Project 1: Q&A RAG-Enabled Chatbot

Introduction

Within recent years, the field of software development has evolved rapidly with the surge in popularity of chatbot applications that use generative AI. One advancement in this space is Retrieval-Augmented Generation (RAG), methodology that incorporates the generative capabilities of LLMs with an external data retrieval mechanism, providing much more enhanced responses that include the retrieved data which is concurrent, accurate and business domain-specific. This project will focus on implementing a RAG-powered Q&A system (using HuggingFace API, Pinecone Library, Langchain Library and Streamlit), which will allow users to get answers that are relevant, authoritative and based on their private document collection. A textbook on biochemistry and cell biology genetics will be used as a knowledge base retrieved from the Open Textbook Library. The potential use case for this project would be that the chatbot enabled with RAG methodology would allow students to easily access the accurate information about particular cell genetics topic and study more effectively.

Course Relevancy

This project is related to several topics covered in the course including the use of diverse and amazing Python-based libraries and tools such as LangChain, Pinecone, HuggingFace API and Streamlit to build powerful Python applications. Agile methodology will be followed during the development process via version control using Github and performing testing and iterative improvements. The application will be broken down into modular components, such as data processing, prompt engineering, retrieval and indexing pipeline. Thus, this project provides experience with Python-based software development, various AI tools and also exposes to best software engineering practices.

2. Overview of Retrieval-Augmented Generation (RAG)

This project will aim to develop a Q&A system that uses Retrieval-Augmented Generation (RAG) to extract information specific to private documents collection as opposed to traditional LLM systems, which generate responses based on the data that was provided during training. RAG will first direct the model to query external data that was previously converted into vector representations by an embedding language model and stored in a vector database. When user submits a query, it will be converted into a vector representation which will be matched against the stored vectors in the database to find the most relevant information. The user prompt can then be augmented with the newly found retrieved information and additional prompt engineering (AWS, 2024).

