Introducción al Proyecto del Chatbot de Gipuzkoa

Visión General

El proyecto del **Chatbot de Gipuzkoa** es una solución interactiva basada en inteligencia artificial diseñada para responder preguntas sobre Gipuzkoa, una provincia en el País Vasco. Este chatbot utiliza el modelo de lenguaje GPT-3.5-turbo de OpenAI para generar respuestas informativas y precisas en euskera, aprovechando una base de datos de preguntas y respuestas predefinidas y generadas dinámicamente.

Objetivo

El principal objetivo de este proyecto es proporcionar una herramienta accesible y útil que pueda responder preguntas sobre diversos aspectos de Gipuzkoa, como su historia, cultura, geografía y otras características relevantes. La aplicación está diseñada para ofrecer una experiencia de usuario fluida y agradable, facilitando la interacción con el chatbot a través de una interfaz web intuitiva.

Paso 1: Descarga de los Audios:

En este primer paso del proyecto, nos enfocamos en la descarga de audios a partir de videos de YouTube. Este proceso requiere la instalación de una herramienta especializada y la configuración de un script que facilite la obtención y conversión de los archivos de audio.

Para comenzar, es necesario instalar la biblioteca `yt-dlp`, que es una herramienta eficiente para descargar videos y audios desde YouTube y otras plataformas similares. La instalación de esta biblioteca garantiza que tengamos acceso a las últimas actualizaciones y características disponibles. La biblioteca `yt-dlp` se debe instalar a través de un gestor de paquetes, como `pip`, asegurando que esté correctamente configurada en el entorno de desarrollo.

Una vez instalada la herramienta, el siguiente paso es importar las bibliotecas necesarias en nuestro script. Usaremos `yt_dlp` para manejar la descarga y conversión de los audios, y `os` para gestionar la creación y manipulación de archivos y carpetas en nuestro sistema operativo.

Luego, se define una función específica para descargar el audio de los videos de YouTube. Esta función toma dos parámetros: la URL del video y la carpeta de destino donde se guardará el archivo de audio. La función configura `yt-dlp` para extraer el audio en formato

MP3 con una calidad de 192 kbps. Esta configuración asegura que el archivo de audio descargado sea de alta calidad y compatible con la mayoría de los reproductores de audio.

Antes de iniciar la descarga, se debe preparar la carpeta de destino. Si la carpeta no existe, se crea para asegurar que todos los archivos de audio se almacenen de manera organizada y accesible. Esto evita problemas relacionados con la falta de directorios y garantiza que los archivos se guarden en el lugar adecuado.

Finalmente, se especifica una lista de videos de YouTube de los cuales se desea extraer el audio. Cada URL en la lista se procesa a través de la función de descarga definida, la cual convierte y guarda el audio en la carpeta previamente configurada. Este proceso automatiza la descarga y conversión de los audios, facilitando la recopilación de los archivos necesarios para los siguientes pasos del proyecto.

Paso 2: Transcripción de los Videos

En el segundo paso del proyecto, nos enfocamos en la transcripción de los audios extraídos de los videos. Este proceso implica varios sub-pasos que incluyen la instalación de librerías necesarias, el procesamiento de los archivos de audio, y la conversión de estos archivos en texto legible.

Primero, es necesario instalar las librerías específicas que facilitán el trabajo con audio y reconocimiento de voz. Estas incluyen `pydub` para la manipulación de archivos de audio, `speechrecognition` para la conversión de audio en texto, `noisereduce` para la reducción de ruido en los audios, `librosa` para el procesamiento avanzado de audio, y `soundfile` para guardar los archivos de audio. Asegurarse de tener estas librerías instaladas es esencial para llevar a cabo el procesamiento de audio de manera efectiva.

Una vez instaladas las librerías, se importan en el script. Cada biblioteca cumple una función específica en el flujo de trabajo: desde la reducción de ruido hasta la transcripción del contenido de audio.

A continuación, se definen varias funciones para manejar el audio de manera eficiente. La primera función se encarga de reducir el ruido en un archivo de audio. Esta función carga el archivo, aplica técnicas de reducción de ruido y guarda una versión limpia en formato WAV, mejorando la calidad del audio para la siguiente etapa.

