**Smart Search Tool for Analytics Vidhya Free Courses**

**Objective**

The goal of this project is to create a smart search tool to help users find relevant free courses from the Analytics Vidhya platform using natural language queries and keywords. The tool leverages a Retrieval-Augmented Generation (RAG) approach and is deployed publicly on Huggingface Spaces for evaluation.

**Approach and Methodology**

**1. Data Collection**

* **Process**: Scraped or manually collected the course data (e.g., titles, descriptions, and curriculum) from Analytics Vidhya's free courses section.
* **Storage**: Stored the course data as text files in a structured format for processing.

**2. System Design**

The project involves the following components:

1. **Data Chunking and Embedding**:
   * Split the course data into manageable chunks using RecursiveCharacterTextSplitter from LangChain. Each chunk is small enough to capture relevant context for a search query.
   * Generated vector embeddings for each chunk using the SentenceTransformer model (all-MiniLM-L6-v2), which provides a compact and high-quality representation.
2. **Vector Database**:
   * Used **Qdrant** as the vector database to store the generated embeddings. Qdrant supports efficient vector similarity searches, enabling quick retrieval of relevant chunks.
3. **Retrieval Mechanism**:
   * Implemented a vector similarity search using the Qdrant API. Given a user query, the system retrieves the top 5 most relevant chunks based on cosine similarity.
4. **Generative Model for Response**:
   * Used Google's Gemini API (or any preferred LLM, e.g., OpenAI's GPT-3.5) to process retrieved chunks and the query to generate a user-friendly response.
5. **Deployment**:
   * Deployed the search tool using **Streamlit** on Huggingface Spaces for easy accessibility and user interaction.

**Implementation Details**

**Libraries and Tools Used**

1. **LangChain**: For text chunking and managing the RAG pipeline.
2. **SentenceTransformers**: For generating embeddings from text.
3. **Qdrant**: Vector database for storing and retrieving embeddings.
4. **Streamlit**: Frontend framework for building the UI.
5. **Google Gemini API**: (Optional) For enhancing generative responses based on retrieved content.

**Key Functions**

1. **Data Processing**:
   * read\_txt\_files(directory): Reads and prepares course data from text files.
   * upload\_chunks\_to\_QDrant(documents, collectionName): Generates embeddings for text chunks and uploads them to Qdrant.
2. **Retrieval**:
   * vector\_search(query, collection\_name, top\_k): Retrieves top-k chunks from Qdrant based on similarity to the query.
3. **Generative Answer**:
   * gemini(query, chunks): Uses a generative AI model to create a response based on retrieved chunks and user query.
4. **Frontend**:
   * **Streamlit**: Provides a user-friendly interface where users can input queries and receive results.

**Deployment**

* Hosted the tool on **Huggingface Spaces** for public access.
* Used **Streamlit** as the web interface for user interaction, allowing quick deployment via Huggingface's simple hosting process.

**How to Use the Tool**

1. **User Query**: Enter a keyword or natural language query (e.g., "Data Science course with Python").
2. **Search Process**:
   * The system retrieves the most relevant chunks of information from the vector database.
   * Processes the results with a generative AI to provide a coherent and concise response.
3. **Output**: A list of courses with brief descriptions and links to the full course details.

**Methodology**

**Embedding Model Selection**

* **SentenceTransformer (**all-MiniLM-L6-v2**)**:
  + Selected for its compact size (vector size = 384) and excellent performance for sentence-level similarity tasks.
  + Efficient for large-scale vector storage and quick retrieval.

**Generative Model Selection**

* **Google Gemini**:
  + Chosen for its capability to process contextual information and generate high-quality, human-like responses.
  + Optionally, OpenAI's GPT-3.5 can also be used based on API availability.

**Evaluation**

**Performance Metrics**

1. **Relevance**: Assessed the quality of search results by comparing retrieved chunks with user queries.
2. **Response Clarity**: Evaluated the coherence and informativeness of responses generated by the generative model.
3. **Speed**: Measured the latency of the search and generation process to ensure a responsive user experience.

**Future Improvements**

1. **Course Links**: Automatically include direct links to course pages in the response.
2. **Improved Chunking**: Experiment with dynamic chunk sizes to better capture course details.
3. **Enhanced Query Understanding**: Incorporate advanced query processing techniques, such as query expansion, for better search accuracy.