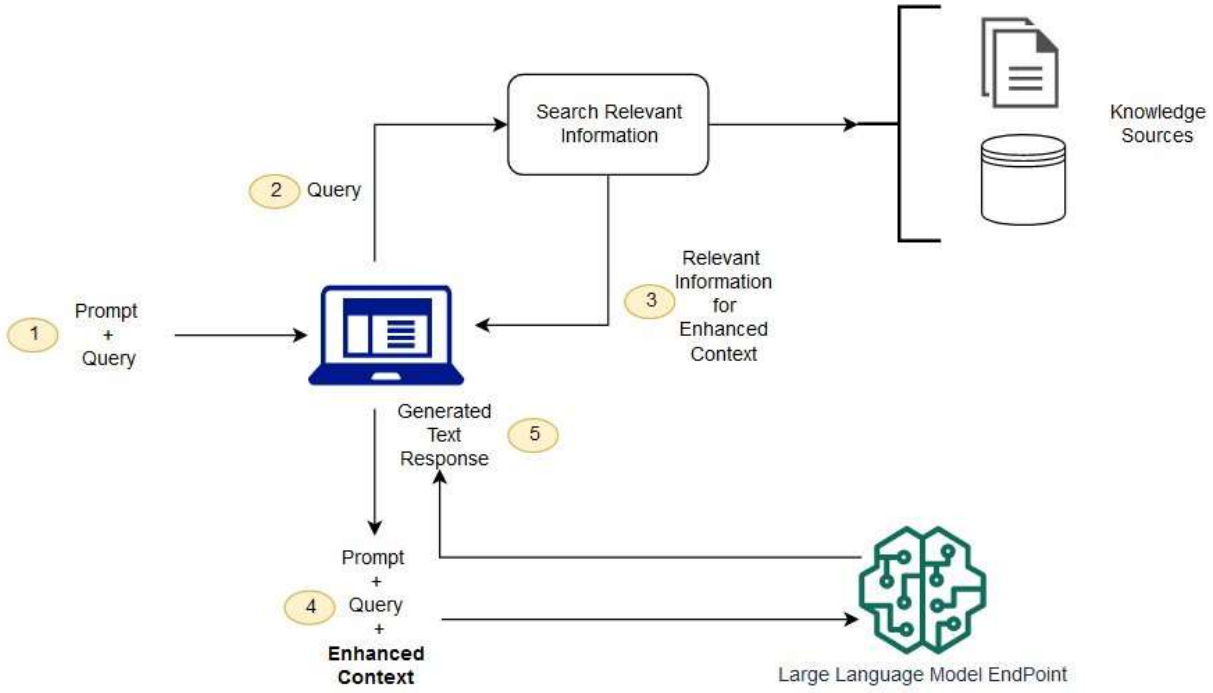
3. Tools/Libraries

- LangChain, an open-source framework which provides structured pipelines for building LLM-powered apps.
- Pinecone, a cloud-native vector database that stores vector embeddings for fast retrieval and similarity search.
- Microsoft's Phi-4 model, a text-generation model from HuggingFace
- Hugging Face Embedding Models, language models that convert data into numerical vector representations
- Streamlit, an open-source Python library that allows to create and share custom web apps for machine learning and data science
- Jupyter Notebook, web-based IDE that will be used for testing and experimentation during the project development before the final implementation
- Knowledge base source is a textbook on biochemistry and cell biology genetics found here: https://open.umn.edu/opentextbooks/textbooks/cell-biology-genetics-and-biochemistry-for-preclinical-students

4. RAG System Implementation

Retrieval Augmented Generation architecture follows the following stages.

Figure 1. *RAG System Conceptual Flow.*



Note. Image retrieved from Amazon AWS, 2024 (https://aws.amazon.com/what-is/retrieval-augmented-generation/)

Creation of External Data Source

The process begins with identifying and collecting external data required for the system, such as data from APIs, databases, documents and files. Large documents are segmented into smaller chunks so that each chunk is contextually coherent to make retrieval more accurate during query processing. Embedding models such as those found on HuggingFace are used to convert each chunk into a vector representation based on their semantics. The generated vectors are then stored in vector databases for similarity search and fast information retrieval (AWS, 2024).

Document Digestion

In this project, one large pdf textbook on biochemistry and cell genetics was loaded using **PyPDFLoader** from Langchain's community document loader, where the loader reads the PDF file and extracts its contents. Multiple files can be loaded using **DirectoryLoader** from a specified directory. This is achieved with **load** method that extracts the contents into a list of document objects.

```
from langchain_community.document_loaders import PyPDFLoader,
DirectoryLoader

def load_pdf_file(data):
    loader = DirectoryLoader(data, glob="*.pdf", loader_cls=PyPDFLoader)
    documents = loader.load()
    return documents
```

Text Chunking

The extracted data is then split into smaller chunks of up to 500 characters using **RecursiveCharacterTextSplitter**, while maintaining the coherence by overlapping 20 characters between the chunks (Langchain, 2024).

```
from langchain.text_splitter import RecursiveCharacterTextSplitter

def text_split(extracted_data):
    text_splitter = RecursiveCharacterTextSplitter(chunk_size=500,
chunk_overlap=20)
    text_chunks = text_splitter.split_documents(extracted_data)
    return text_chunks
```

Embedding Generation

Text chunks were converted into vector embeddings using HuggingFace's **sentence-transformers/all-MiniLM-L6-v2** embedding model (HuggingFace, 2025). The HuggingFaceEmbeddings class then loads the model which maps sentences to a 384-dimensional dense vector space (Langchain Sentence Transformers, 2025).

```
from langchain_community.embeddings import HuggingFaceEmbeddings

def download_hugging_face_embeddings():
    embeddings = HuggingFaceEmbeddings(model_name='sentence-
transformers/all-MiniLM-L6-v2')
    return embeddings
```

Vector Storage

Embeddings can be then upserted in Pinecone's index for similarity search (Pinecone Notebooks, 2025).

```
from pinecone import Pinecone, ServerlessSpec
from langchain_pinecone import PineconeVectorStore

def store_embeddings_in_pinecone(text_chunks, embeddings, index_name,
api_key):
    pc = Pinecone(api_key=api_key)
    pc.create_index(
        name=index_name,
        dimension=384,
        metric="cosine",
        spec=ServerlessSpec(cloud="aws", region="us-east-1")
    )
    docsearch = PineconeVectorStore.from_documents(
        documents=text_chunks,
```

```
        index_name=index_name,
        embedding=embeddings,
    )
    return docsearch
```