La segunda función convierte cualquier archivo de audio, como MP3, a formato WAV. El formato WAV es preferido para el procesamiento de audio debido a su calidad sin compresión, lo que facilita una mejor transcripción.

La tercera función divide el archivo de audio en segmentos de 30 segundos. Esta división permite manejar los archivos en trozos más pequeños, lo cual es útil para el proceso de transcripción, ya que facilita el manejo de grandes cantidades de audio.

La cuarta y última función utiliza un servicio de reconocimiento de voz para convertir cada segmento de audio en texto. Si el audio es difícil de entender o si ocurre un error, la función devuelve un mensaje que indica el problema.

Una vez definidas las funciones, se preparan las carpetas necesarias para almacenar los archivos procesados. Se crea una carpeta específica para los archivos WAV si aún no existe, asegurando que los archivos estén organizados y disponibles para el procesamiento.

Luego, se inicia el procesamiento de cada archivo de audio en la carpeta de audios. Para cada archivo MP3, se realiza la conversión a WAV, se reduce el ruido del archivo WAV, se divide el archivo en segmentos de 30 segundos y se transcribe cada segmento a texto. Finalmente, todas las transcripciones se combinan en una sola cadena de texto, y la transcripción completa de cada archivo se presenta en la salida del notebook.

Este paso automatiza el procesamiento completo del audio, desde la conversión y limpieza hasta la transcripción, facilitando la obtención de texto a partir de los archivos de audio extraídos de los videos.

Paso 3: Importación del PDF

En esta fase del proyecto, nos enfocamos en importar y procesar archivos PDF para extraer su contenido en formato de texto. Este proceso consta de varias etapas, cada una crucial para transformar el contenido del PDF en un formato utilizable.

El primer paso es instalar la librería necesaria para manejar archivos PDF, que en este caso es `PyPDF2`. Esta librería es fundamental para la extracción de texto de archivos PDF, facilitando la conversión de documentos en formato PDF a texto legible.

Una vez instalada la librería, se importan las herramientas necesarias en el script. Utilizamos `os` para gestionar rutas de archivos y `PyPDF2` para leer el contenido del PDF. Estas herramientas permiten la manipulación y lectura de archivos en el entorno de trabajo.

Para cargar un archivo PDF desde tu ordenador a Google Colab, empleamos la función `files.upload()` proporcionada por Google Colab. Esta función abre un cuadro de diálogo que permite seleccionar y subir archivos desde tu dispositivo local al entorno de Colab.

Después de subir el archivo, se obtiene una lista de los archivos subidos. Asumiendo que se ha subido un único archivo o que el archivo de interés es el primero en la lista, se toma el primer archivo para su procesamiento.

Definimos una función que convierte un archivo PDF en texto. Esta función abre el archivo PDF y utiliza el objeto `PdfReader` de `PyPDF2` para leer cada página del documento. Extrae el texto de cada página y lo concatena en una única cadena de texto. Posteriormente, guarda este texto en un archivo de texto (.txt) en una carpeta específica para su almacenamiento.

Creamos una carpeta destinada a almacenar los archivos de texto resultantes. Nos aseguramos de que esta carpeta exista antes de proceder a guardar los archivos de texto generados.

Finalmente, se llama a la función de conversión de PDF a texto para procesar el archivo PDF subido. El texto resultante se guarda en la carpeta de transcripciones, y se imprime la ruta del archivo de texto para facilitar el acceso a los resultados.

En resumen, este paso automatiza el proceso de conversión de archivos PDF a texto. Permite subir un archivo PDF, extraer su contenido textual y guardar ese contenido en un archivo .txt en una ubicación específica dentro del entorno de Google Colab. Además, se definen procedimientos para leer y guardar el contenido en la carpeta de destino, asegurando que los datos extraídos sean fácilmente accesibles y organizados.

