INTELLEXA.AI

# Problem Statement

· Extract data from a single Wikipedia page.

· Load the extracted data into a vector database (preferred) or use in-memory storage.

· Utilize a generative AI model to answer questions based on the loaded data.

· Implement the solution using Fast API.

· Create two Fast API endpoints:

1. An endpoint for loading the Wikipedia data into the vector database.
2. An endpoint for querying the generative AI model to generate answers from the stored data.

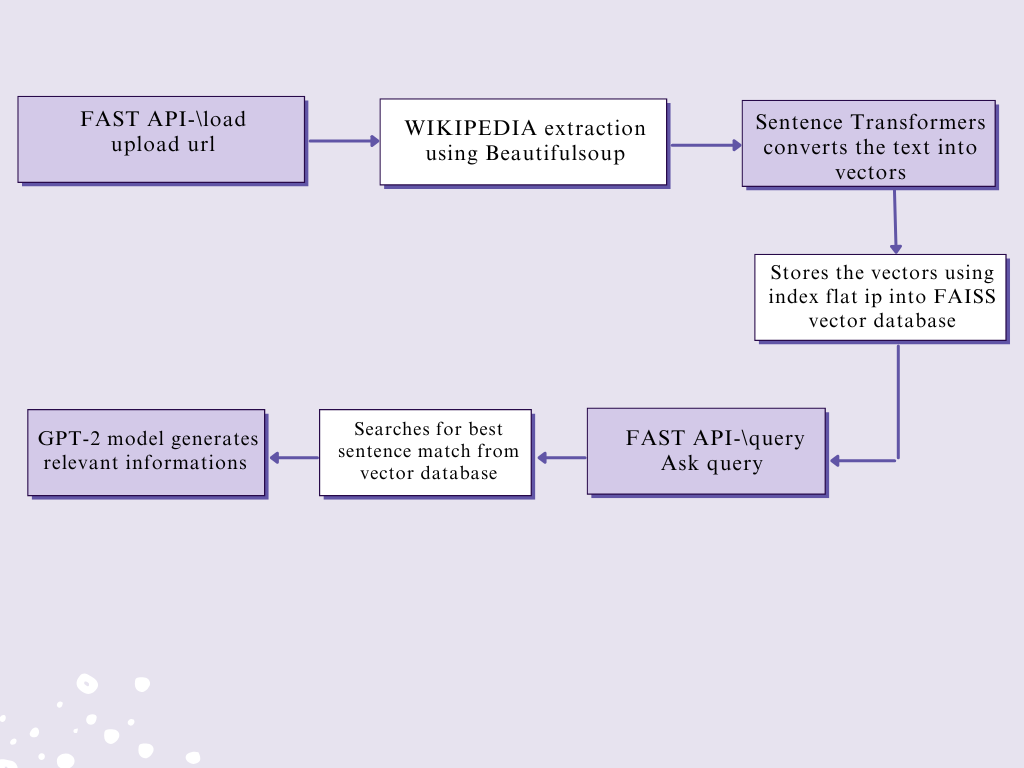
# Objective

To design and implement a Python-based system that efficiently extracts data from a single Wikipedia page dynamically and loads it into a vector database. The system will leverage a generative AI model to answer user queries based on the loaded data. The solution will include two FastAPI endpoints to facilitate:

1. Loading Wikipedia data into the vector database.
2. Querying the generative AI model to provide relevant answers from the stored data.

This solution aims to demonstrate how AI models can be used for knowledge extraction and question-answering tasks with dynamic data inputs.

# Flow Chart



# Modules

1. Web Scraping
2. Load to Vector Database
3. Generative AI Model
4. Creating Fast API
5. Ngrok

# Web Scraping In this process, BeautifulSoup, along with Python's requests library, is used to extract data directly from Wikipedia's web pages. This method involves sending HTTP requests to Wikipedia, retrieving the HTML of the desired page, and parsing the page to extract the relevant content.

# Load to Vector Database

To load the extracted data from Wikipedia into a vector database, the extracted texts are converted into vectors (numerical representations) that is stored in a vector database for future querying. To convert these sentences into vector, sentence transformer such as sentence-transformers/all-MiniLM-L6-v2 is used for embeddings. These are then stored in FAISS database.

