment is built on anaconda software for package, library, environment, dependency management for this project.

The detailed information including the version of the python and associative packages are as follows:

It is not allowed to install the llama index packages using proxies due to network config restrictions for pip install, and conda install can’t find this package. So you need to install under domestic network and run the follows: pip install llama-index -i <https://pypi.tuna.tsinghua.edu.cn/simple>. Similar for chromadb: pip install chromadb -i <https://pypi.tuna.tsinghua.edu.cn/simple>.

To solve the merging issue regarding different version of llamaindex, you need to “pip install llama-index llama-index-vector-stores-chroma -i <https://pypi.tuna.tsinghua.edu.cn/simple>.”

Actually many inconveniences are caused by the great fire wall, such as installing necessary packages, api requests can’t be called through built-in functions, but their necessary steps need to be decomposed and modifed, and the involved data formats need changes as well. The main issue is about calling openai api during the whole procedure of llama index: usually the LLM used for embedding and querying is set up before llama index usage, LLM can be both locally or called remotely such as openai api, but openai api can’t be called even if we use the GPT service provided by app4gpt company, so we need to circumvent a lot on many different necessary functionalities of llama index, causing unexpected bugs sometimes. So another choice would be to install a open-sourced LLM locally, in this way, everything solved, but local model is subject to a bit long processing time. For local embedding model, we have “pip install llama-index-embeddings-huggingface -i <https://pypi.tuna.tsinghua.edu.cn/simple>”. pip install -U sentence-transformers -i <https://pypi.tuna.tsinghua.edu.cn/simple>. But the local model inference speed is subject to performance of our computer.

Also, there are many fucking problems with llama index itself, it has many mismatch between its documentation and package installed regarding function and module naming, hierarchy and so on.

Some unexpected errors occur when save vector data to vector database chromadb persistently, and there seems lack of methods to retrieve only the desired set of documents from the built collection in chromadb, adding the documents incrementally. Thus, we decide to build in-memory chromadb storage, and initialize that upon user entering his user dashboard, by utilizing this temporary vector storage, this user can query with GPT. Anyway, as the number of users and uploaded documents increase, a solution without repeated indexing and embedding is in need.

2, Backend design

RAG Q&A engine:

Regarding the Q&A mechanism, retrieval augmented generation (RAG) is utilized, and the general work flow for that is as follows:

1), Embed: Text data from the knowledge base is processed through an embedding model to generate vector representations.

2), Index: These embeddings are then indexed using an appropriate data structure to optimize search.

3), Query Embedding: When a query is received, it is also converted into an embedding using the same model.

4), Retrieve: The indexing system uses this query embedding to quickly find the most relevant document embeddings.

5), Generate: The retrieved documents are then fed into the LLM, along with the query, to generate an informed and accurate response.

We begin from the simplest use case:

A user has multi-round chats with the system, and in each round of the chat, all the documents collected from user uploads and web scrapers need to be indexed, and all the most relevant content from these documents from either direct upload by users and web scraper need to be used alongside with the prompt template and user query to generate the answer with the help of LLM.

The native method is to index the integrated content from all the documents stored (you can archieve simply), and the more complex one is to index each document separately, which can offer significant advantages, particularly in terms of scalability and efficiency when dealing with updates such as new document uploads, content modification of the documents. The second one will be used in our case, and the brief comparison for the two methods are as follows:

**Integrated Indexing of All Content**

Pros:

1), Simplicity: Single index to manage.

2), Query Performance: Potentially faster queries as all data is in one index.

Cons:

1), Re-indexing Overhead: Every time a new document is added or other modifications on the content happen, the entire index potentially needs to be updated, which can be time-consuming and resource-intensive.

2), Resource Intensive: Requires more computational resources as the index size grows.

**Separate Indexing of Each Document**

Pros:

1), Scalability: More scalable as adding or updating a document only requires re-indexing that specific document.

