AI635-Multi-Agent Systems & Game Theory
Assignment 01

Miltiades Karras
GRN 2024EUC4019

RAG System Report: AI-Powered Q&A on the Rules of a Robotics Competition

Introduction

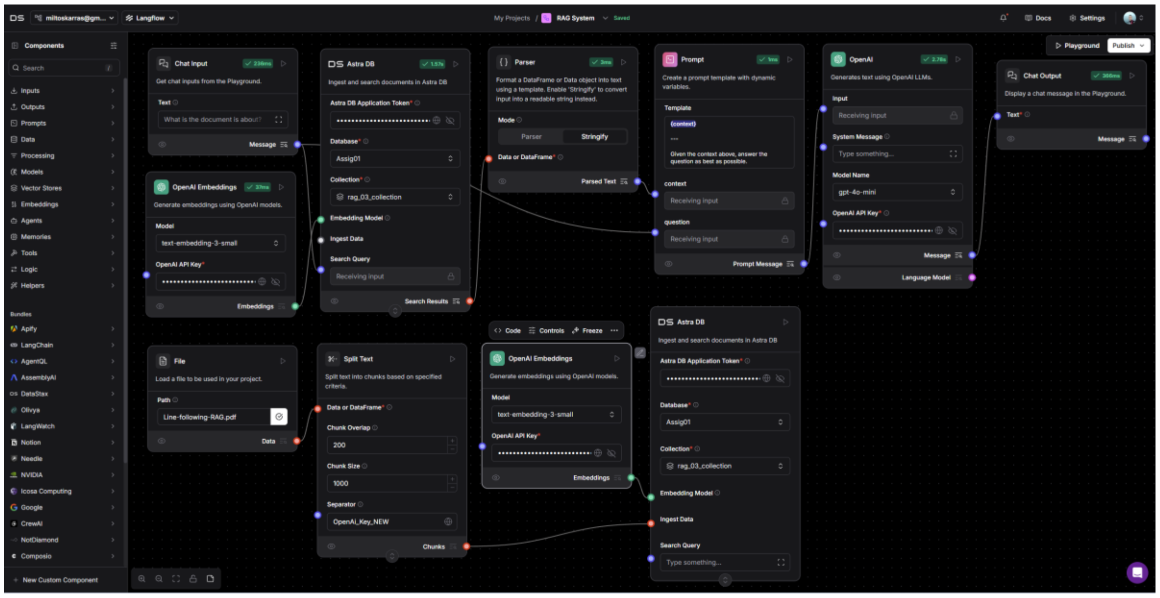
For this Assignment I will describe the development of an interactive AI-powered Retrieval-Augmented Generation (RAG) system using Langflow with the Astra DB document database. As the Head of the Technical Committee of the Robotex Robotics Competition in Cyprus, my goal was to enable the participants of the competition to query a knowledge base with the rules of this robotics competition and receive accurate and relevant answers powered by OpenAI's GPT-4o mini model. On a second level, this could be implemented on our organization's website.

System Architecture & Approach

For building this system I integrated Langflow's online visual interface with DataStax Astra DB as the vector database and I used the OpenAI GPT-4o mini LLM (Large Language Model) for generating responses. The original document was first converted to .pdf format and converted into structured chunks using the 'SplitText' component and was embedded using OpenAI's embedding model. These embeddings were then stored in Astra DB for semantic search. When the system receives a user query, it retrieves the relevant chunks and passes both the query and the context into a prompt template. The prompt is processed by GPT-4o mini to generate a precise and context-aware response.

