Final Project: Portfolio Project

Subject: INFO 7390 - Advances in Data Sciences and Architecture

Course Recommender Assistant Powered by AI (LLM) and RAG

Group Number 15

1.Chinmay Kulkarni (002209448)

2.Pranitee Majukar (002209824)

3.Shreeanant Bharadwaj (002649430)

UNDER THE GUIDANCE OF

Professor Nicholas Brown

1. Introduction

The Course Recommender Assistant leverages advanced AI techniques, including Language Model (LLM) capabilities and Retrieval-Augmented Generation (RAG), to provide personalized and contextually accurate course recommendations tailored to user queries. By integrating state-of-the-art data preprocessing methods, robust vector database management, and a dynamic frontend interface, the system creates an efficient and user-friendly experience for learners seeking relevant educational content across multiple platforms. The assistant is designed to streamline the course discovery process by understanding user intent through natural language inputs, retrieving the most relevant results, and providing advanced filtering options to refine recommendations further.

At its core, the assistant employs a modular architecture that ensures seamless integration of its components. The workflow begins with data ingestion, where structured datasets are collected from sources such as Kaggle and web scraping. These datasets, containing information on course titles, platforms, durations, ratings, and user reviews, are then subjected to rigorous preprocessing. This involves cleaning textual data, extracting meaningful features, and generating embeddings using OpenAI\u2019s Ada-002 model. These embeddings, stored in Pinecone\u2019s high-speed vector database, enable efficient similarity searches that match user queries with the most relevant courses.

The user interface, built with Streamlit, provides a clean and interactive platform where users can input queries in natural language, apply filters based on criteria such as duration, platform, and ratings, and receive personalized recommendations. The integration of the RAG pipeline allows the system to combine retrieved course embeddings with the contextual understanding capabilities of LLMs, ensuring recommendations are both relevant and context-aware. Dynamic updates to the UI ensure a seamless user experience, with results displayed in real-time as filters are adjusted or new queries are entered.

This assistant not only addresses the challenges of course discovery in an overwhelming sea of online learning options but also ensures scalability and adaptability for future enhancements. By leveraging cutting-edge AI tools and techniques, the Course Recommender Assistant bridges the gap between learners and quality educational resources, making it an indispensable tool for personalized learning. This document outlines the technical implementation, system design, and advanced features of the assistant, providing an in-depth view of its architecture and functionalities.

2. Objectives

The primary objective of the Course Recommender Assistant is to provide a streamlined and user-friendly solution for learners to discover relevant courses across multiple platforms. Specific objectives include:

- Personalizing course recommendations based on user queries.
- Enabling advanced filtering options to refine search results.
- Delivering quick and accurate responses using vector search and embeddings.

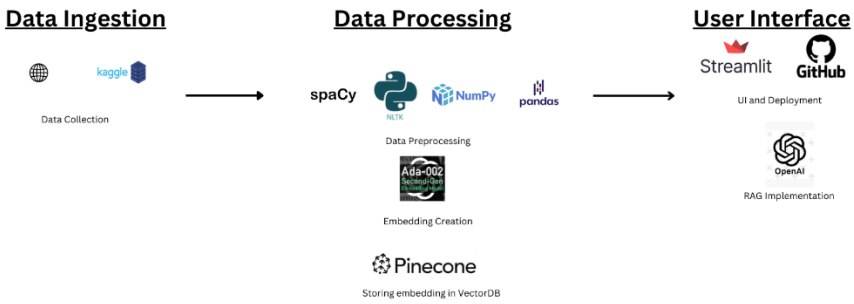
3. Technologies and Tools Used

- **Data Preprocessing:** Pandas, Regex, NumPy, Natural Language Processing (NLP) tools.
- **Database Management:** Pinecone for vector similarity search.
- **AI/ML Models:** OpenAI API for generating embeddings.
- **Frontend Development:** Streamlit.
- **Miscellaneous Tools:** Docker for deployment, Git for version control.

