# LLM-OPS TRADING BOT

## AI-POWERED FINANCIAL INSIGHTS: BUILDING AN INTELLIGENT TRADING BOT WITH LARGE LANGUAGE MODELS

Github-Link

# INTRODUCTION

- Project Overview:
- This project aims to develop a trading bot that leverages Large Language Models (LLMs) to analyze financial data, provide insights, and automate trading decisions.
- Purpose:
- To enhance decision-making in financial markets using AI and facilitate automated trading.
- Motivation:
- The desire to explore AI's potential in finance and create a tool that can assist users in navigating complex financial information.

# PROJECT GOALS AND OBJECTIVES

- Primary Goals:
- Build a system capable of processing and analyzing large volumes of financial data.
- Enable users to ask natural language questions and receive relevant financial advice.
- Automate trading strategies based on generated insights.
- Specific Objectives:
- Implement a robust data ingestion pipeline.
- Develop a retrieval-augmented generation (RAG) pipeline for information retrieval and response generation.
- Ensure secure and efficient storage and retrieval of data.

# PROJECT WORKFLOW

1. Data Ingestion:
   - Loading and processing financial documents (e.g., PDFs) containing market data, reports, and analyses.
2. Embeddings and Storage:
   - Converting documents into vector representations using OpenAI's embeddings.
   - Storing these vectors in AstraDB for efficient similarity search.
3. Retrieval:
   - Retrieving relevant information based on user queries using similarity search.
4. Response Generation:
   - Generating responses using a chat model, providing concise and informative financial advice.

# DATA INGESTION AND STORAGE

- Data Source and Collection:
- Financial data collected from various sources, such as reports, articles, and market analyses.
- Example data file: finance_data.pdf.
- Data Processing Pipeline:
- Loading:
  - PDFs are loaded using PyPDFLoader.
- Text Extraction:
  - Extract text from each page and concatenate to form a raw text corpus.
- Text Splitting:
  - Split the raw text into chunks using RecursiveCharacterTextSplitter to create manageable pieces for embedding

Vector Storage:

- AstraDB:
  - A scalable vector store that supports efficient storage and retrieval of document vectors.
- Document Embeddings:
  - Generated using OpenAI's embeddings API, converting textual data into vector representations.

# CODE EXAMPLE: DATA INGESTION

```python
from langchain_astradb import AstraDBVectorStore
from langchain_openai import OpenAIEmbeddings
from tradingbot.helper import load_file
import os

# Load environment variables
load_dotenv()
OPENAI_API_KEY = os.getenv("OPENAI_API_KEY")
ASTRA_DB_API_ENDPOINT = os.getenv("ASTRA_DB_API_ENDPOINT")
ASTRA_DB_APPLICATION_TOKEN = os.getenv("ASTRA_DB_APPLICATION_TOKEN")
ASTRA_DB_KEYSPACE = os.getenv("ASTRA_DB_KEYSPACE")

# Initialize embeddings and vector store
embedding = OpenAIEmbeddings(api_key=OPENAI_API_KEY)
```

```python
embedding = OpenAIEmbeddings(api_key=OPENAI_API_KEY)

def ingestdata(status):
    vstore = AstraDBVectorStore(
        embedding=embedding,
        collection_name="financebot",
        api_endpoint=ASTRA_DB_API_ENDPOINT,
        token=ASTRA_DB_APPLICATION_TOKEN,
        namespace=ASTRA_DB_KEYSPACE,
    )
    if status is None:
        docs = load_file()
        inserted_ids = vstore.add_documents(docs)
        return vstore, inserted_ids
    return vstore
```

Explanation:

- This code initializes the vector store with embeddings and manages the ingestion of documents. It stores vector representations in AstraDB for efficient retrieval.

# TECH STACK

- Languages and Frameworks:
- Python: Core language for developing the bot and handling data processing.
- Jupyter Notebook: Used for prototyping and testing.
- CSS and HTML: For frontend design if applicable.
- Libraries and Tools:
- langchain: For LLM and chatbot functionalities.
- OpenAI: Provides embeddings and chat models.
- AstraDB: Vector database for storing document embeddings.
- dotenv: For managing environment variables securely.