# Generative AI Model

After the vector database (FAISS) stores the Fast API representations (Fast API) of large text data, a pre-trained generative AI model (GPT-2) is used to generate an answer to the user query. The retrieved context is combined with the user’s query to form a prompt that is passed to the generative AI model. The GPT-2 model processes this prompt and generates a coherent, context-aware answer based on the provided information. The model’s pre-training enables it to understand the relationship between the question and the context, allowing it to generate an appropriate answer.

# Fast API

The application consists of two main endpoints:

1. **Load Data Endpoint**: This endpoint allows users to specify a Wikipedia URL from which data will be extracted, processed, and stored in a vector database.
2. **Query Endpoint**: This endpoint accepts user questions and retrieves relevant information from the vector database to generate answers based on the previously loaded data.

# Ngrok:

Ngrok is used in this code to expose the FastAPI application running on a local machine to the internet via a secure, public URL. This is useful for testing APIs, web applications, or sharing your development environment without needing to deploy it to a remote server or configure complex network settings.

1. **Authentication**: Ngrok requires an authentication token to use its services. The code contains an Ngrok auth token, which is used to authenticate the session (!ngrok authtoken {ngrok\_auth\_token}).
2. **Tunneling**: Ngrok starts a tunnel to forward requests made to the local server (running on port 8000) to a public URL generated by Ngrok (ngrok.connect(8000)). The public\_url printed out can be accessed over the internet.
3. **Usage**: Once the tunnel is established, anyone with the public URL can access your local Fast API application.

# Procedure:

### Step 1: Import Required Libraries

The script starts by importing all the necessary libraries for:

* Web scraping (requests, BeautifulSoup)
* Embeddings and search (sentence\_transformers, faiss, numpy)
* Model-based text generation (transformers)
* FastAPI for API creation
* Pyngrok for exposing the local API to the internet
* Miscellaneous utilities like file handling (os), etc.

### Step 2: Initialize FastAPI App and CORS Middleware

The FastAPI application is initialized:

* CORS (Cross-Origin Resource Sharing) is enabled for all origins, methods, and headers so that the API can be accessed from any domain.

### Step 3: Load Sentence Transformer and Hugging Face Generative Model

* A Sentence Transformer model (all-MiniLM-L6-v2) is loaded for sentence embedding.
* A Hugging Face model (e.g., GPT-2) is loaded for text generation.

### Step 4: FAISS Index Setup and Data Handling

* The FAISS index is initialized as None and will later store text embeddings.
* The stored\_sentences list will hold the sentences from the scraped text.
* SCRAPED\_TEXT\_FILE is defined to store the scraped Wikipedia content.

### Step 5: Define Fast API Request Models

Two request models are created using Pydantic:

* LoadDataRequest: Handles the URL to scrape data from.
* QueryRequest: Handles the user’s question.

### Step 6: Wikipedia Data Scraping

A function extract\_wikipedia\_data(url) is defined to scrape content from the Wikipedia page:

* It fetches the content using the requests library.
* Extracts text from the HTML <p> tags using BeautifulSoup.
* Saves the scraped text into a file.

### Step 7: Embedding and FAISS Storage

**embed\_and\_store\_text(text):**

* Splits the scraped text into individual sentences.
* Converts these sentences into embeddings using the Sentence Transformer.
* Stores these embeddings in a FAISS index using cosine similarity.

### Step 8: Load and Query Endpoints

Two FastAPI endpoints are defined:

1. **/load**: This endpoint scrapes the Wikipedia URL and stores the embeddings in FAISS. The request body contains the Wikipedia URL.
2. **/query**: This endpoint handles user queries by searching the FAISS index for the most relevant sentences, then passing these sentences as a prompt to the GPT-2 model for text generation.

### Step 9: Set Up Ngrok for Public Access

The script authenticates and sets up a tunnel using Ngrok to make the local FastAPI application accessible over the internet.

### Step 10: HTML Page and Background Image

* The script defines an HTML page for the frontend interface that allows users to input a Wikipedia URL or a query.
* The background is served via the /background route.
* JavaScript functions are used to trigger the loadData and queryData API calls.