4. System Architecture

The project follows a modular architecture for seamless integration of its components. Below is the workflow:

1. **User Query Input:** Users enter queries through the frontend.
2. **Query Preprocessing:** The system preprocesses the input query for embedding generation.
3. **Vector Search:** Pinecone performs a similarity search to retrieve relevant course embeddings.
4. **Results Aggregation:** Retrieves matching course details and ranks them based on relevance.
5. **Frontend Display:** Displays results with course details and user-friendly filters.



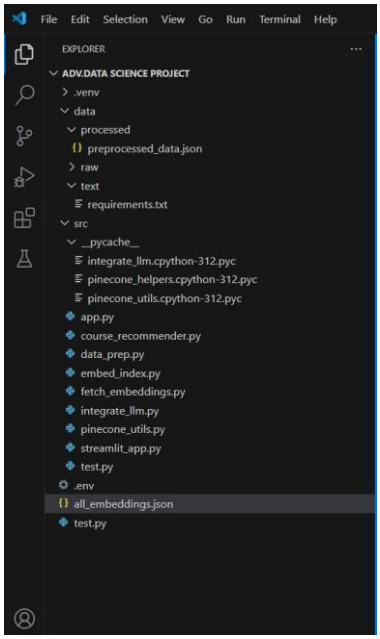
Explanation of the work flow: The workflow for the Course Recommender Assistant begins with **Data Ingestion**, where raw course data is collected from platforms like Kaggle and other web resources. This data includes attributes such as course titles, descriptions, platforms (e.g., Coursera, Udemy, edX), duration, ratings, and user reviews. The purpose of this step is to ensure a rich and structured dataset that forms the basis for meaningful recommendations. The collected raw data is then prepared for the subsequent processing phase.

In the **Data Processing** stage, the collected data undergoes extensive preprocessing using tools like spaCy, NLTK, Pandas, and NumPy. The process involves cleaning the text by removing unnecessary characters such as HTML tags and special symbols, tokenizing the course descriptions into smaller, meaningful text units, and extracting features such as keywords, platforms, and durations. This ensures the data is standardized and ready for downstream tasks. After preprocessing, the textual data is converted into numerical vector representations, or embeddings, using OpenAI's Ada-002 (Second Gen) embedding model. These embeddings capture the semantic meaning of course descriptions, making them suitable for similarity-based recommendations. The embeddings are then stored in Pinecone, a high-speed vector database that enables efficient similarity searches. Pinecone also supports metadata indexing, such as category, platform, and ratings, allowing for advanced filtering during query processing.

The final stage involves the **User Interface (UI)** and **RAG (Retrieval-Augmented Generation) Implementation**. The UI is built using Streamlit, which provides an interactive web-based platform for users to input queries, apply filters, and view recommendations. GitHub is used for version control and maintaining the project repository. The interface allows users to type free-text queries like "Courses on Machine Learning" and refine their results with filters based on duration, platform, or ratings. The RAG pipeline combines the power of OpenAI's LLM and Pinecone's vector database. User queries are processed to generate embeddings, which are then matched against the stored vectors in Pinecone through a similarity search. The system retrieves and ranks the most relevant courses based on the query, and the results are displayed dynamically in the UI.

This workflow demonstrates an efficient integration of data ingestion, preprocessing, vector storage, and user interaction, ensuring a seamless and personalized course recommendation experience. It leverages modern AI tools and frameworks to provide accurate and scalable results while maintaining a modular structure for future enhancements.

2. Steps and Implementation Details



Step 1: Data Collection and Preprocessing

1.1 Data Collection

Data collection serves as the foundation of the Course Recommender Assistant by gathering structured and relevant course information from multiple reliable sources.

- **Sources:**
    - **Kaggle Datasets:** These provide structured and pre-labeled course data, including course titles, descriptions, and metadata, to establish a baseline dataset for the project.
    - **Web Scraping:** Scraping techniques are employed to extract additional course-related information from various learning platforms such as Coursera, edX, and Udemy. This ensures a broader and richer dataset that captures up-to-date and platform-specific course details.
- **Information Collected:**
    - **Course Title:** The name of the course for identification and retrieval.
    - **Platform:** Specifies the source platform (e.g., Coursera, edX, Udemy) where the course is hosted.
    - **Duration:** The estimated time required to complete the course.
    - **Ratings and User Reviews:** Includes average ratings and qualitative feedback from learners.
    - **Course URL:** Direct links to the respective course pages for easy navigation.
    - **Course Description:** A summary of the course content, which is critical for understanding the subject matter and for embedding generation.