Query Processing

When user submits a query, it is transformed into a vector representation which is compared and matched semantically against the stored vectors in the database to retrieve most relevant documents. Here, a similarity_**search** method retrieves the top 10 documents similar to the user's question, where their contents are combined into the context for response generation (Langchain Vector stores, 2025).

```
def retrieval_qa_chain(question, docsearch, client):
    retrieved_docs = docsearch.similarity_search(question, k=10)
    context = " ".join([doc.page_content for doc in retrieved_docs])
    answer = chat_with_qwen(question, context, client)
    return {"question": question, "answer": answer, "source_documents":
retrieved_docs}
```

LLM Prompt Augmentation and Response Generation

The user input is then augmented with retrieved information context. The query can further be guided with a system prompt engineering that allows the LLM to generate more targeted and accurate answers (HuggingFace Phi-4, 2025).

```
from huggingface_hub import InferenceClient


def chat_with_qwen(question, context, client):
  messages = [
    {"role": "system", "content": prompt_template},
    {"role": "user", "content": f"Context: {context}\n\nQuestion: {question}"}
  ]
```

```
completion = client.chat.completions.create(
    model=" microsoft/phi-4",
    messages=messages,
    max_tokens=500,
)
return completion.choices[0].message["content"]
```

Updates of Vector Database

To keep the LLM responses relevant and up-to-date, periodic batch processing is required. This is enabled by running "python store_index.py" which creates index and upserts document embeddings in Pinecone (AWS, 2024).

Comparison with Alternative Approaches

There are various implementation approaches at each stage of RAG system development from document ingestion, text splitting, embedding generation and vector storage to query processing and response generation.

Comparison of LLM App Building Frameworks

LangChain

LangChain is a Python-based library that facilitates deployment of LLMs for building NLP applications such as question-answering systems. It offers text-generation and question-answering, broad support for multiple LLMs and is optimal for constructing chatbots that are capable of answering queries on a particular domain and need to deliver precise and relavtn responses. It is also useful for producing concise summaries of lengthy documents (Ahmad, 2023). However, users have reported challenges with its complexity and debugging difficulties due to abstractions and performance issues in production environments (Vasilis, 2025). From personal experience, while LangChain provides helpful abstractions, these layers add some complexity as most LLM applications just require string handling, API calls and loops and vector database if RAG is used.

LlamaIndex

LlamaIndex, formerly known as GPT-Index, is an open-source data orchestration framework designed for simpler integration of data sources with LLMs. It offers tools for data ingestion, indexing and querying which is helpful for building context-augmented AI apps. LlamaIndex also allows for topic extraction from unstructured data and for creation of networks of RAG systems (Ahmad, 2023). However, there are some downsides, such as the challenge of handling large volumes of data as it may face difficulties in quickly indexing the extensive datasets. Moreover, it requires careful configuration and continuous optimization to return most relevant results based on specific queries (McLane, 2023).

Haystack

Haystack, is an open-source framework for building production-ready LLM applications, RAG pipelines and advanced search systems. It allows developer to create customized workflows with its modular architecture by connecting components such as document stores, retrievers and readers. It supports integration with transformer models for tasks such as document retrieval and question answering and is suitable for both small scale and large scale deployments (Haystack, 2025). Though, there are several challenges and limitations such as Haystack's reliance on large pre-trained models which leads to significant resource consumption and requirement for powerful GPUs. There is also complexity in customization as developers need to write a lot of custom code to fit the pipelines for specific requirements, hence increasing time and complexity (Zilliz, 2025).

Comparison of Vector Databases

Pinecone

Pinecone is a fully managed, serverless vector database that is optimized for fast and scalable similarity searches. It provides real-time indexing and querying, automatic scaling and is optimized for Approximate Nearest Neighbour Search for high dimensional vectors. Most common use cases are sematic search to understand context and meaning and for recommendation systems. Its limitations are that it has higher operational costs (Ozkaya, 2024). From personal experience, Pinecone's SDK is quite straightforward, and the search and retrieval is quite fast.