Langflow Workflow Overview

Below are screenshots from the Langflow visual interface with brief descriptions of each step in the pipeline:



Chat Input — Get chat inputs from the Playground.
Text: What is the document is about?
Message

OpenAI Embeddings — Generate embeddings using OpenAI models.
Model: text-embedding-3-small
OpenAI API Key*: ••••••••••••••••••••••••
Embeddings

Astra DB — Ingest and search documents in Astra DB
Astra DB Application Token*: ••••••••••••••••••••••••
Database*: Assig01
Collection*: rag_03_collection
Embedding Model
Ingest Data
Search Query: Receiving input
Search Results

Parser — Format a DataFrame or Data object into text using a template. Enable 'Stringify' to convert input into a readable string instead.
Mode: Parser | Stringify
Data or DataFrame*
Parsed Text

Prompt — Create a prompt template with dynamic variables.
Template:
{context}

Given the context above, answer the question as best as possible.
context: Receiving input
question: Receiving input
Prompt Message

OpenAI — Generates text using OpenAI LLMs.
Input: Receiving input
System Message: Type something...
Model Name: gpt-4o-mini
OpenAI API Key*: ••••••••••••••••••••••••
Message
Language Model

Chat Output — Display a chat message in the Playground.
Text*
Message

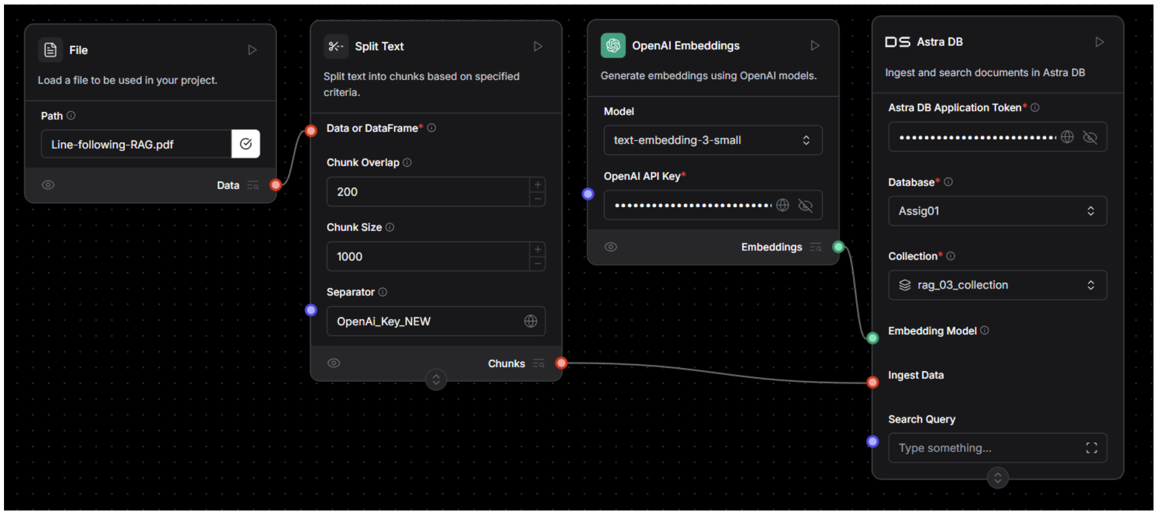
File — Load a file to be used in your project.
Path: Line-following-RAG.pdf
Data

Split Text — Split text into chunks based on specified criteria.
Data or DataFrame*
Chunk Overlap: 200
Chunk Size: 1000
Separator: OpenAi_Key_NEW
Chunks

OpenAI Embeddings — Generate embeddings using OpenAI models.
Model: text-embedding-3-small
OpenAI API Key*: ••••••••••••••••••••••••
Embeddings

Astra DB — Ingest and search documents in Astra DB
Astra DB Application Token*: ••••••••••••••••••••••••
Database*: Assig01
Collection*: rag_03_collection
Embedding Model
Ingest Data
Search Query: Type something...

The above screenshot shows the entire Langflow pipeline, from the document upload to chat output. Each node is visually connected, representing the flow of data through components such as file upload, text splitting, embedding, semantic search in Astra DB, prompt engineering, and final response generation.