1.2 Data Preprocessing

Once the data is collected, it is cleaned, structured, and prepared for embedding generation and storage in a vector database. The preprocessing phase ensures that the data is standardized, accurate, and optimized for downstream tasks.

- **Technologies Used:**
    - **Pandas:** For data manipulation, handling missing values, and transforming raw data into structured formats.

- **Regex (Regular Expressions):** For text cleaning by identifying and removing unwanted patterns like special characters and HTML tags.
- **NumPy:** For handling numerical operations during preprocessing.
- **NLP Tools (spaCy, NLTK):** For tokenization and feature extraction.

- **Steps Involved:**

1. **Text Cleaning:**
   - Removed special characters (e.g., #, @), HTML tags, and irrelevant symbols to clean and standardize text descriptions.
   - Addressed missing data issues by either removing incomplete records or filling in missing values using techniques like mean or mode imputation with Pandas.
2. **Tokenization:**
   - Split the course descriptions into meaningful tokens or words using spaCy or NLTK. This step allows the system to break down descriptions into granular units for easier processing and analysis.
3. **Feature Extraction:**
   - Extracted key attributes like keywords, categories, and course durations to better represent course data.
   - Identified domain-specific features that could enhance recommendation accuracy, such as technical tags (e.g., Python, Machine Learning).
4. **Embedding Preparation:**
   - Normalized the text data to ensure uniformity (e.g., converting all text to lowercase, removing excess whitespace).
   - Vectorized the textual inputs, transforming them into embeddings compatible with vector-based systems. This step is critical for enabling semantic similarity searches in the vector database (Pinecone).

Step 2: Vector Database Setup

2.1 Selection of Tools

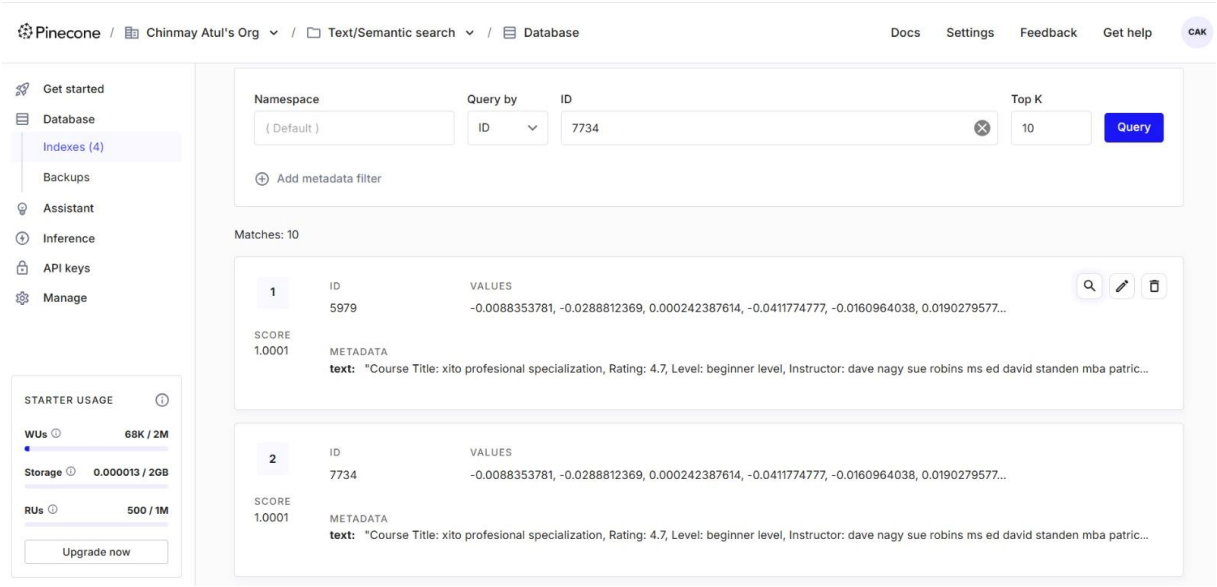
To enable high-speed and accurate similarity searches, the system uses the following tools:

- **Database:** Pinecone, a vector database optimized for managing and querying embeddings at scale, is used to store and retrieve course embeddings efficiently.
- **API:** OpenAI's state-of-the-art LLM models are employed for generating embeddings. These embeddings represent the semantic meaning of course descriptions and user queries, making them suitable for similarity-based retrieval.

2.2 Indexing and Data Storage

The preprocessed data is indexed and stored in Pinecone to facilitate real-time searches and filtering:

1. **Configuration:** A vestores and retrievesctor database instance is set up on Pinecone to handle large-scale embedding storage and querying.
2. **Uploading Embeddings:** The preprocessed embeddings generated from course descriptions are uploaded into Pinecone. These embeddings allow for efficient retrieval of similar courses based on user queries.
3. **Metadata Tagging:** To enable advanced filtering, each course embedding is indexed with metadata tags such as:
   - **Categories:** Technical domains (e.g., Data Science, Machine Learning).
   - **Platforms:** Specific sources like Coursera, edX, or Udemy.
   - **Keywords:** Domain-specific terms and technical tags.
   - **Ratings:** User reviews and average ratings.



2.3 Implementing Filtering Mechanisms

To refine search results, the system incorporates advanced filtering mechanisms:

- **Keyword-Based Filtering:** Courses are grouped by technical domains (e.g., Data Science, Artificial Intelligence, Web Development) to ensure more relevant results.
- **Custom Filters:** Additional options are provided to let users narrow down their search results, including:
    - **Duration:** Courses are categorized into Short (0–3 hours), Medium (3–10 hours), and Long (10+ hours).
    - **Platform:** Filters allow users to specify the course provider (e.g., Coursera, edX).
    - **Ratings:** Users can display only the top-rated courses above a specified threshold for quality assurance.

Step 3: Frontend Development

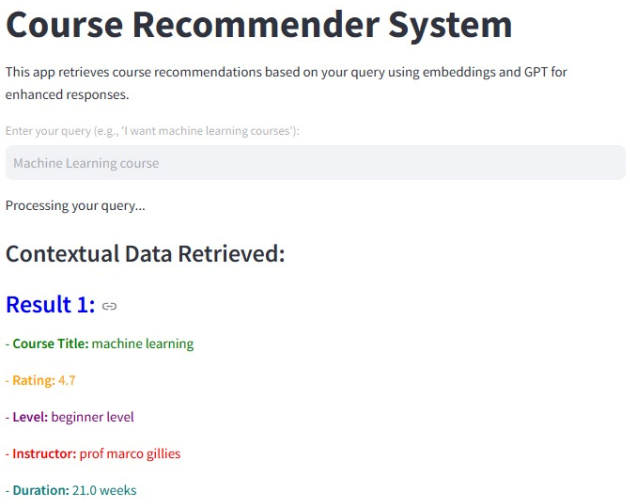
3.1 Framework Selection

- **Primary Framework:** The frontend is developed using Streamlit, which allows for fast prototyping and simple UI implementation, making it ideal for an interactive recommender system.
- **Alternative Option:** React.js can be explored in the future for enhanced scalability and advanced interactivity.