# RETRIEVAL AND GENERATION

- Retrieval Process:
- User queries are processed, and relevant documents are retrieved using a similarity search mechanism based on vector representations.
- Prompt Engineering:
- A prompt template is designed to guide the LLM in generating appropriate responses. For instance:

```
PRODUCT_BOT_TEMPLATE = """
Your finance bot is an expert in finance-related advice.
Ensure your answers are relevant to the query context and refrain from straying off-to
Your responses should be concise and informative.

CONTEXT:
{context}

QUESTION: {question}

YOUR ANSWER:
```

Response Generation:

- The retrieved context is fed into the LLM, which generates a response based on the provided template and user query.

# CODE EXAMPLE: RETRIEVAL AND GENERATION

```python
from langchain_core.output_parsers import StrOutputParser
from langchain_core.prompts import ChatPromptTemplate
from langchain_core.runnables import RunnablePassthrough
from langchain_openai import ChatOpenAI
from tradingbot.data_ingestion import ingestdata

def generation(vstore):
    retriever = vstore.as_retriever(search_kwargs={"k": 3})
    prompt = ChatPromptTemplate.from_template(PRODUCT_BOT_TEMPLATE)
    llm = ChatOpenAI()
    chain = ({"context": retriever, "question": RunnablePassthrough()} | prompt | llm | St
    return chain

if __name__ == '__main__':
    vstore, _ = ingestdata(None)
    chain = generation(vstore)
    print(chain.invoke("What is Market For Registrant’s Common Equity?"))
```

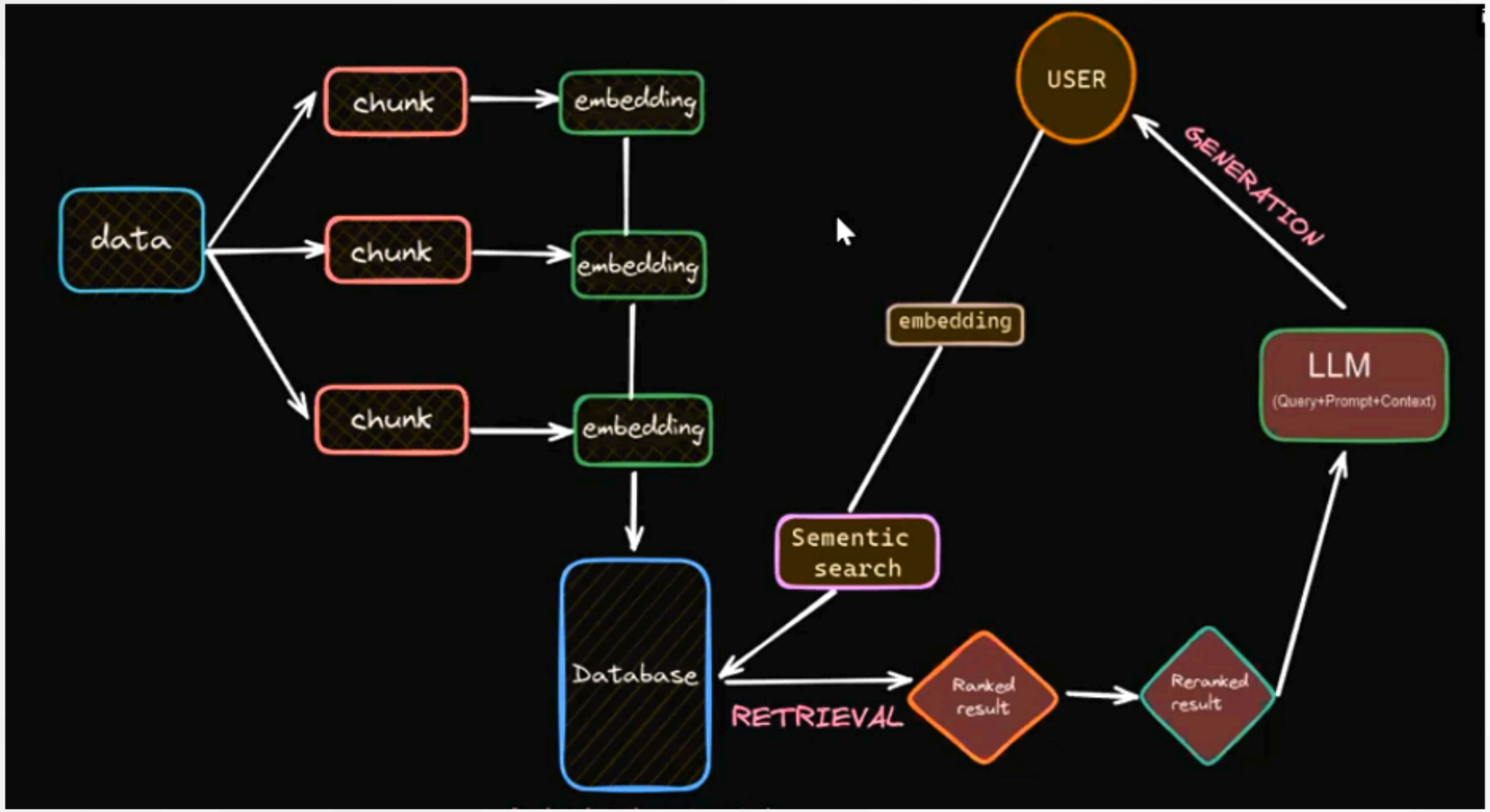
Explanation:

- This code demonstrates the end-to-end flow of retrieving relevant data and generating responses. The generation function sets up the prompt and handles the interaction with the LLM.

# RESULTS AND ACHIEVEMENTS

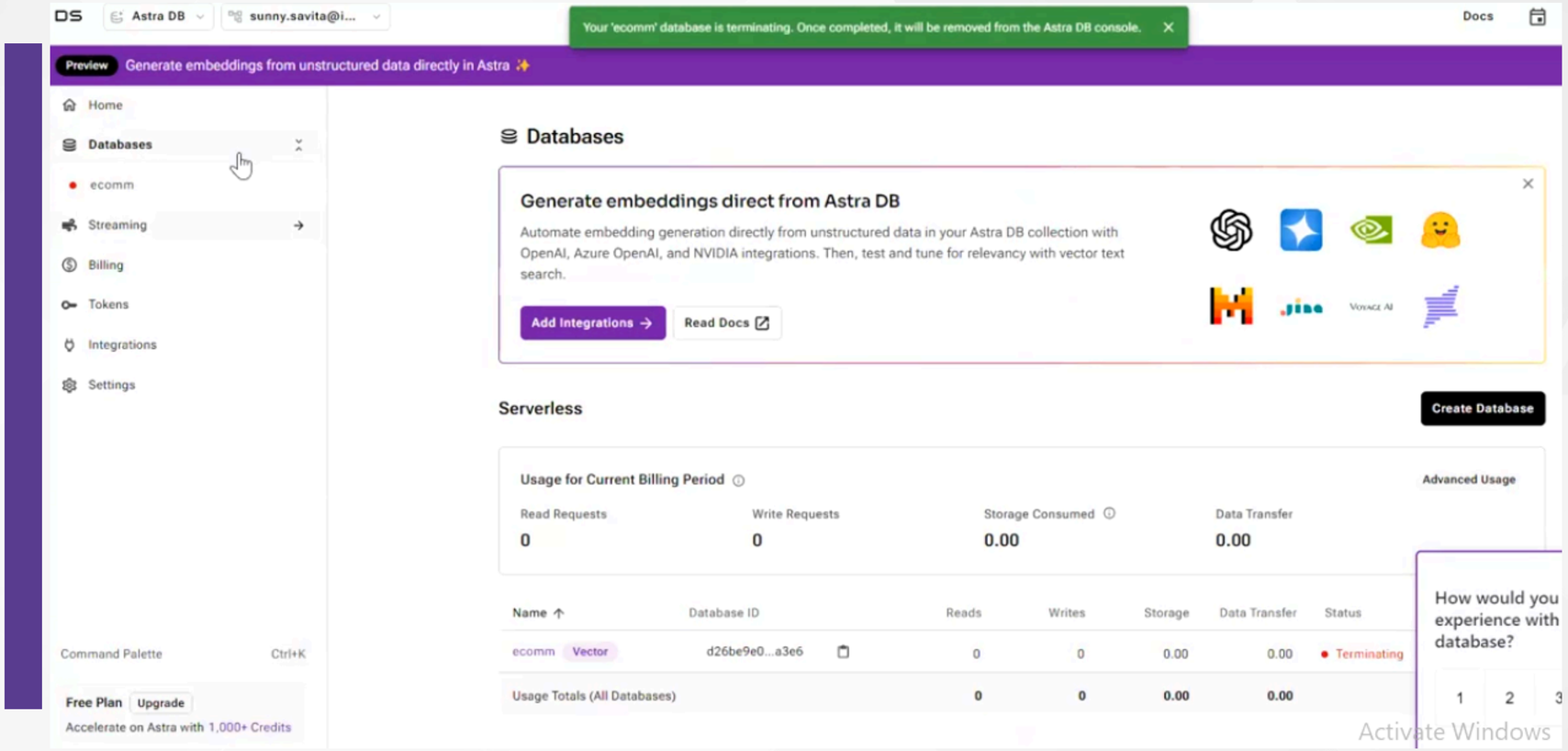
- Performance Metrics:
- Accuracy: High accuracy in retrieving relevant information and generating correct responses.
- Efficiency: Quick response time and efficient handling of large datasets.
- User Experience: Positive feedback on the bot's usability and the quality of advice provided.
- Key Achievements:
- Successful implementation of a functional trading bot.
- Development of a robust data ingestion and retrieval system.
- Demonstrated the potential of LLMs in the finance domain.

