En este documento se explicará cada archivo que estará presente en la carpeta del programa al ejecutarlo. Algunos archivos no estarán presentes antes de la primera ejecución por lo que se recomienda ejecutar correctamente el programa siguiendo las instrucciones de instalación antes de leer la documentación provista en este documento.

---

app.py

Este código está diseñado para crear una aplicación web interactiva que ayuda con información generar sobre gatos domésticos, usando Streamlit. La aplicación permite a los usuarios interactuar con un chatbot llamado “Miaumis”, que responde preguntas sobre el cuidado de gatos domésticos.

Primero, se configuran las librerías necesarias para cargar imágenes y ejecutar funciones del chatbot. La interfaz de la aplicación muestra una imagen de bienvenida y un título centrado, para hacer la experiencia más visual y atractiva.

La interfaz está organizada en columnas, colocando el título y la imagen del chatbot en el centro. También incluye un botón para borrar el historial de conversaciones, que borra tanto los mensajes visibles como la memoria del chatbot, permitiendo comenzar una nueva conversación desde cero.
El sistema de mensajes guarda cada interacción, mostrando tanto las preguntas del usuario como las respuestas del bot en formato de chat, usando emojis para distinguir entre ambos. El historial de mensajes se almacena en una variable de sesión, asegurando que el contexto de la conversación se mantenga incluso al actualizar la página.

Para mejorar la experiencia, la respuesta del bot se presenta de manera progresiva, simulando una respuesta en tiempo real al mostrar cada palabra con un pequeño retraso. Esto imita la sensación de estar "escribiendo", lo cual puede hacer que la interacción se sienta más natural y dinámica.

---

main.py

Este bloque de código configura y ejecuta el backend del chatbot "Miaumis", usando herramientas avanzadas de procesamiento de lenguaje natural, memoria conversacional y búsqueda de información específica sobre el cuidado de gatos. A continuación, se explica cada sección:

Primero, se cargan variables de entorno utilizando la librería ‘dotenv’, que permite gestionar configuraciones y claves de API sensibles desde un archivo externo ‘.env’. Esto garantiza que el chatbot tenga acceso a la información necesaria para interactuar con las APIs de OpenAI y otras herramientas sin exponer información confidencial en el código.

Luego, se configura la función de embeddings y se inicializa el modelo de lenguaje ‘ChatOpenAI’, que emplea GPT-3.5 con una temperatura 0 para mantener la precisión y consistencia en las respuestas. La función de embeddings convierte las preguntas del usuario en vectores numéricos, facilitando la búsqueda semántica en la base de datos y ayudando a Miaumis a localizar información relevante.

El siguiente paso es la configuración de la base de datos Chroma, que almacena

información sobre el cuidado de gatos en un formato de vectores, permitiendo que el chatbot acceda a la información en función de la similitud semántica de las preguntas. Esto hace que las respuestas sean rápidas y relevantes, enfocadas en el tema específico del cuidado de gatos domésticos.

Se implementa una memoria conversacional para que Miaumis pueda recordar el contexto de las últimas cuatro interacciones del usuario. Esto se hace con un buffer que limita la cantidad de mensajes almacenados y permite que el bot mantenga coherencia sin verse afectado por un historial demasiado largo.

Además, se agregan herramientas adicionales, como REPL de Python que le permite al chatbot resolver preguntas que requieran cálculos o análisis de datos en tiempo real. También se añade una herramienta de búsqueda que permite acceder rápidamente a información sobre gatos desde la base de datos, y otra que busca videos en YouTube, ampliando las fuentes de información y mejorando la experiencia del usuario.

Para que Miaumis tenga un estilo específico, se configura un prompt que ajusta el tono del bot, haciéndolo amigable y accesible para el público argentino, limitando las respuestas a 200 palabras y usando términos propios de la región. Esto asegura que el chatbot mantenga una identidad coherente y adecuada para su audiencia, manteniendo un tono respetuoso y profesional, enfocado exclusivamente en temas relacionados con el cuidado de gatos.