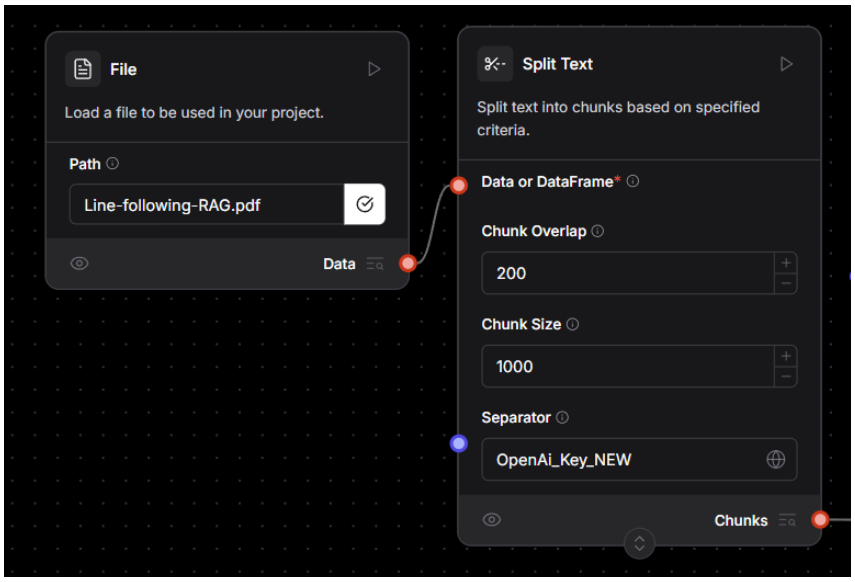
File — Load a file to be used in your project.
Path: Line-following-RAG.pdf
Data

Split Text — Split text into chunks based on specified criteria.
Data or DataFrame*
Chunk Overlap: 200
Chunk Size: 1000
Separator: OpenAi_Key_NEW
Chunks

OpenAI Embeddings — Generate embeddings using OpenAI models.
Model: text-embedding-3-small
OpenAI API Key*: ••••••••••••••••••••••••
Embeddings

Astra DB — Ingest and search documents in Astra DB
Astra DB Application Token*: ••••••••••••••••••••••••
Database*: Assig01
Collection*: rag_03_collection
Embedding Model
Ingest Data
Search Query: Type something...

This part of the flow begins with the File node, which loads the PDF document. It connects to the Split Text node that breaks the document into manageable chunks of 1000 characters with 200-character overlap. This ensures contextual continuity across segments.



File

Load a file to be used in your project.

Path: Line-following-RAG.pdf

Data

Split Text

Split text into chunks based on specified criteria.

