EEO Case Management

Streamlining the process by leveraging a Large Language Model (GPT 3.5)

Submitted by: **MOKSHA KOTHARI**

Job Role: **LLM Engineer**

INDEX

Abstract: EEO (Equal Employment Opportunity) Case Management refers to the systematic process of handling complaints and cases related to workplace discrimination, harassment, and other violations of equal employment opportunity laws and regulations.

The goal is to ensure that all employees are treated fairly and that any issues of discrimination or unfair treatment are addressed promptly and effectively.

Managing and analyzing Equal Employment Opportunity (EEO) cases can be a complex and time-consuming task. EEO cases often involve sensitive details that need to be categorized, summarized, and analyzed efficiently to ensure fair and just resolutions. The manual processing of these cases is prone to human error and can be highly inefficient.

Problem Statement: Implementing a language model (LLM) to enhance EEO case management by building a Streamlit app that uses an LLM to process, categorize, and analyze case descriptions.

The system should have the following capabilities:

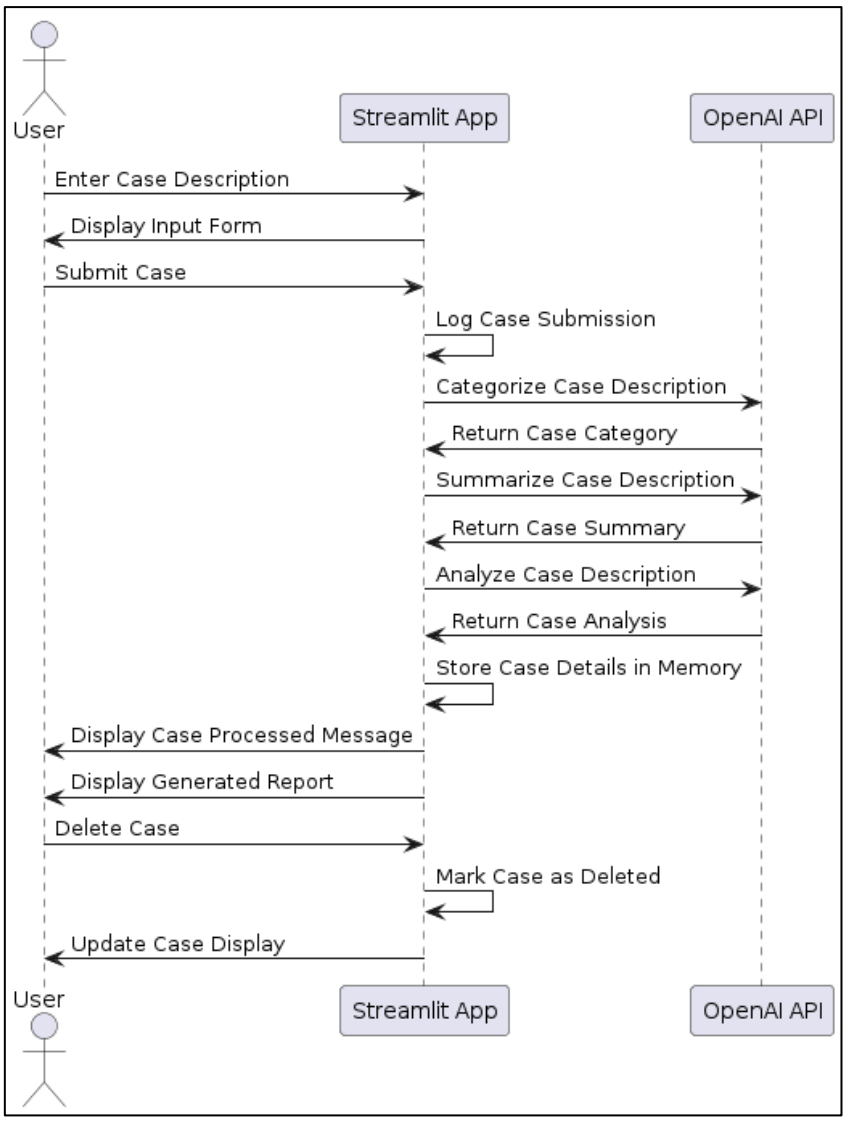
- **Input Case Description**: An interface to input new case descriptions.
- **Case Categorization**: Use an LLM to categorize the case into predefined categories
- **Case Summarization**: Generate a concise summary of the case description using the LLM.
- **Case Analysis**: Extract key entities and sentiments from the case description.
- **Case Management**: Display a list of all cases with their categories, summaries, and analyses. And, allow users to delete cases.

Approach: This project leverages OpenAI's language model and Streamlit's interactive web application framework to manage and analyze EEO cases. The application provides functionalities such as case categorization, summarization, analysis, and management, aiming to streamline the process and enhance efficiency.