Chroma

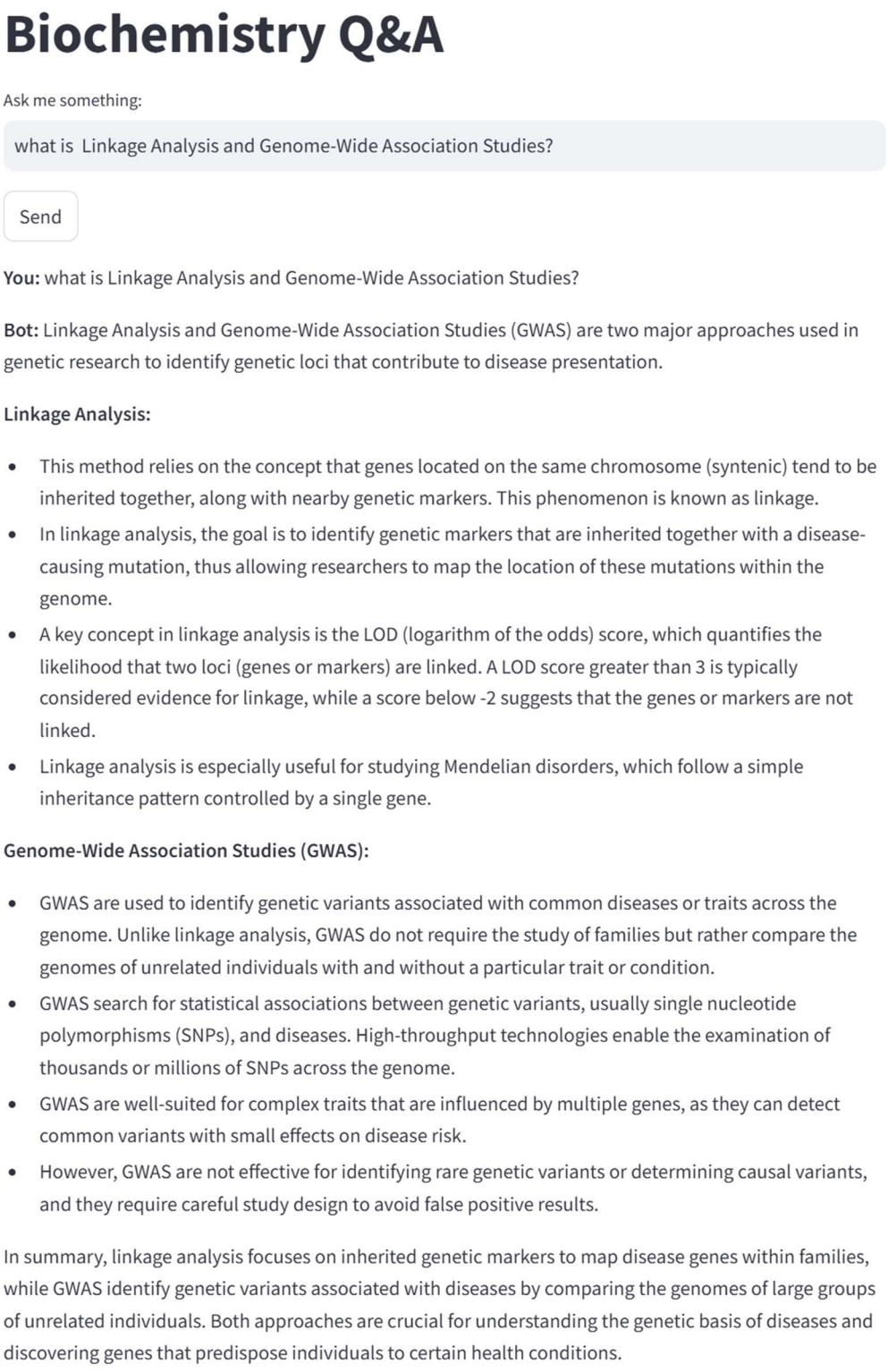
Chroma is an open-source vector database which provides ease of use, simplicity and is highly customizable, as well as suitable for applications with smaller datasets. Its limitations are the scalability constraints as Chroma's single-node architecture limits it from scaling efficiently (up to 1 million vector points). Chroma also may lack some advanced features such as in other databases such as extensive indexing options or distributed data management (Feng, 2024).