Finalmente, se crea y configura el agente que ejecutará la lógica completa del chatbot, integrando el modelo de lenguaje, las herramientas de búsqueda y el prompt personalizado. La función ‘chatbot’ recibe las consultas del usuario junto con el historial de la conversación y retorna respuestas contextualizadas. Con esta estructura, Miaumis puede ofrecer información precisa y mantener un enfoque claro en su especialidad, que es ayudar a los amantes de los gatos con dudas sobre su cuidado y bienestar.

---

embeddings.ipynb

Este bloque de código se encarga de cargar, dividir y preparar un documento en formato PDF para ser utilizado por Miaumis. A través de un proceso llamado ‘chunking’, el documento se segmenta en partes más pequeñas, facilitando la búsqueda de información específica y mejorando el rendimiento en consultas que requieren acceso a datos específicos sobre el cuidado de gatos. A continuación, se describe cada sección.

En primer lugar, se cargan las credenciales necesarias desde un archivo ‘.env’ usando la librería ‘dotenv’, para que el sistema tenga acceso a la clave de API de OpenAI. Esto es fundamental para operar el sistema de embeddings, que convierte texto en representaciones vectoriales, optimizando la búsqueda semántica.

Luego, se define la ruta del archivo PDF que contiene la información sobre gatos, y se utiliza ‘PyPDFLoader’ para cargar el documento en memoria. Esta herramienta permite leer el PDF y estructurar el contenido en texto, lo cual es clave para que el chatbot pueda analizar y responder preguntas sobre temas específicos que están en el PDF.

A continuación, se configura un ‘text splitter’ o divisor de texto, estableciendo el tamaño de cada segmento ‘chunk’ en 1000 caracteres, con un solapamiento de 100 caracteres entre

ellos. Este solapamiento permite que el final de un fragmento esté también presente al comienzo del siguiente, ayudando a mantener el contexto y coherencia entre los diferentes fragmentos. Así, si una pregunta del usuario abarca contenido que se divide en varias partes, el sistema tiene más probabilidad de entender y responder correctamente.

Una vez segmentado el documento en fragmentos, estos se almacenan en la base de datos Chroma, que organiza cada fragmento en vectores mediante la función de embeddings. Esta etapa es crucial, ya que permite que cada consulta del usuario se compare de manera eficiente con los fragmentos de información en la base de datos, logrando respuestas precisas basadas en la similitud semántica.

Finalmente, se imprimen tanto el contenido original del PDF como el número de fragmentos creados. Esto permite verificar que el documento se haya cargado y dividido correctamente. También se muestran el contenido y la longitud de cada fragmento para asegurar que el proceso de 'chunking' ha funcionado según lo esperado, manteniendo la integridad y la utilidad del contenido original. Esta organización facilita que Miaumis pueda acceder rápidamente a cualquier parte del documento cuando el usuario formule una pregunta.

---

requirements.ipynb

Esta lista de dependencias corresponde a un proyecto de Python que incluye una variedad de bibliotecas necesarias para implementar un chatbot avanzado, como Miaumis. A continuación, se destacan algunas de las bibliotecas y sus funciones clave dentro del proyecto:

- **aiohttp, fastapi, uvicorn, starlette**: Estas bibliotecas manejan la arquitectura del servidor y permiten el desarrollo de una API web en tiempo real. fastapi es el framework principal, mientras que uvicorn se utiliza para ejecutar el servidor de manera eficiente.
- **langchain, langchain_chroma, langchain_community, langchain-openai**: Estas bibliotecas permiten el procesamiento de lenguaje natural y el uso de modelos de lenguaje como ChatGPT. langchain facilita la integración de herramientas, la gestión de memoria conversacional y el procesamiento semántico, mientras que chromadb es útil para la indexación y recuperación de datos.
- **openai, tiktoken**: La biblioteca openai es clave para acceder a los modelos GPT de OpenAI, y tiktoken ayuda a gestionar los tokens en las consultas a estos modelos, especialmente útil para optimizar el uso de límites de tokens en las respuestas.
- **pydantic, python-dotenv**: pydantic valida y gestiona los datos que pasan por la aplicación, mientras que python-dotenv facilita la carga de variables de entorno, como claves de API, de forma segura.
- **PyPDFLoader, pypdf**: Estas bibliotecas permiten cargar y procesar documentos PDF, transformando el texto en datos útiles que el chatbot puede utilizar para responder preguntas específicas.
- **Chroma, chroma-hnswlib**: Estas herramientas administran la base de datos de embeddings, que es la estructura de datos vectorial que facilita la búsqueda rápida y precisa de información relevante.
- **streamlit**: streamlit crea una interfaz web simple y rápida para el chatbot, mejorando la experiencia del usuario y permitiendo una interacción visual en tiempo real.