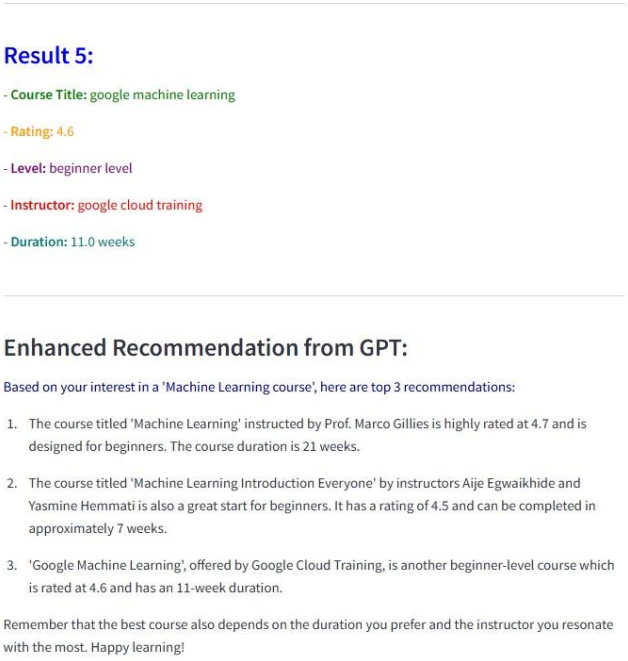
3.2 Designing the Interface

The interface is designed to be intuitive and user-friendly:

1. **Homepage:**
    - Includes a welcome message to introduce users to the recommender system.
    - Provides a brief explanation of how the system works.



2. **Query Input Section:**
    - A text box is provided for users to enter natural language queries (e.g., "I want to learn Python for data analysis").
    - A button is added to submit the query for processing.



3. **Results Display:**
    - Displays a summarized list of recommended courses, including:
    - **Title:** The name of the course.
    - **Platform:** The source of the course (e.g., Coursera, Udemy).
    - **Duration:** Time required to complete the course.
    - **Average Rating:** Aggregated user feedback for quality assurance.

4. **Filter Sidebar:**
    - Interactive dropdown menus or sliders allow users to filter results based on:
    - **Duration:** Short, Medium, or Long.
    - **Platform:** A specific course provider.
    - **Ratings:** Minimum acceptable rating.

5. **Interactivity:**
    - The interface dynamically updates the results in real time as users adjust filters.
    - Recommendations that closely match the user query are highlighted for easy selection.

3.3 Enhancing User Experience

To ensure a seamless and engaging experience, several features are implemented:

- **Responsive Design:** The UI is optimized for both mobile and desktop devices to accommodate a wide range of users.
- **Tooltips and Instructions:** Brief tooltips guide users on how to use filters and input queries effectively.
- **Loading Spinner:** A spinner indicates the system is processing a query, improving user feedback and interaction.

---

3. Advanced Features and Future Enhancements

Current Advanced Features

1. **Natural Language Understanding:** The RAG pipeline enables the system to interpret user queries and retrieve contextually relevant results from the vector database.
2. **Keyword Augmentation:** User queries are automatically expanded with synonyms and related terms to improve matching accuracy and recall.
3. **Real-Time Updates:** The interface updates dynamically as users interact with filters, ensuring a smooth and responsive experience.

Proposed Future Enhancements

1. **Enhanced Personalization:**
    - Incorporate user profiles to tailor recommendations based on individual preferences and past interactions.
    - Use browsing and search history to refine course suggestions further.
2. **Cross-Platform Integration:**
    - Expand the system to support multiple learning platforms simultaneously, allowing users to discover courses from a variety of sources in one interface.
3. **Recommendation Scores:**
    - Display confidence scores for each recommended course, helping users understand how closely a course matches their query.
4. **Improved Frontend:**
    - Transition to a React.js-based UI for better performance, advanced interactivity, and visually appealing aesthetics.

• **References:**

o Streamlit Documentation

o Pandas Documentation

o NumPy Documentation

o spaCy Documentation

o NLTK Documentation

o Pinecone Documentation

o OpenAI API Documentation

o Regex Reference

o Docker Documentation

o GitHub Documentation

• **Source Code:**

4. Conclusion

The Course Recommender Assistant successfully combines advanced AI methodologies with intuitive user interaction to address the challenge of personalized course discovery. By using RAG, vector databases, and modern frontend technologies, it delivers a scalable and user-friendly solution that bridges the gap between learners and quality educational content.