2), Efficiency: Reduces unnecessary processing since only new or updated documents are processed.

3), Flexible Updates: Easier to manage and update individual pieces of content without affecting the entire dataset.

Cons:

1), Complexity in Retrieval: More complex retrieval logic as you must aggregate results from multiple indices.

2), Potential for Inconsistency: More moving parts could lead to inconsistencies if not managed properly.

To avoid that the content from the totally irrelevant document is extracted and then is treated equally during answer generation, we need to also set a threshold number TH, first the highest similarity score from each of the documents’ indexing is ranked, then only the relevant chunks retrieved from the content of documents with the biggest TH similarity scores will be used for answer generation. This filtering design has the following benefits:

**Benefits of Using a Similarity Threshold (TH):**

1), Improves Relevance: By filtering out documents or chunks that do not meet the threshold for similarity, you ensure that only content that is most likely to be relevant to the query is used in the answer generation process.

2), Reduces Noise: Eliminating lower-scored documents from the answer synthesis phase prevents irrelevant information from diluting the quality of the answer.

3), Enhances User Satisfaction: Users receive answers that are more concise and relevant, which can significantly improve their satisfaction and trust in the system.

4), Optimizes Resource Use: Processing only the most relevant chunks reduces computational load and can minimize API usage, which is particularly important when using costly LLM APIs.

**Database and how to connect with the App:**

The relational databases are used for managing user access to documents, handling user feedback, and monitoring chat history, and so on. We use Database: PostgreSQL, ORM: SQLAlchemy (to interact with the database from Flask), and API: Flask RESTful (for creating RESTful APIs). Four main tables: ‘Users’, ‘Documents’, ‘Document\_Access’, ‘and Feedback’ are created.

For PostgreSQL in Windows, you need to do the follows to install and configure:

1), Download the PostgreSQL installer from the official site. This installer includes the PostgreSQL server, pgAdmin; a graphical tool for managing and developing your databases, and StackBuilder; a package manager that can be used to download and install additional PostgreSQL tools and drivers. Stackbuilder includes management, integration, migration, replication, geospatial, connectors and other tools.

2), Run the installer. It typically includes PostgreSQL server, pgAdmin (a graphical management tool), and optionally the Stack Builder for additional extensions like PostGIS.

3), Follow the installer instructions, set a password for the postgres user, and note down the port (default is 5432).

4), After installation, you can access PostgreSQL via pgAdmin or the command line.

If your PostgreSQL server is installed on the same machine where you are developing your Flask application, you typically use localhost as your hostname. This refers to the local machine. If the database is hosted on a remote server (e.g., a cloud database on AWS RDS, Google Cloud SQL, or another server), the hostname will be the URL or IP address provided by the hosting service.

3, User and document management system

We designed a detailed user and document management system with specific authentication, administration, and access control mechanisms. Here are the key components and functionalities:

**User Management:**

1), Registration and Authentication: Separate login interfaces for users and admins. Users can register with a username, account ID, and password, all of which must be unique.

2), Admin and User Login: A pre-set admin account with restricted login capabilities (no new admin registration). After login, users are directed to relevant interfaces based on their role (user or admin).

**Admin Features:**

1), Document Management: Administering document records, including viewing and deleting, with the ability to differentiate documents based on their origin (uploaded or scraped).

2), Feedback Management: Review and manage user feedback.

3), Q&A Record Management: Oversee and manage records of user queries and answers.

**Document Access Management:**

1), User-Specific Document Lists: Each user has a list showing document name, owner, and a status indicating whether the document can be used for querying.

2), Admin Document Controls: Admins view a similar list but with the ability to disable documents for all users.

3), Access Matrix Management: A visual interface for the admin to manage document access for each user. Admin can search for users and modify document access via a similar list where documents can be enabled or disabled.

**Technical Implementation Considerations:**

1), Documents can only be physically deleted from the database directly to minimize risk and error.

2), All interface actions reflect changes in an underlying "access matrix" that maps user permissions to documents.