Paso 4: Preguntas y Respuestas

En esta etapa del proyecto, nos enfocamos en la generación y gestión de preguntas y respuestas a partir de un archivo de texto, utilizando la API de OpenAI para facilitar la automatización de estos procesos. A continuación, se detalla cada parte del proceso:

Importación de Librerías

Primero, se importan las librerías necesarias para el funcionamiento del código. Utilizamos `openai` para interactuar con la API de OpenAI, y `os` para manejar variables de entorno y rutas de archivos. La clave API de OpenAI se establece directamente en el código para autenticar las solicitudes a la API.

Leer Texto del Archivo

Definimos una función llamada `irakurri_testua_fitxategitik`, la cual se encarga de leer el contenido de un archivo de texto y devolverlo como una cadena. Esta función es esencial para obtener el texto que se usará para generar preguntas y respuestas.

Generación de Preguntas

La función `galderak_sortu` se encarga de generar preguntas basadas en el contenido del texto. Utiliza la API de OpenAI para crear un número especificado de preguntas a partir de los primeros 1000 caracteres del texto. Esto permite obtener preguntas relevantes y bien formuladas sobre el contenido.

Generación de Respuestas

Una vez generadas las preguntas, la función `erantzunak_sortu` utiliza la API de OpenAI para proporcionar respuestas a cada una de las preguntas. Las respuestas generadas mantienen el estilo y personalidad del euskera, garantizando que sean coherentes con el contexto del texto original.

Proceso Principal

El proceso general incluye la lectura del archivo de texto, la generación de preguntas a partir del texto leído, y la creación de respuestas para esas preguntas. Finalmente, se imprimen las preguntas y respuestas generadas para su revisión y uso posterior.

Guardar en CSV

Para almacenar las preguntas y respuestas generadas, se utiliza la librería `csv`. La función `guardar_en_csv` abre un archivo CSV en modo escritura y escribe una fila de encabezados junto con cada pregunta y su respuesta en filas separadas. Se define el nombre del archivo CSV donde se guardarán estos datos y se confirma que la operación se completó exitosamente.

Convertir y Guardar en JSONL

El siguiente paso es convertir las preguntas y respuestas en formato JSONL, utilizando la librería `json`. Se define una lista de diccionarios con preguntas y respuestas en euskera, y se guarda cada entrada en el archivo `preguntas_respuestas_chat.jsonl` en formato JSON Lines (JSONL). Cada línea del archivo contiene un diccionario con un rol de "user" para la pregunta y "assistant" para la respuesta.

Cargar el Archivo JSONL en OpenAI

Se importa el módulo `os` para gestionar el sistema operativo y se configura la clave API de OpenAI como una variable de entorno. Se abre el archivo `preguntas_respuestas.jsonl` en modo lectura binaria y se carga en la plataforma de OpenAI utilizando la función `openai.File.create` para el ajuste fino (fine-tuning). Se imprime la respuesta de la API para confirmar que el archivo se ha cargado correctamente.

Configuración de Variables de Entorno

Finalmente, se configura la clave API de OpenAI como una variable de entorno llamada `OPENAI_API_KEY`. Esto permite que otras partes del código o aplicaciones puedan acceder a la clave sin necesidad de codificarla directamente en el código fuente. Se verifica y muestra si la clave API está correctamente configurada.

Paso 5: Fine-Tuning

Introducción

El ajuste fino (fine-tuning) es un proceso crucial en el desarrollo de modelos de inteligencia artificial, que permite personalizar un modelo preentrenado para que se adapte mejor a un conjunto de datos específico. En el contexto de OpenAI, este proceso implica ajustar un modelo base con datos adicionales para mejorar su rendimiento en tareas particulares.

Configuración Inicial

Para iniciar el proceso de ajuste fino, es necesario importar las librerías adecuadas y configurar la clave API de OpenAI. En este caso, se utilizan las librerías `openai` para interactuar con la API y `os` para manejar las variables de entorno. La clave API se configura mediante una variable de entorno llamada `OPENAI_API_KEY`, la cual autentica todas las solicitudes realizadas a la API de OpenAI.