Data or DataFrame*

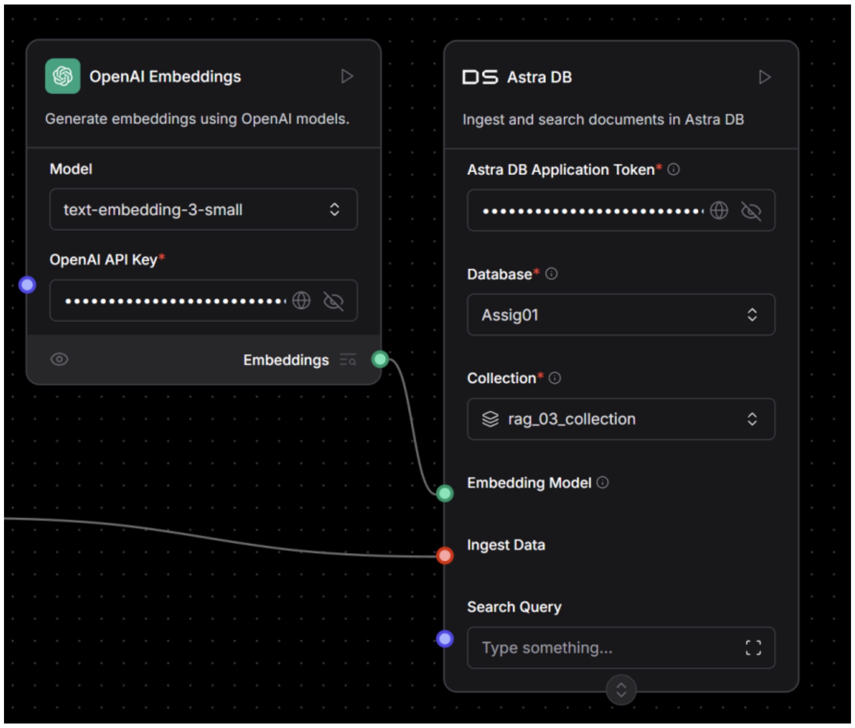
Chunk Overlap: 200

Chunk Size: 1000

Separator: OpenAi_Key_NEW

Chunks

The Split Text output feeds into the OpenAI Embeddings node, which transforms each text chunk into numerical vector representations using the text-embedding-3-small model. These vectors are essential for semantic similarity search in the database.



OpenAI Embeddings

Generate embeddings using OpenAI models.