Milvus

Milvus is an open-source vector database designed for large-scale and high-performance applications. It is optimized for low-latency and high-throughout vector searched and supports elastic and horizontal scaling (billions of vector points). Milvus also offers support for various indexing algorithms, tunable consistency and multi-language SDK. However, it comes at a cost of being more resource intensive and having more complex deployment and management due to extensive features and distributed architecture (Feng, 2024).

Using Q&A Chatbot

Figure 2. Question-Answering Chatbot Demonstration



The developed application interface and the web app deployment was enabled by Streamlit, an open-source Python library that allows to create and share custom web apps. Microsoft's text-generation model phi-4 was used which is built upon the blend of synthetic datasets and data filtered from public domain websites and acquired academic books and Q&A datasets with goal of providing high-quality answers and advanced reasoning (HuggingFace Phi-4, 2025). The response in Figure 2 was generated after supplying a query that contains both a system prompt as a guidance for the model to answer the question, the user's query as well as the context from the documents retrieved from Pinecone database which can be viewed in the console window in Figure 3.

Figure 3.1 Retrieved Context from Vectors Stored in Pinecone database

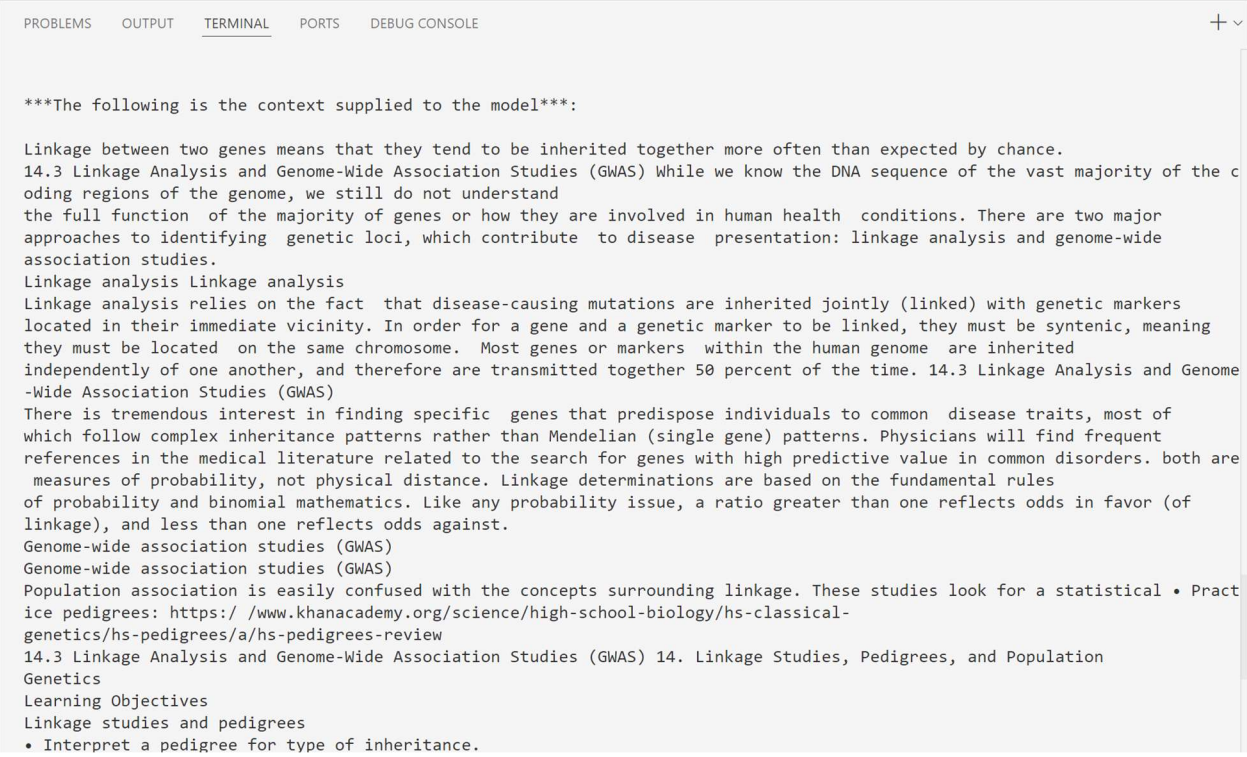


Figure 3.2 Retrieved Context from Vectors Stored in Pinecone database (Continued)

```
Linkage studies and pedigrees
• Interpret a pedigree for type of inheritance.
• Describe how Mendelian disease genes are identified by linkage mapping.
Population genetics
• Perform simple Hardy-Weinberg calculations for autosomal recessive disease.
• Recognize populations that are not in Hardy-Weinberg equilibrium.
• Understand the following terms related to genetic variation: polymorphism, SNP, haplotype, HapMap. appear in the medical li
terature. This is a highly sophisticated type of case-control study for which careful study design
is required to avoid spurious findings. These studies provide information related to common genetic traits but do not
help address genetic manifestations of rare traits in a population (figure 14.7).
14.3 Linkage Analysis and Genome-Wide Association Studies (GWAS) When markers are close enough together on the same chromosom
e, crossing over fails to separate them frequently
enough for them to be inherited independently of one another. This is evidenced by coinheritance of greater than 50
percent.
The unit of measure in linkage studies is “centimorgans.” This concept can be confusing because we refer to the
“distance” between two traits, but what is measured experimentally is the frequency of coinheritance, not physical
distance.
```