### Step 11: Fast API Server Execution

The script starts the Fast API server using uvicorn.

# Result:

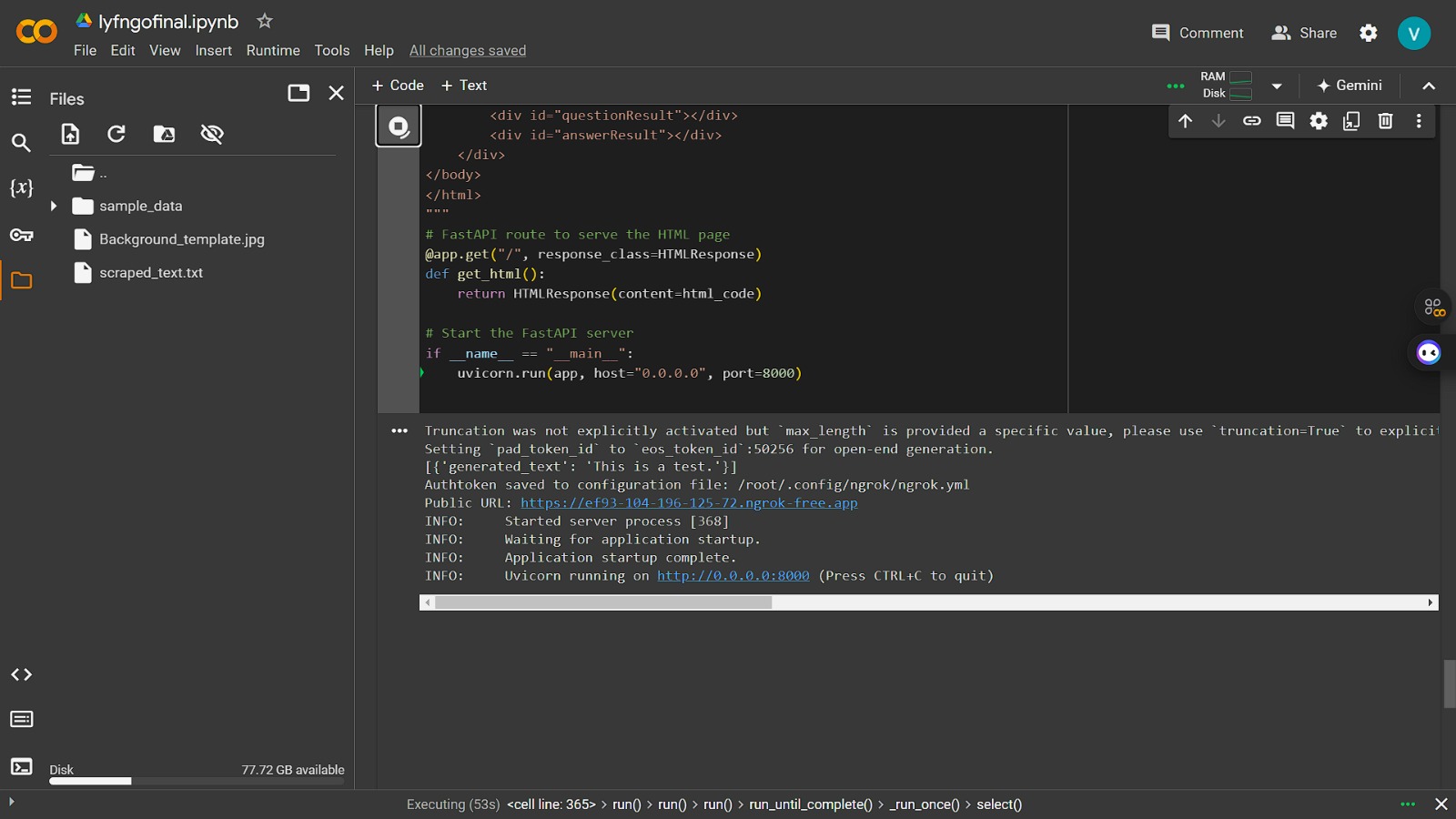


