UNIVERSIDAD PRIVADA DE TACNA

FACULTAD DE INGENIERÍA

Escuela Profesional de Ingeniería de Sistemas

INFORME

"Desarrollo de un Asistente Conversacional con IA para Recomendación de Contenido"

Curso: Sistemas de Información

Docente: Ing. Refugio Valdivia Vargas

Fernandez Villanueva, Daleska Nicolle (2021070308)

Tacna – Perú
2025

ÍNDICE

INFORME

TEMA: Desarrollo de un Asistente Conversacional con IA para Recomendación de Contenido

I. INFORMACIÓN GENERAL

- **Objetivos:**
  - Conocer los fundamentos de la creación de aplicaciones web interactivas con **Streamlit**.
  - Implementar un sistema de recomendación basado en contenido utilizando **Scikit-learn** para procesar y analizar un dataset real.
  - Integrar y controlar un Modelo de Lenguaje Grande (LLM) local (**Meta Llama 3**) utilizando la librería **LangChain** para la generación de respuestas en lenguaje natural.
  - Diseñar y construir un **Agente de IA** con herramientas especializadas para manejar diversas intenciones del usuario, aplicando los principios de **RAG (Retrieval-Augmented Generation)**.
  - Aplicar principios de **Programación Orientada a Objetos (POO)** para estructurar y organizar un proyecto de IA complejo.

- **Equipos, materiales, programas y recursos utilizados:**
  - **Hardware:** Computadora con sistema operativo Windows 10/11, procesador multi-núcleo y preferiblemente una GPU NVIDIA con soporte CUDA para la aceleración del modelo de IA.

  - **Software y Entorno de Desarrollo:**
    - **Python 3.10 o superior.**
    - **Visual Studio Code** como editor de código.
    - **Terminal (PowerShell/CMD)** para la ejecución de comandos.

  - **Librerías Principales de Python:**
    - **Streamlit:** Para la creación de la interfaz web interactiva.

- **Pandas:** Para la carga, manipulación y limpieza de datos.
- **Scikit-learn:** Para la implementación del motor de recomendación (TF-IDF y Similitud del Coseno).
- **LangChain y LangChain Community:** Para orquestar el LLM, crear prompts y la arquitectura de Agente.
- **llama-cpp-python:** Para ejecutar el modelo Llama 3 de forma eficiente en hardware local.
- **fuzzywuzzy y python-Levenshtein:** Para la búsqueda de texto aproximada y flexible.
- **Plotly Express:** Para la generación de los gráficos del dashboard.

- **Recursos de IA y Datos:**
  - **Modelo LLM:** Meta-Llama-3-8B-Instruct-v2.Q3_K_M.gguf (Modelo de 8 mil millones de parámetros cuantizado para ejecución local).
  - **Dataset:** netflix_titles.csv obtenido de Kaggle, conteniendo metadatos de más de 8,000 películas y series.

II. MARCO TEÓRICO

Streamlit: Es un framework de código abierto en Python que permite a los desarrolladores crear y compartir aplicaciones web para proyectos de Machine Learning y ciencia de datos de forma rápida y con pocas líneas de código. Transforma scripts de Python en interfaces interactivas.

Sistema de Recomendación Basado en Contenido: Es una técnica de Machine Learning que sugiere ítems a un usuario basándose en las características de los ítems que al usuario le han gustado en el pasado. En nuestro caso, se utiliza TF-IDF (Term Frequency-Inverse Document Frequency) para convertir la descripción, género, director y reparto de una película en un vector numérico. Luego, la Similitud del Coseno mide el ángulo entre estos vectores para determinar qué tan "parecidas" son dos películas.

LangChain Agents y Tools: Es una arquitectura avanzada para construir aplicaciones con LLMs. Un Agente es el "cerebro" (el LLM) que, en lugar de responder directamente, utiliza un conjunto de Herramientas (Tools). Cada herramienta es una

función de Python que realiza una tarea específica (ej: buscar en una base de datos, hacer un cálculo). El Agente decide qué herramienta usar basándose en la pregunta del usuario, ejecuta esa herramienta y luego formula una respuesta final con la información obtenida.

RAG (Retrieval-Augmented Generation): Es un paradigma de IA donde un LLM no responde solo con el conocimiento con el que fue entrenado, sino que primero Recupera (Retrieval) información relevante de una fuente de datos externa (nuestro .csv). Luego, Aumenta (Augmented) su conocimiento con esa información y finalmente Genera (Generation) una respuesta mucho más precisa y actualizada. Nuestro bot utiliza RAG cuando busca los detalles de una película para responder preguntas como "¿de qué trata?".

III. PROCEDIMIENTO

Paso 1: Configuración del Entorno y Dependencias

- **Creación del Proyecto:** Se estableció una carpeta de proyecto para contener todos los archivos necesarios.

- **Instalación de Librerías:** Se procedió a instalar todas las dependencias de Python requeridas para el proyecto. Esto se realizó mediante la ejecución de un único comando en la terminal que instaló streamlit, pandas, plotly, scikit-learn, langchain, llama-cpp-python y fuzzywuzzy.

- **Gestión de Recursos:** Se descargaron los dos activos principales: el dataset netflix_titles.csv y el modelo de lenguaje Meta-Llama-3-8B-Instruct-v2.Q3_K_M.gguf, y se ubicaron en la raíz de la carpeta del proyecto para que el script app.py pudiera acceder a ellos.