- **tenacity, backoff**: Ambas son útiles para gestionar reintentos en la conexión con APIs o servicios externos, asegurando la estabilidad del chatbot incluso en caso de fallos temporales en la conexión.
- **youtube-search**: Esta biblioteca permite buscar videos en YouTube directamente desde el chatbot, ofreciendo al usuario recomendaciones de contenido multimedia relevante.

Esta configuración incluye todo lo necesario para crear un sistema robusto, con un chatbot que puede acceder y procesar grandes cantidades de información sobre el cuidado de gatos, tanto desde documentos locales como de fuentes externas.

.env

Para cargar la clave de API de OpenAI de manera segura en la aplicación, el usuario deberá configurarla a través de un archivo de entorno '.env'. Esto garantiza que la aplicación pueda acceder a la clave sin incluirla directamente en el código.
Primero, en el archivo '.env', el usuario debe definir su clave de API de OpenAI asignándola a la variable 'OPENAI_API_KEY'. El código de Python luego utilizará la biblioteca python-dotenv para cargar esta variable en el entorno. La clave se cargará automáticamente cada vez que se inicie la aplicación, permitiendo la autenticación en la API de OpenAI sin exponerla en el código.
Por ejemplo, el usuario debe crear un archivo .env en la raíz del proyecto y colocar en él la línea OPENAI_API_KEY="su_clave_aqui". Luego, el programa puede acceder a esta clave cargando el archivo de entorno con load_dotenv() y utilizando os.getenv("OPENAI_API_KEY") para obtener el valor de la clave al inicializar el cliente de OpenAI.
Este método garantiza que la clave esté disponible en tiempo de ejecución sin necesidad de modificar el código cada vez que se inicie la aplicación.

__pycache__

El directorio __pycache__ es una carpeta que se crea automáticamente por Python para almacenar archivos de bytecode compilado. Cuando se ejecuta un script de Python, el intérprete compila el código fuente en bytecode para mejorar la velocidad de carga y ejecución. Este bytecode se guarda en el directorio __pycache__ con una extensión .pyc, que es un archivo que contiene el código compilado.
El archivo main.cpython-311.pyc dentro de este directorio es el bytecode compilado del módulo main, que fue escrito en Python y luego compilado para la versión 3.11 del intérprete de Python, como indica la parte cpython-311 en el nombre del archivo. La ventaja de tener estos archivos es que, si el mismo script se ejecuta nuevamente y no ha habido cambios en el código fuente, Python puede cargar directamente el bytecode desde el archivo .pyc en lugar de volver a compilar el código, lo que mejora el rendimiento.
El uso de __pycache__ es completamente transparente para el usuario y se gestiona automáticamente por Python. En general, este directorio ayuda a optimizar el proceso de ejecución de scripts, manteniendo la eficiencia en el uso de recursos y el tiempo de carga.