Model: text-embedding-3-small

OpenAI API Key*: ••••••••••••••••••••••••••

Embeddings

Astra DB

Ingest and search documents in Astra DB

Astra DB Application Token*: ••••••••••••••••••••••••••

Database*: Assig01

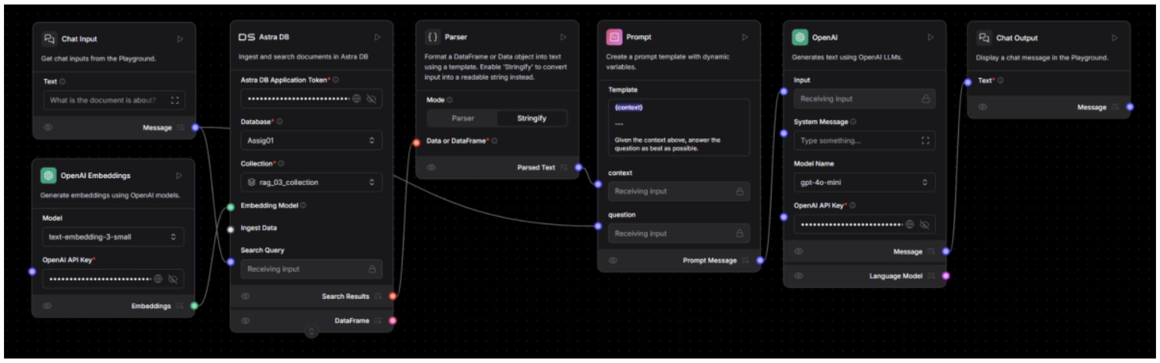
Collection*: rag_03_collection

Embedding Model

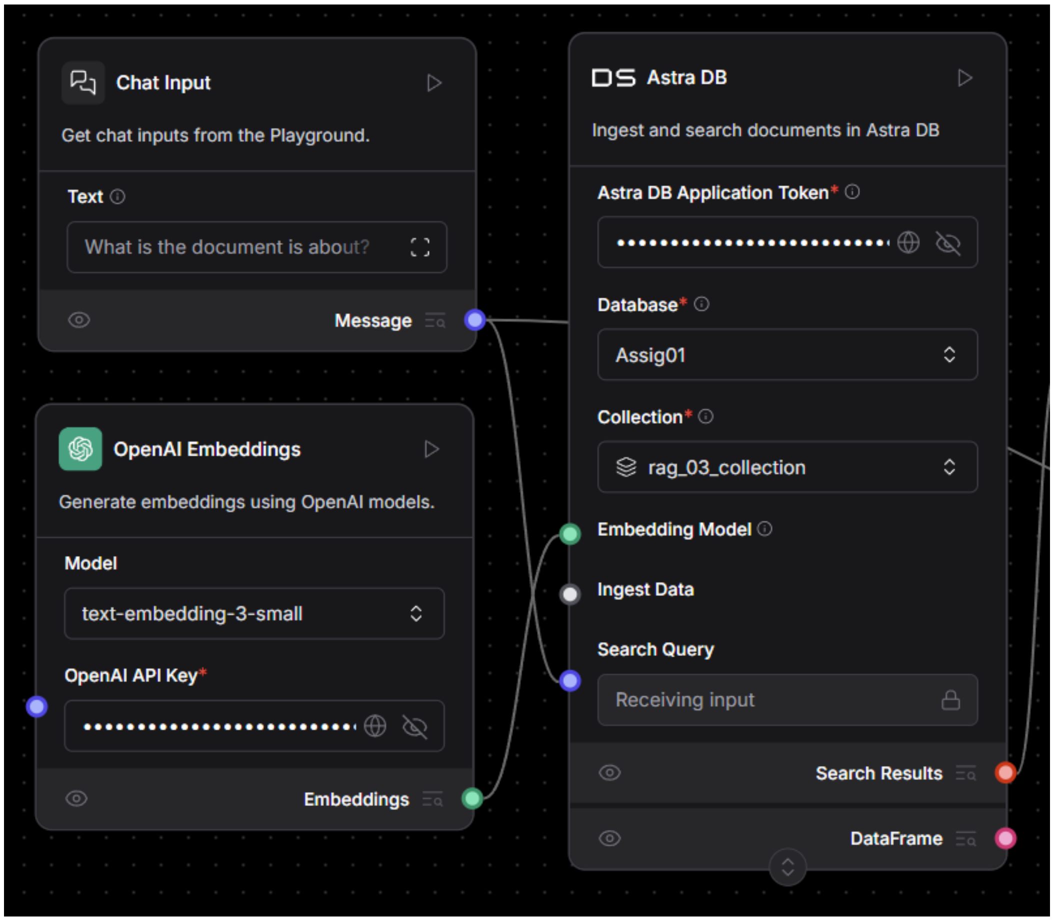
Ingest Data

Search Query: Type something...

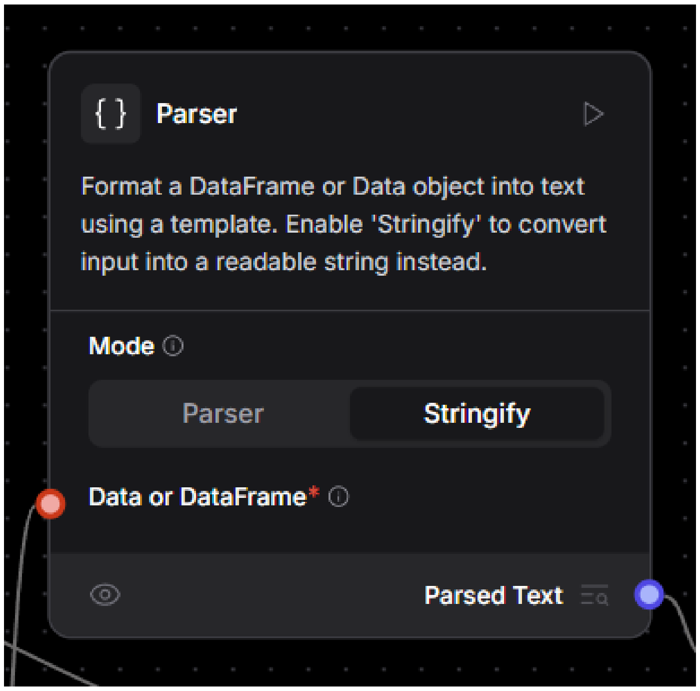
The vector embeddings are stored in Astra DB via the “Ingest Data” field in the Astra DB node. This establishes the knowledge base that the system can later search through using semantic queries.



This image illustrates the complete user query processing pipeline. The user's message is passed from the Chat Input node to the Astra DB for context retrieval, then through a Parser and Prompt Template, into the OpenAI LLM, and finally to the Chat Output.



This image shows the search phase of Astra DB. It shows the use of the same embedding model for consistent query processing, ensuring the semantic match between user input and stored chunks.



Parser

Format a DataFrame or Data object into text using a template. Enable 'Stringify' to convert input into a readable string instead.