Technologies Used:

- **Python**: The primary programming language used for developing the application. Python 3.10 used in this project.
- **Streamlit**: A fast and easy-to-use framework/chat interface for creating data-driven web applications.
- **OpenAI**: GPT3.5 Large Language Model for categorizing, summarizing, and analyzing case descriptions.
- **VS Code**: Project workspace

<u>Code Workflow</u>: The workflow diagram of the code i.e. “model.py” along with work breakdown structure:



1. **Initial Configuration**:
    - **Libraries**: Imported necessary libraries including:
        - Streamlit for the web app interface,
        - OpenAI for language model interactions, and
        - logging for tracking events.
    - **API Configuration**: Initialized the API:
        - Base(endpoint) URL and key for accessing the OpenAI service.
        - Configured the OpenAI client with the appropriate API type, version, base URL, and API key.
2. **Logging Setup**:
    - Configured logging to capture information and errors for monitoring the application's activity. A console handler was added to output logs to the console.

3. **Streamlit Application Interface**:
    - **Markdown for Styling**: Used Streamlit's markdown feature to create a styled header and set up a sidebar for input.
    - **Case Description Input**: Added a text area in the sidebar for users to input their case descriptions.
    - **Submit Button Functionality**: Implemented a submit button to trigger the processing of the case description.
4. **LLM Interaction for Case Processing**:
    - **Categorization**: Created a prompt to categorize the case description into predefined categories and obtained the response from the LLM.
    - **Summarization**: Generated a concise summary of the case description using the LLM.
    - **Analysis**: Extracted key entities and sentiments from the case description through the LLM.
5. **Data Storage and Case Management**:
    - **In-Memory Storage**: Used st.session_state to store the cases and track deleted cases within the session, and stored it as a list.
    - **Case Display**: Displayed the list of cases with their details including category, summary, and analysis. Provided functionality to delete cases, marking them as deleted and updating the display accordingly.
6. **Output and Interaction Handling**:
    - Displayed the processed cases in the main interface, including their category, summary, and analysis.
    - Added a delete button for each case, allowing users to mark cases as deleted and refresh the display.

Assumptions

1. **API Key and Endpoint Security**:
    - The API key is hardcoded for simplicity in this project, assuming a secure environment for testing.
2. **Session Management**:
    - Assumed that `st.session_state` is sufficient for managing session-specific data, given the requirement to store case data in an in-memory data structure.
3. **User Input**:
    - Assumed that users will provide case descriptions that are clear and suitable for categorization, summarization, and analysis by the LLM.
4. **API Response Handling**:
    - Error handling is implemented to log any issues with the API calls, assuming that network connectivity and API availability are generally reliable.
5. **Predefined Categories**:
    - Assumed that the predefined categories provided are comprehensive enough to cover most case descriptions users will input.

- Initialized as: ["Employment Discrimination", "Harassment", "Unfair Dismissal", "Other"]

6. **LLM Capabilities**:
   - Assumed that the LLM is capable of accurately categorizing, summarizing, and analyzing the case descriptions based on the provided prompts.

Advantages and Uniqueness

- **Efficiency**: Automates the categorization, summarization, and analysis of EEO cases, significantly reducing manual effort and processing time.
- **Accuracy**: Leverages GPT 3.5 to ensure accurate and comprehensive data collection and analysis.
- **Interactive UI**: Provides an easy-to-use and interactive interface for managing EEO cases, improving user experience and accessibility.
- **In-Memory Storage**: Uses Streamlit's session state for efficient and temporary in-memory data storage, ensuring smooth and seamless user interactions.
- **Logging**: Logging actions like case submission, processing, and deletion help in tracking the application's activity, which is useful for debugging and monitoring.
- **Session Management**: Storing cases and deleted cases in "st.session_state" ensures that the data persists across user interactions within the same session.

Other Solutions: The approach used in this project is simple and basic, which is implemented with the help of GPT3.5. But when we plan to integrate such a model in our organization, we must keep in mind the feasibility and security of our internal data.

Thus, we can make use of a locally downloaded LLM such as LLAMA2 (with our own API key) which can be integrated with the help of the Langchain framework to create a similar chatbot. It can be connected with a database in the backend to keep track of records.

Timechart: This project, including the working and documentation, was created within a day after the API dummy key was provided by the organization.

Vision: This project can be scaled efficiently and integrated with the organization's current ticket management system. Each case can be sent as a ticket to the respective HR representative. It must be treated as a task and the complaints must be resolved within the allotted timeframe.

This approach was discussed in a personal interview with Mr. Harshil Shah.