---

docs

El directorio docs es una estructura de carpetas destinada a almacenar documentos relacionados con el proyecto, en este caso, contiene un archivo llamado gatos.pdf. Este archivo PDF contiene información relevante sobre el cuidado de gatos, sirviendo como fuente de datos que la aplicación puede utilizar para responder a las consultas de los usuarios.
Dentro del directorio docs, también se encuentra un subdirectorio llamado Chroma. Este subdirectorio está destinado a almacenar la base de datos utilizada por Chroma, que es una herramienta de almacenamiento y recuperación de datos en forma de vectores. En el interior de Chroma, se encuentra un archivo llamado chroma.sqlite3, que es la base de datos en formato SQLite que almacena la información procesada, así como los vectores generados a partir del contenido del PDF y otros datos relevantes.
Dentro del directorio Chroma, hay otro subdirectorio con un nombre único, por lo general es un código hash de este estilo, d8c66dd51-63b0-4462-b161-a15ac97a9d13, que se utiliza para organizar y almacenar datos específicos de una sesión o conjunto de datos particular. Este directorio contiene cinco archivos: data_level0.bin, header.bin, index_metadata.pickle, length.bin y link_lists.bin. Estos archivos son componentes esenciales para la operación de la base de datos y la gestión de los vectores.
data_level0.bin contiene los datos de nivel cero de la estructura de almacenamiento, mientras que header.bin incluye información sobre la estructura del archivo.
index_metadata.pickle almacena información sobre el índice de los datos, length.bin contiene datos relacionados con la longitud de los vectores, y link_lists.bin es utilizado para gestionar las conexiones o referencias entre diferentes elementos en la base de datos. En conjunto, estos archivos permiten que la aplicación acceda, almacene y recupere eficientemente la información necesaria para brindar respuestas a los usuarios.

---

env

El directorio env se crea cuando se ejecuta el comando ‘python -m venv env’ y representa un entorno virtual en Python. Un entorno virtual es un espacio aislado que contiene su propio intérprete de Python, así como bibliotecas y dependencias instaladas, independientes del sistema operativo y otros proyectos. Esto permite trabajar en proyectos con versiones específicas de paquetes sin interferir con otras configuraciones en el mismo equipo.
Después de crear el entorno virtual con ‘python -m venv env’, el siguiente paso es activarlo. En sistemas basados en Windows, esto se hace ejecutando el comando ‘env\scripts\activate’ (o ‘source env/bin/activate’ en sistemas Unix o MacOS). Activar el entorno virtual redirige los comandos de Python y pip para que funcionen dentro de este espacio aislado, lo que facilita la administración de las dependencias específicas de un proyecto. Al activarse, la terminal muestra el nombre del entorno para indicar que los comandos se ejecutarán en el entorno env.
Finalmente, al ejecutar ‘pip install -r requirements.txt’, Python lee el archivo ‘requirements.txt’ para instalar todas las bibliotecas listadas en él dentro del entorno virtual. Este archivo suele contener las versiones específicas de cada dependencia que requiere el proyecto, lo que garantiza que el entorno sea consistente en diferentes configuraciones y máquinas. Una vez instaladas, todas las bibliotecas estarán disponibles exclusivamente en el entorno env, lo que asegura que el proyecto use solo las dependencias y versiones especificadas en ‘requirements.txt’.

.streamlit

El directorio ‘.streamlit’ es una carpeta de configuración específica para aplicaciones desarrolladas con Streamlit. Este directorio contiene archivos de configuración que definen opciones de apariencia y comportamiento para la aplicación, personalizando la experiencia visual y funcional del usuario.
Dentro de ‘.streamlit’, el archivo ‘config.toml’ almacena configuraciones para el tema de la aplicación, en este caso usando formato TOML (Tom's Obvious, Minimal Language). En este archivo, la sección [theme] define aspectos visuales del tema. La línea ‘base="dark"’ indica que la aplicación usará un esquema de colores oscuro como base, lo cual es útil para proporcionar una interfaz visualmente atractiva en entornos de poca luz o para usuarios que prefieren interfaces oscuras. La línea ‘font="monospace"’ establece que la fuente de texto en la aplicación será de tipo monoespaciado, ofreciendo un estilo uniforme en el que cada carácter ocupa el mismo espacio horizontal, lo cual puede mejorar la legibilidad en algunos contextos.
Estas configuraciones permiten personalizar la apariencia general de la aplicación sin necesidad de cambiar su código. Además, el archivo ‘config.toml’ facilita a los desarrolladores modificar estos ajustes de forma centralizada, aplicándose automáticamente cada vez que se ejecuta la aplicación en un entorno donde exista este archivo de configuración.

DIAGRAMA DE ARQUITECTURA