We use Vue.js for frontend development, the following steps are in need to install that:

Step 1: Install Node.js

Vue CLI requires Node.js to run. Ensure that Node.js is installed on your system. You can download it from Node.js official website. This installation includes both Node.js and npm (node package manager).

Step 2: Install Vue CLI

Once Node.js is installed, you can install Vue CLI globally using npm. This allows you to run Vue commands from anywhere on your system. Open your command prompt or terminal and run the following command:

“npm install -g @vue/cli”

Step 3: Verify Installation

After installing, you can verify that Vue CLI is correctly installed by checking its version:

“vue --version”

This command should display the version of Vue CLI installed, indicating that the installation was successful.

When this front-end project will be transferred to another system, though all the files from the project directory are copied over, including the ‘node\_modules’ folder if you want to avoid having to reinstall all packages, it's often recommended to exclude this folder from transfers and rebuild it on the new machine. The reasons include that some npm packages include binaries compiled for a specific platform. Moving them to a different operating system (e.g., from Windows to macOS) can cause compatibility issues. The ‘node\_modules’ folder can be quite large and might take considerable time to copy. It’s often more efficient to regenerate this folder using ‘npm install’.

**Document uploads and web scraping:**

Upon entering the user dashboard page though login in, the users can upload .txt documents or extract the texts from web scraper if providing url, these texts are then used to build documents according to predefined format. There are also boxes, supporting content preview before conforming.

**Bonus Features:**

**(useful if you want to add more advanced content to enrich your thesis)**

1, Integrating Structured Data Capabilities

**Structured Data Indexing:**

1), Structured Data Setup: As described, using SQLAlchemy to manage structured data like a SQL database for city statistics can be paralleled in handling structured metadata about your text documents (e.g., uploader, upload date, document type). This structured information can be indexed and queried efficiently alongside the unstructured text embeddings.

2), Building Table Index: This concept can be adapted to create a hybrid index that includes both the full-text embeddings of documents and their structured metadata. This allows complex queries that consider both the semantic content of documents and their metadata attributes.

**Natural Language SQL Queries:**

1), Using NLSQL: The capability to convert natural language queries into SQL queries can greatly enhance the user experience by allowing users to make queries in natural language that are internally converted into SQL to fetch relevant metadata or document identifiers.

2), Query Engine for Structured and Unstructured Data: By setting up a query engine like NLSQLTableQueryEngine, you could create a similar engine that handles both types of data. Users could ask complex questions that span the document's semantic content and its metadata, such as asking for documents uploaded by a certain user within a specific time frame that also discuss a specific topic.

When a query comes in, use the structured data query engine to retrieve document metadata based on user-supplied criteria. Retrieve document embeddings from ChromaDB using the metadata results (e.g., vector IDs) and perform semantic searches to find the most relevant documents. For instance, the query string can be “query\_str = "Show documents uploaded by 'John Doe' in January 2021”

2, More details on registration, logging, authentications, and settings on the welcome page

This system is flexible enough to be extended with more features such as password recovery, multi-factor authentication. And also the session and cookie features for authentication (allow the password input appears only once to get credentials) requiring backend verification for state persistence, which is not core function for our project.

3, More details and requirements for documents savings

Simplest mechanism: If name of one document conflict with any of documents stored already in the database, the LLM needs to be used.

Webs can’t be accessed with proxies such as VPN, only domestic webs can be scraped under the condition without VPN, future version of app that can scrape international webs using proxies like SSR and VPN can be made by setting some configurations.

4, Now that only access is managed, more details on easy deletion directly on the web rather than do that manually in the database

5, Detailed user information management, such as to block or delete the user accounts.

6, Advanced features for RAG Q&A

To show the referred relevant chunks extracted, and to show their similarity at the same time.

7, Admin and users checking feedback and chat history

Admin can check feedback and chat history for all users, while a specific user can only be able to check their own feedback and chat history.