Subida del Archivo de Datos

Una vez configurada la clave API, el siguiente paso es subir el archivo que contiene los datos para el ajuste fino. Este archivo, denominado `preguntas_respuestas_chat.jsonl`, debe estar en formato JSON Lines (JSONL), donde cada línea representa un objeto JSON. El archivo se abre en modo binario para ser cargado en la API de OpenAI. Esta operación permite que el archivo esté disponible para ser usado en el ajuste fino del modelo.

Creación del Trabajo de Ajuste Fino

Después de subir el archivo, se recibe una respuesta de la API que incluye un ID único para el archivo. Este ID es esencial para referenciar el archivo en el siguiente paso, que es la creación del trabajo de ajuste fino. Se especifica el archivo de entrenamiento y se selecciona el modelo base, como `gpt-3.5-turbo`, que se ajustará con los datos proporcionados. Además, se configuran parámetros adicionales como el número de épocas (`n_epochs`), el tamaño del lote (`batch_size`), y el multiplicador de la tasa de aprendizaje (`learning_rate_multiplier`), que afectan cómo se realiza el ajuste fino.

Verificación y Monitoreo del Trabajo

Una vez creado el trabajo de ajuste fino, es importante verificar su estado para asegurarse de que se está ejecutando correctamente. Se define una función que consulta el estado del trabajo periódicamente utilizando el ID del trabajo. El proceso de monitoreo implica revisar el estado del trabajo a intervalos regulares, generalmente cada 60 segundos. Este bucle se detiene cuando el trabajo alcanza un estado de éxito o falla. Los detalles del estado se imprimen para verificar el progreso.

Evaluación de Respuestas

Tras completar el ajuste fino, el siguiente paso es evaluar la precisión de las respuestas generadas por el modelo ajustado. Este proceso implica comparar las respuestas generadas con las respuestas esperadas. Se leen ambos archivos JSONL, uno con las respuestas esperadas y otro con las respuestas generadas. Se convierte cada línea en un objeto JSON para facilitar la comparación. La función de evaluación cuenta cuántas respuestas generadas coinciden con las respuestas esperadas y calcula la precisión como un porcentaje. También se imprimen las discrepancias encontradas para una revisión detallada.

Preprocesamiento del Texto

Antes de convertir el texto en vectores, es fundamental realizar un preprocesamiento para asegurar que el texto esté limpio y normalizado. Se lee el archivo de texto original y se aplica una función de preprocesamiento que elimina saltos de línea y espacios adicionales. Se utilizan expresiones regulares para reemplazar múltiples espacios en blanco por un solo espacio y se eliminan los espacios al inicio y al final del texto. Este proceso asegura que el texto esté en un formato adecuado para su posterior conversión en vectores.

Conversión del Texto a Vectores

Una vez que el texto está preprocesado, se divide en fragmentos manejables. Cada fragmento se convierte en un vector utilizando un modelo de embeddings, como `all-MiniLM-L6-v2` de la librería `sentence-transformers`. Este modelo convierte el texto en vectores de alta dimensión que pueden ser utilizados para diversas tareas de procesamiento y análisis.

Manejo de Datos con Chroma

Finalmente, se utilizan herramientas como Chroma para gestionar los vectores generados. Se importa la biblioteca `chromadb` para interactuar con los servicios de Chroma. Se crea una instancia del cliente de Chroma y se define un nombre para la colección donde se almacenarán los vectores. Si es necesario, se elimina una colección existente y se crea una nueva para asegurar que los datos se almacenan de manera organizada y actualizada.

Paso 6: Proceso de Indexación y Búsqueda con ChromaDB y Modelos de Embeddings

El siguiente proceso detalla cómo se lleva a cabo la indexación de documentos en una colección de ChromaDB utilizando embeddings generados por el modelo `paraphrase-MiniLM-L6-v2` de la biblioteca `sentence-transformers`. Este proceso es esencial para permitir búsquedas eficientes y relevantes en grandes conjuntos de datos textuales.

Importaciones y Carga del Modelo

El proceso comienza con la importación de las librerías necesarias. Se utiliza `chromadb` para interactuar con la base de datos Chroma, y `sentence-transformers` para trabajar con modelos de embeddings que convierten texto en vectores numéricos. Específicamente, el modelo `paraphrase-MiniLM-L6-v2` es cargado para generar estos vectores, lo que facilita la representación semántica del contenido textual.