As we can see from Figure 4 illustrating the text from the textbook about Linkage Analysis and Genome-Wide Association Studies, the context that was retrieved was accurate and relevant to the question asked by the user, as well as the model response incorporated the information from the context and accurately answered the question.

Figure 4. Textbook Text on Linkage Analysis and Genome-Wide Association Studies

14.3 Linkage Analysis and Genome-Wide Association Studies (GWAS)

There is tremendous interest in finding specific genes that predispose individuals to common disease traits, most of which follow complex inheritance patterns rather than Mendelian (single gene) patterns. Physicians will find frequent references in the medical literature related to the search for genes with high predictive value in common disorders.

While we know the DNA sequence of the vast majority of the coding regions of the genome, we still do not understand the full function of the majority of genes or how they are involved in human health conditions. There are two major approaches to identifying genetic loci, which contribute to disease presentation: linkage analysis and genome-wide association studies.

Linkage analysis

Linkage analysis relies on the fact that disease-causing mutations are inherited jointly (linked) with genetic markers located in their immediate vicinity. In order for a gene and a genetic marker to be linked, they must be syntenic, meaning they must be located on the same chromosome. Most genes or markers within the human genome are inherited independently of one another, and therefore are transmitted together 50 percent of the time.

Linkage between two genes means that they tend to be inherited together more often than expected by chance.

Conclusion

This project showcases an innovative implementation of Retrieval-Augmented Generation methodology for Q&A on private documents using Pinecone vector database, Langchain framework and HuggingFace Microsoft Phi-4 model. By integrating both advanced retrieval techniques with large language models and efficient prompt engineering, the system allows for precise knowledge retrieval, accuracy and ease of scalability. It provides an opportunity to explore and utilize various Python tools and libraries in conversational AI systems development showcasing how LLM-powered applications can be useful in a specific domain area.

App Link

https://appragchatbot-rwedmfwycmrvsun8yz4eiv.streamlit.app/

GitHub Link

https://github.com/btp405-w25/project-1-kolossi101/tree/main

References

Alammar, J., & Grootendorst, M. (n.d.). Chapter 7: Advanced Text Generation Techniques and Tools. In Hands-On Large Language Models. O'Reilly Media, Inc. Retrieved from https://learning.oreilly.com/library/view/hands-on-large-language/9781098150952/ch07.html

Pinecone. (2025). Build a RAG chatbot. Pinecone Documentation. Retrieved from https://docs.pinecone.io/guides/get-started/build-a-rag-chatbot

Ahmad, S. H. (2023, May 19). LangChain vs LlamaIndex vs Haystack vs Hugging Face. FAUN — Developer Community. https://faun.pub/langchain-vs-llamaindex-vs-haystack-vs-hugging-face-3c512ae3b07f