# DATA PROCESSING AND RESPONSE GENERATION WORKFLOW IN LLM-OPS TRADING BOT




This diagram illustrates the workflow of the LLM-Ops Trading Bot. It shows the process of data chunking, embedding generation, storage in a database, retrieval via semantic search, and response generation using a large language model (LLM) based on user queries

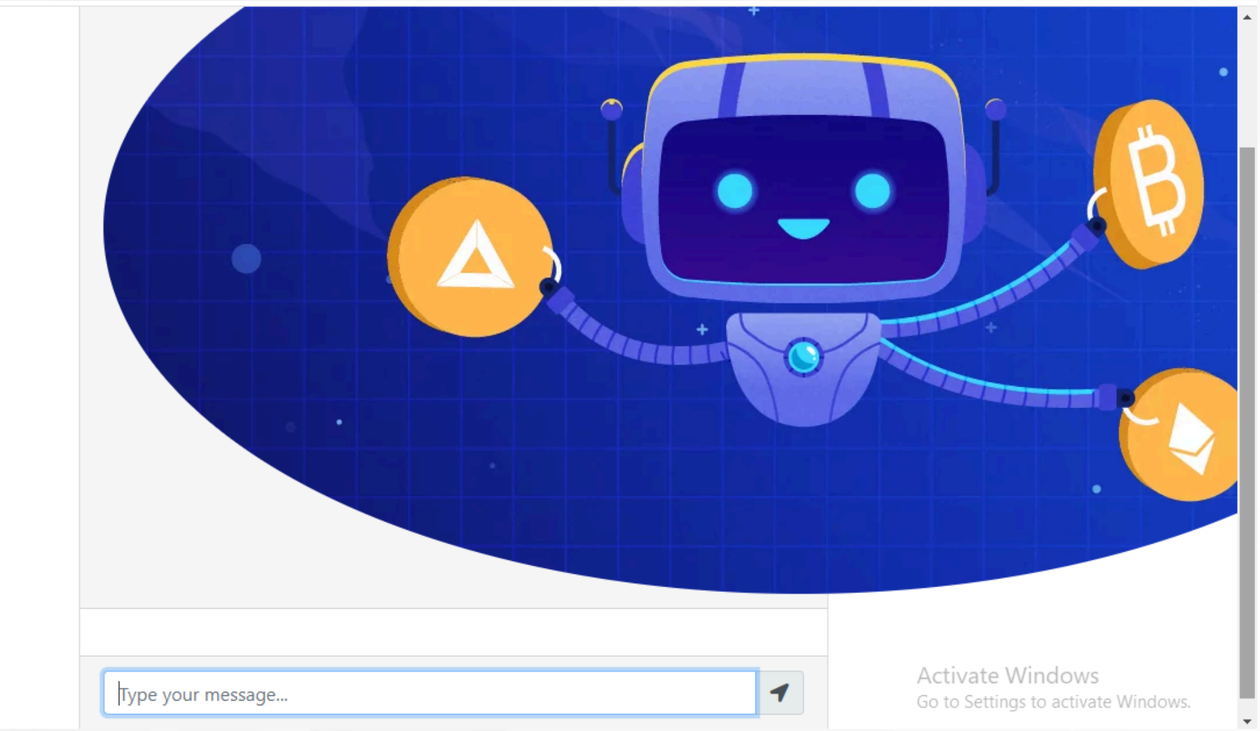
# ASTRA DB CONSOLE




here we generate --->>>

```
ASTRA_DB_API_ENDPOINT=os.getenv("ASTRA_DB_API_ENDPOINT")
ASTRA_DB_APPLICATION_TOKEN=os.getenv("ASTRA_DB_APPLICATION_TOKEN")
ASTRA_DB_KEYSPACE=os.getenv("ASTRA_DB_KEYSPACE")
```

# RESULTS





this my end result ,,my chat bot is built and the conversation is done with it