Inicialización del Cliente de ChromaDB y Gestión de Colecciones

Una instancia del cliente de ChromaDB es creada para manejar las interacciones con la base de datos. A continuación, se define el nombre de la colección donde se almacenarán los documentos. Se intenta crear una nueva colección con este nombre; si ya existe una colección con el mismo nombre, se recupera la existente. Este manejo de excepciones asegura que el proceso sea robusto ante posibles errores de duplicación.

Funciones de Utilidad

Varias funciones de utilidad son definidas para facilitar el proceso de indexación. `read_text_file(filepath)` es una función que lee el archivo de texto y retorna una lista de líneas, cada una representando un documento. `get_vector_from_text(text)` toma un texto como entrada y genera su correspondiente vector de embeddings utilizando el modelo cargado. Finalmente, la función `index_documents(filepath, batch_size=100)` lee el archivo de texto, procesa sus líneas en lotes y las indexa en la colección de ChromaDB. Este enfoque en lotes optimiza la eficiencia de la indexación.

Indexación de Documentos

El proceso de indexación involucra leer el archivo de texto y procesar cada línea como un documento individual. Cada documento se asigna un identificador único y se genera un vector de embeddings para su contenido. Estos documentos, junto con sus vectores e identificadores, se añaden a la colección en lotes, mejorando la eficiencia del almacenamiento y la posterior recuperación.

Búsqueda de Documentos Similares

Una vez que los documentos están indexados, es posible buscar documentos similares en la colección. La función `search_documents(query_text, top_k=5)` convierte el texto de una consulta en un vector de embeddings y utiliza este vector para buscar en la colección. Los resultados incluyen identificadores de documentos y las distancias entre el vector de la consulta y los vectores de los documentos, donde una menor distancia indica una mayor similitud semántica.

Generación de Respuestas con GPT-3.5-turbo

Además de buscar documentos, el sistema puede generar respuestas basadas en los textos recuperados utilizando el modelo GPT-3.5-turbo de OpenAI. La función `generate_response(query_text, retrieved_texts)` crea un prompt para el modelo de GPT-3.5-turbo, que genera una respuesta contextualizada en base a los documentos recuperados. Esto permite la creación de aplicaciones que proporcionan respuestas informativas basadas en una colección de documentos específicos.

Ejecución del Pipeline y Consultas de Prueba

Para validar el funcionamiento del sistema, se ejecuta un pipeline con una serie de consultas de prueba. El pipeline coordina la búsqueda de documentos y la generación de respuestas, asegurando que cada consulta sea procesada correctamente y que las respuestas generadas sean relevantes y precisas. Cada consulta de prueba es procesada a través del sistema, y las respuestas generadas son evaluadas para verificar su pertinencia y calidad.

Conclusión

Este proceso combina técnicas de embeddings y búsqueda vectorial con generación de lenguaje natural para crear un sistema poderoso de indexación y búsqueda semántica. Esto es especialmente útil en aplicaciones donde se necesita encontrar información relevante en grandes bases de datos textuales de manera eficiente y precisa. La integración de ChromaDB con modelos de lenguaje como GPT-3.5-turbo amplía las capacidades del sistema, permitiendo no solo la búsqueda de documentos, sino también la generación de respuestas complejas y contextuales.

Paso 7: Descripción de la Interfaz Web para Chatbot utilizando Streamlit

Este fragmento de código describe la creación de una aplicación web interactiva que permite a los usuarios interactuar con un chatbot, utilizando Streamlit como plataforma principal. La aplicación incorpora un modelo de lenguaje avanzado (GPT-3.5-turbo) para generar respuestas y también ofrece funcionalidades para gestionar el historial de conversaciones. A continuación, se desglosan los componentes clave y funcionalidades del sistema:

Configuración Inicial

1. **Carga de Variables de Entorno:** Utilizando la biblioteca `dotenv`, se cargan las variables de entorno desde un archivo `.env`, que típicamente incluye configuraciones sensibles como claves API.

