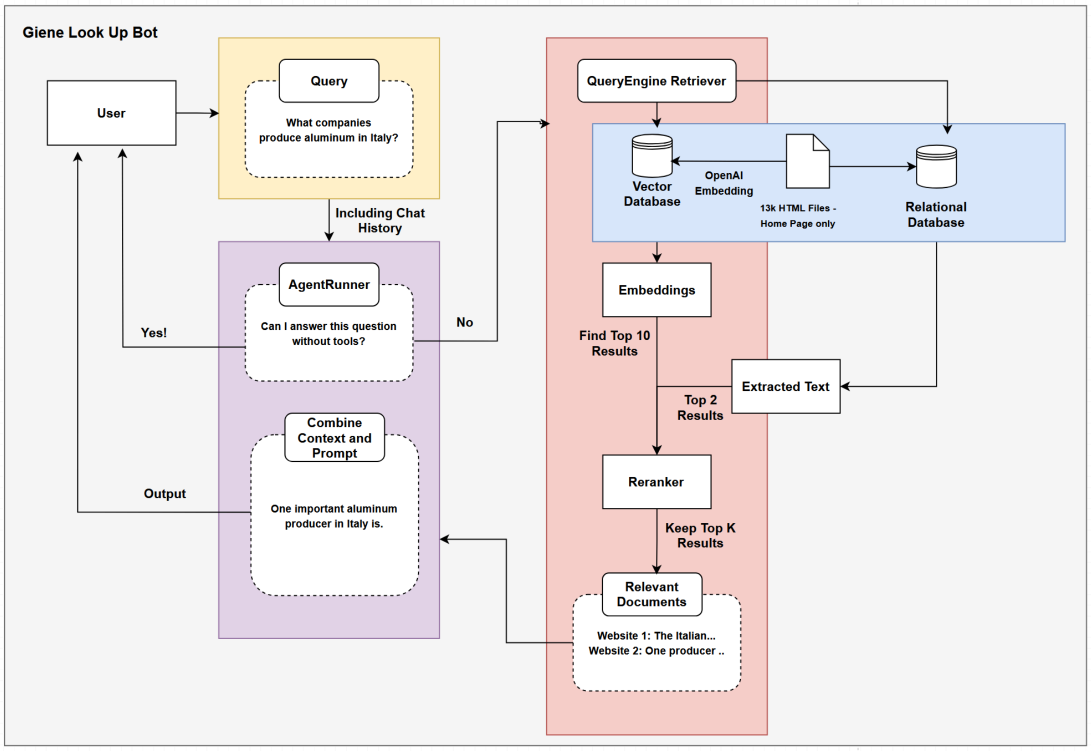
System Design



Overview:

Img 1: Agentic flow

Initial considerations:

- The task was to design a research tool for a supply chain director overseeing vendor relationships, logistics and procurement.
- Queries of the web pages therefore need to go both in depth (into subpages of the company url), as well as have a broad and general overview over the entire landscape.
- Dataset cleaning is important, we found redundant text and irrelevant chars.

1. Frontend Interface:
   Streamlit provides a simple, interactive user interface with sidebars for dynamic configuration (e.g., "Top k results" and "Max Tokens"). It handles user inputs, manages the chat history, and displays responses in a conversational format.

2. Backend Logic and Models:

Retrieval:

1) **Semantic Search** using llama_index integrated with *OpenAI's embeddings*. The embeddings offer a very rich representation, compared to smaller local models used for embeddings. Sample queries led to already great results. We were taking the budget into consideration, only using the landing page as sufficient representation for the information.
   Tradeoff: Rich embeddings, reduced information.
2) **Keyword Search** with SQLite FTS, offers efficient full text search based on keywords. The keywords are generated by 4o-mini. The search is fast and comprehensive.
   Tradeoff: No semantic search for larger dataset.

Knowledge Base Design

The webpages were cleaned before ingestion using dataset_filter().

- Vector DB: llama_index offers a VectorStoreIndex that can be persisted to storage. This proved to be quite flexible in deploying the solution. The vector embeddings are <2 GB
- Relational DB: The SQLite FTS is ~20 GB but has no big memory footprint when running.

Hybrid Search:

- The top-k results from both Semantic Search (k_1) and Keyword Search (k_2) are compiled to a list of intermediate search results. We chose hyperparameter by actively selecting a promising subset and validating those manually. We found that generally Semantic Search results are augmented by specific files from the Keyword Search. We therefore chose k_1 = 5 * k_2. This is another answer to our Initial considerations.

3. Reranker:
   Reranks results using an LLM-based Reranker to enhance response relevance. Deals with irrelevant, as well as redundant information in the intermediate results.

4. Output Generation:
   1. Temperature is 0 to limit hallucination & randomness in the output.
   2. Reasoning model is used to adequately address questions that rely on counting, thought chaining, etc.
   3. Prompt: We thoroughly tested the prompt and engineered it to encourage interactions and ask clarifying questions.
   4. Agent interactions: If the initial query is not precise enough to be adequately answered with the knowledge base, the agent asks to clarify the request.

Future Ideas

Some ideas that did not make it into the MVP (yet)

1. **Active Fine-Tuning**
   To better address the relevance problem, one can use test time fine tuning to optimize information gain about the query rather then naive similarity search. This automatically identifies relevant information and reduces uncertainty in the intermediate results. (see https://jonhue.github.io/activeft/). It's provably better then top-k results.
   This approach was unfortunately abandoned. We found out that we are limited by compute and storage. It was not possible to deploy this solution to the size of this problem, while maintaining sufficiently rich text embeddings. With more compute, one could easily improve Retrieval of Semantic Search.
   Tradeoff: Research backed, but compute intensive. - not employed
2. **More Semantic Search**
   using a *small local model* (multilingual-e5-small) instead of the large openai embeddings, offer great speeds for average results (impressive for the size though). We compared this approach to the one in production and found that further tuning will be necessary to be competitive with openai's rich embeddings. This option could also be combined with idea 1.
   Tradeoff: All wep pages, but less accurate search. - not employed
3. **Metadata extraction**
   Utilize structured LLMs to extract meaningful features from webpages. Employed with fast relational database. Integrate with Hybrid Search or emply different modes of RAG.