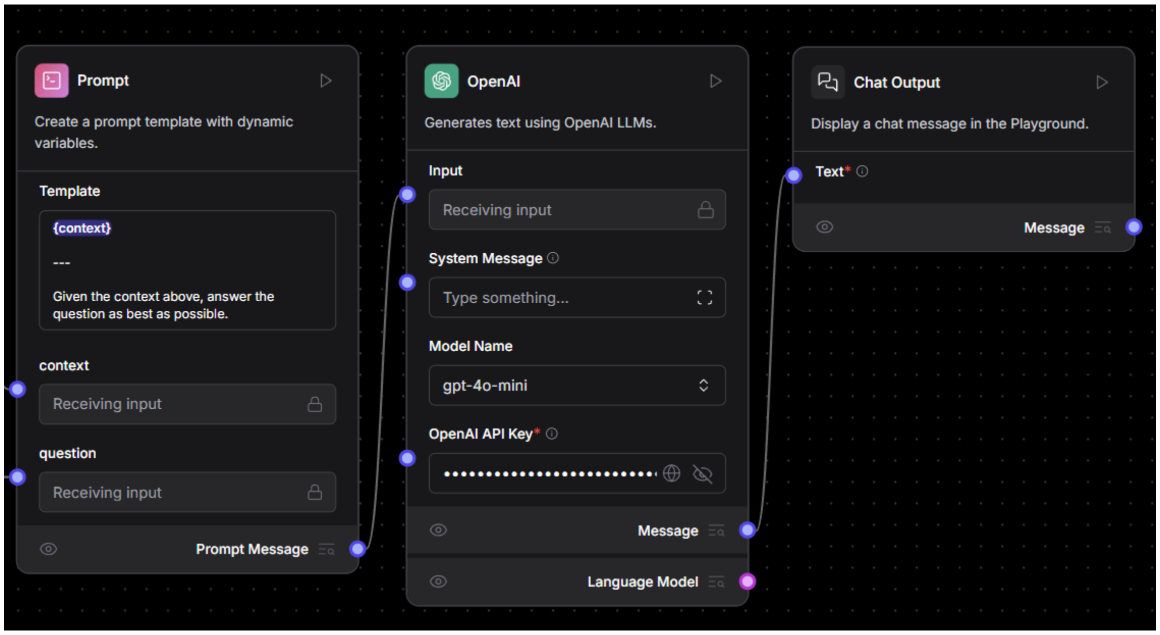
Mode

Parser [✓ Stringify]

Data or DataFrame*

Parsed Text

The Parser node reformats the search results into a clean textual format, which is easier to pass into the prompt. This acts as a translator between Astra DB's data and natural language prompts.



Prompt

Create a prompt template with dynamic variables.

Template

```
{context}

---

Given the context above, answer the question as best as possible.
```

context: Receiving input

question: Receiving input

Prompt Message

OpenAI

Generates text using OpenAI LLMs.

Input: Receiving input

System Message: Type something...

Model Name: gpt-4o-mini

OpenAI API Key*: ••••••••••••••••••••••••••••

Message

Language Model

Chat Output

Display a chat message in the Playground.

Text*

Message

This section contains the core reasoning logic. A templated prompt receives both the retrieved context and the user question. It is passed to GPT-4o-mini for processing and generates a context-aware response.

Challenges and Solutions

During the implementation of this system, I faced the challenges below:

The first challenge was to install Langflow. After many unsuccessful attempts, on a Windows system and since my Linux system was underpowered, I decided to build the RAG online.
This led to the second challenge that I encountered, which was slow response times, likely due to network latency or model loading in Langflow Playground.
The platform crashed one time, for unknown reasons but fortunately it didn't require a rebuild as everything was saved in advance.
I was not aware of the acceptable file sizes of the uploaded documents in the beginning but I addressed that by limiting the size of the documents and also by limiting chunk sizes to improve performance (didn't see too much of an improvement on that though).
The upload of multiple documents is something that I didn't have the time to resolve, hope I can manage that until the next Robotex competition.

Additional Features

While the base system returns answers from the uploaded document, a proposed improvement would be to add citation links or paragraph references to the returned answer. This would allow users to verify the accuracy of the results. And of course the implementation of the system to our website for our robotics teams to consult.