2. **Configuración de la API de OpenAI:** La clave API de OpenAI es configurada para autenticar las solicitudes, permitiendo el acceso a los modelos de lenguaje de OpenAI, como GPT-3.5-turbo.

Carga y Gestión de Datos

1. **Pares de Preguntas y Respuestas (QA Pairs):** La función `load_qa_pairs(filename)` carga un archivo JSONL que contiene pares de preguntas y respuestas, almacenándolos en un diccionario. Este diccionario sirve como una base de datos de respuestas predefinidas.

2. **Generación de Respuestas con GPT-3.5-turbo:** La función `get_gpt_answer(prompt)` utiliza el modelo GPT-3.5-turbo para generar respuestas basadas en un prompt proporcionado.

3. **Selección de Respuesta:** La función `get_answer(question)` busca en el diccionario de pares predefinidos para encontrar una respuesta a una pregunta dada. Si no se encuentra una respuesta, se utiliza GPT-3.5-turbo para generar una respuesta nueva.

Historial de Chats

1. **Guardado de Historial:** La función `save_chat_history(chat_id, messages)` almacena el historial de un chat en un archivo JSON, lo que permite su persistencia y recuperación futura.

2. **Carga de Historial:** `load_chat_history()` recupera el historial de chats almacenado, permitiendo al usuario acceder a conversaciones pasadas.

3. **Eliminación de Historial:** `delete_chat_history(chat_id)` permite eliminar un chat específico del historial.

Configuración de la Interfaz de Usuario

La interfaz de usuario está construida utilizando Streamlit, una herramienta que facilita la creación de aplicaciones web interactivas en Python.

1. **Sidebar:**

   - **Botón para Nuevo Chat:** Permite a los usuarios iniciar una nueva conversación, almacenando el historial del chat anterior si es necesario.

   - **Historial de Chats:** Muestra una lista de chats anteriores, permitiendo al usuario seleccionar, visualizar o eliminar chats.

2. **Área Principal:**

   - **Mostrar Mensajes:** Presenta el historial del chat actual de manera visualmente agradable.

- **Entrada de Usuario:** Campo de texto donde los usuarios pueden ingresar sus preguntas.

- **Botón de Envío:** Envía la pregunta y muestra la respuesta generada por el chatbot.

Estilos Personalizados

Se utiliza CSS para definir estilos personalizados para varios elementos de la interfaz, incluyendo el contenedor del chat, los mensajes del usuario y del asistente, así como los botones y campos de texto, mejorando la experiencia del usuario.

Funciones Adicionales

1. **new_chat():** Inicia un nuevo chat y guarda el historial del chat actual si es necesario.

2. **select_chat(chat_id):** Carga un chat específico desde el historial, permitiendo al usuario continuar conversaciones anteriores.

Ejemplo de Funcionamiento:

Inicialización

El sistema se inicializa cargando la clave API de OpenAI y los pares de preguntas y respuestas predefinidos. La interfaz de usuario se configura y despliega utilizando Streamlit, con opciones para iniciar, seleccionar y eliminar chats.

Interacción del Usuario

El usuario puede interactuar con el sistema escribiendo preguntas en un campo de texto. El sistema intenta responder utilizando respuestas predefinidas o, si no encuentra una correspondencia, generando una respuesta con GPT-3.5-turbo.

Gestión de Chats

Los chats son almacenados en un archivo JSON, lo que permite a los usuarios crear nuevos chats, seleccionar chats previos, o eliminar chats, ofreciendo una experiencia de usuario fluida y continua.

Integración con Ngrok

Para hacer la aplicación accesible desde fuera de la red local, se utiliza Ngrok. Este servicio crea un túnel seguro que expone la aplicación de Streamlit a través de una URL pública. Se configura un nuevo token de autenticación y se inicia un túnel en el puerto 8501, lo que facilita la demostración y el acceso remoto a la aplicación.

Esta arquitectura proporciona una interfaz web intuitiva para interactuar con un chatbot, aprovechando tanto respuestas predefinidas como generadas por un modelo de lenguaje avanzado, y es accesible a través de Internet gracias a la integración con Ngrok.