Fig 1.1 Ngrok url directs to webpage



Fig 1.2 Fast API/load



Fig 1.3 Fast API/query

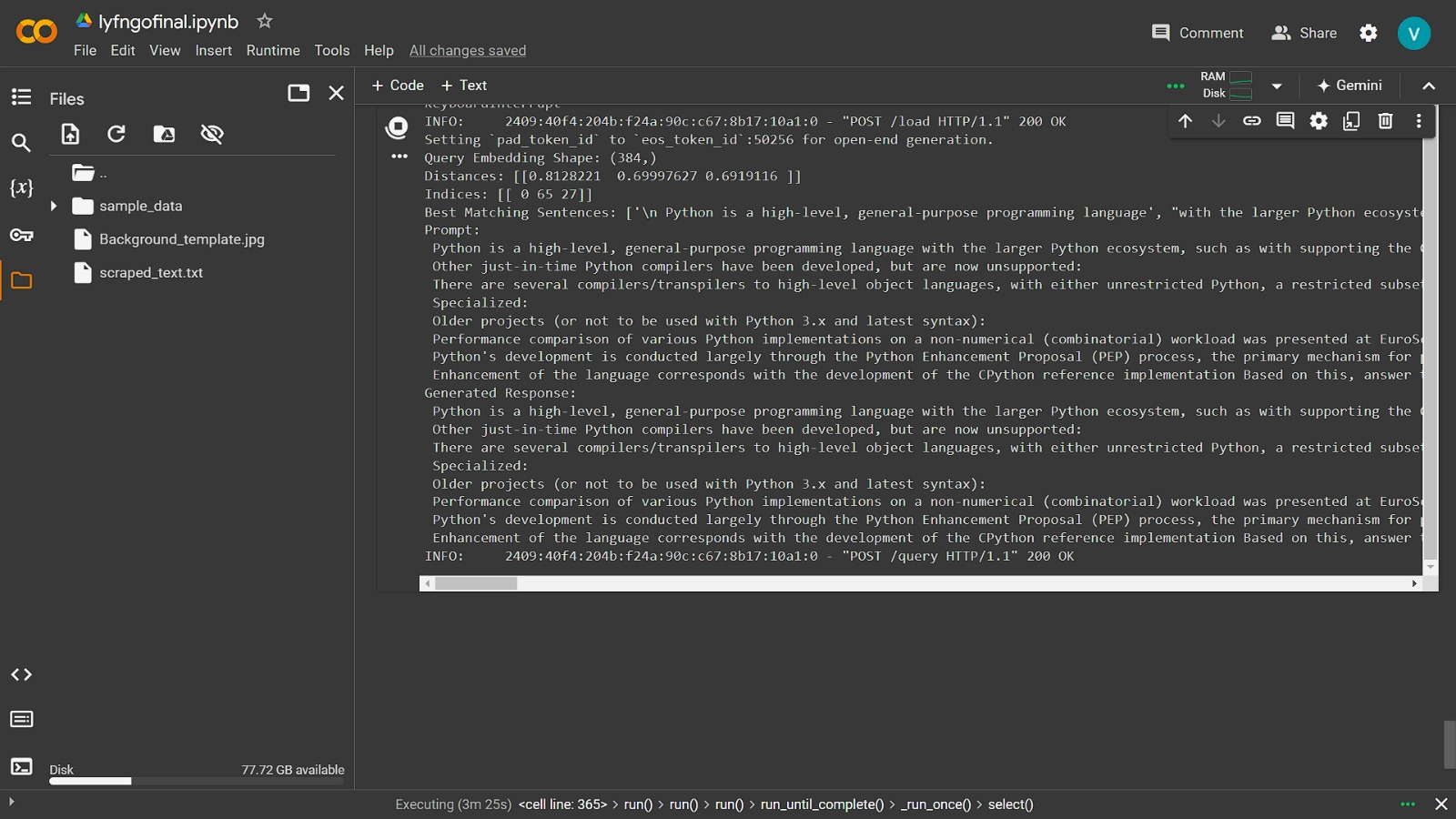


Fig 1.4 Generating results