Vasilis, T. (2025, March 11). 12 open-source LangChain alternatives. APIFY. https://blog.apify.com/langchain-alternatives/

McLane, B. (2023, November 1). What is LlamaIndex? Exploring LLM orchestration frameworks. DataStax. https://www.datastax.com/guides/what-is-llamaindex

Deepset. (2025). What is Haystack? Haystack. https://haystack.deepset.ai/overview/intro

Zilliz. (n.d.). What are the limitations of Haystack in large-scale NLP applications? Zilliz. https://zilliz.com/ai-faq/what-are-the-limitations-of-haystack-in-largescale-nlp-applications

Ozkaya, M. (2024, December 3). Exploring vector databases: Pinecone, Chroma, Weaviate, Qdrant, Milvus, PgVector, and Redis. Medium. https://mehmetozkaya.medium.com/exploring-vector-databases-pinecone-chroma-weaviate-qdrant-milvus-pgvector-and-redis-f0618fe9e92d

Feng, F. (2024, February 5). Choosing a vector database: Milvus vs. Chroma DB. Zilliz. https://zilliz.com/blog/milvus-vs-chroma

Amazon AWS. (2024). Retrieval Augmented generation. https://aws.amazon.com/what-is/retrieval-augmented-generation/

Langchain. (2024). Retrieval. Textsplitters. Recursively split by character. https://python.langchain.com/v0.1/docs/modules/data_connection/document_transformers/recursive_text_splitter/

HuggingFace. (2025). Sentence Transformers: all-MiniLM-L6-v2. https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2

Langchain. (2025). Sentence Transformers on Hugging Face. https://python.langchain.com/docs/integrations/text_embedding/sentence_transformers/

Pinecone. (2025). Pinecone docs notebooks. https://docs.pinecone.io/examples/notebooks

Langchain. (2025). Vector stores. https://python.langchain.com/v0.1/docs/modules/data_connection/vectorstores/

HuggingFace. (2025). Microsoft Phi-4. https://huggingface.co/microsoft/phi-4

Appendix

Chatbot Application Setup

This document provides instructions on how to set up and run the q&a chatbot application.

Setup Instructions

1. **Create a Virtual Environment:**

   Open your terminal or command prompt and navigate to the project directory. Run the following command to create a virtual environment named chatbot:

   python -m venv chatbot

2. **Select Python Interpreter in VS Code:**

   - Open the project in VS Code.
   - Press Ctrl+Shift+P (or Cmd+Shift+P on macOS) to open the command palette.
   - Type "Python: Select Interpreter" and press Enter.
   - Select the Python interpreter from the chatbot virtual environment. It will typically be located at .\chatbot\Scripts\python.exe (Windows) or ./chatbot/bin/python (macOS/Linux).

3. **Bypass Execution Policy (Windows Only):**

   If you are using Windows, you might need to bypass the execution policy to run the activation script. Open PowerShell as an administrator and run:

   Set-ExecutionPolicy -Scope Process -ExecutionPolicy Bypass

   Note: This command temporarily bypasses the execution policy for the current process. It is generally safe for setting up virtual environments.

4. **Activate the Virtual Environment:**

   - **Windows:**

     .\chatbot\Scripts\activate

- **macOS/Linux:**

  source chatbot/bin/activate

5. Your terminal prompt should now indicate that the virtual environment is active (e.g., (chatbot) C:\path\to\project>).
6. **Install Dependencies:**

   Install the required Python packages using pip from the requirements.txt file:

   pip install -r requirements.txt

7. **(Optional) Recreate Pinecone Index:**

   If you want to create a new Pinecone index (it is already created), run the following command:

   python store_index.py

Important: This step requires that you have your Pinecone API key and environment variables properly set up. Also, make sure to change the name of the variable "index_name" for your new index.

8. **Run the Application:**

   Start the Streamlit application by running:

   streamlit run app.py

This will open the application in your default web browser. Click the following link to view the deployed app: https://appragchatbot-rwedmfwycmrvsun8yz4eiv.streamlit.app/

Notes

- Pinecone API key and HuggingFace API key will be provided on Blackboard as .env file.