Paso 2: Implementación de la Lógica de Carga y Preprocesamiento de Datos

- **Carga de Datos:** Se implementó la función load_data_from_csv(filepath) que utiliza la librería **Pandas** para leer el archivo netflix_titles.csv en un DataFrame.

- **Limpieza de Datos:** Dentro de la misma función, se realizó una limpieza básica de los datos, rellenando valores nulos en columnas clave como director, cast, country y rating con el valor "Unknown". Además, se eliminaron las filas que no contenían información esencial en las columnas description, listed_in o title.

- **Cacheo de Datos:** Se aplicó el decorador @st.cache_data a esta función. Esto le indica a Streamlit que ejecute esta operación de lectura de archivo solo una vez y guarde el DataFrame resultante en memoria caché, optimizando significativamente el rendimiento de la aplicación en recargas posteriores.

Paso 3: Desarrollo del Motor de Recomendación (Modelo 1)

- **Función de Entrenamiento:** Se creó la función train_recommender_model(_df), la cual es responsable de construir el modelo de recomendación basado en contenido.

- **Ingeniería de Características:** Se creó una nueva columna en el DataFrame llamada soup, que concatena el contenido de las columnas listed_in, director, cast y description. Esto genera un "supertexto" que representa el ADN de cada película.

- **Vectorización:** Se utilizó la clase TfidfVectorizer de **Scikit-learn** para convertir la columna soup en una matriz de vectores numéricos, limitando el vocabulario a las 4000 palabras más frecuentes para optimizar la velocidad.

- **Cálculo de Similitud:** Se aplicó la función cosine_similarity de Scikit-learn a la matriz TF-IDF para generar una matriz de similitud, la cual contiene un puntaje de "parecido" entre cada par de títulos del catálogo.

- **Cacheo de Recursos:** Se aplicó el decorador @st.cache_resource a esta función, ya que el modelo entrenado es un recurso computacionalmente costoso de generar. Esto asegura que el entrenamiento solo se realice una vez por sesión.

Paso 4: Implementación del Modelo de Lenguaje (Modelo 2)

- **Carga del LLM:** Se desarrolló la función load_llm_model(model_path). Esta función utiliza la clase LlamaCpp de LangChain para cargar el archivo .gguf en memoria.

- **Optimización de Parámetros:** Se configuraron los parámetros del modelo (n_ctx, max_tokens, temperature, n_threads) buscando un equilibrio óptimo entre la calidad de la respuesta y la velocidad de generación, con el objetivo de mantener las respuestas por debajo de los 40 segundos.

- **Cacheo de Recursos:** Al igual que el modelo de recomendación, esta función se decoró con @st.cache_resource para evitar la recarga del pesado modelo de 4 GB en cada interacción.

Paso 5: Diseño de la Lógica de Conversación (Clase

- **Programación Orientada a Objetos:** Se creó la clase CineBot para encapsular toda la lógica de interacción. El constructor (__init__) recibe los modelos ya entrenados y el "banco de intenciones", una serie de listas de palabras clave para identificar rápidamente la intención del usuario.

- **Métodos Especializados:** Se implementaron métodos para cada tarea: get_recommendations, get_movie_description, get_movie_genre y get_recommendations_by_genre. Cada método accede al DataFrame y a los modelos para realizar su tarea específica.

- **Lógica de Despacho** (Se creó un método principal get_response que actúa como un "router". Este método, a través de una serie de condicionales if/else en Python, analiza la entrada del usuario.

    - Si detecta un saludo o una pregunta específica sobre detalles (usando el banco de intenciones), llama al método correspondiente y devuelve una respuesta instantánea sin usar el LLM.

    - Solo si detecta una petición de recomendación basada en un título, procede a llamar al LLM para que redacte la respuesta final de forma creativa. Este enfoque minimiza las llamadas al LLM, que es la parte más lenta del proceso.

Paso 6: Construcción de la Interfaz Gráfica con Streamlit

- **Layout:** Se definió un layout de dos columnas con `st.columns()`.

- **Dashboard (Columna Izquierda):** Se utilizaron los componentes `st.metric` para mostrar datos numéricos y `st.plotly_chart` para visualizar los gráficos de barras y de pastel creados con **Plotly Express**.

- **Chat (Columna Derecha):** Se implementó la interfaz de chat.
    - Se usó st.session_state para mantener el historial de mensajes (messages) y el objeto cinebot persistentes entre las interacciones.

- Se iteró sobre st.session_state.messages para mostrar la conversación usando st.chat_message.

- Se utilizó st.chat_input para la entrada de texto del usuario, lo que garantiza que la caja para escribir se mantenga fija en la parte inferior.

- **Problema Encontrado y Solución:** Inicialmente, el chat no se actualizaba correctamente. Se resolvió implementando un flujo con `st.rerun()`. Cuando el usuario envía un mensaje, este se guarda en el historial y se llama a `st.rerun()`, lo que fuerza a la aplicación a redibujarse. En la nueva ejecución, el script detecta que el último mensaje es del usuario y procede a generar y mostrar la respuesta del bot, asegurando un flujo visual correcto y limpio.

IV. ANALISIS E INTERPRETACION DE RESULTADOS

V. ¿Qué indican los resultados?

Los resultados demuestran la viabilidad de crear un asistente de IA conversacional y robusto utilizando modelos de lenguaje locales. El sistema es capaz de interpretar una amplia variedad de preguntas en lenguaje natural, acceder a una base de datos de conocimiento (el CSV) y formular respuestas coherentes y contextuales.

VI. ¿Qué se ha encontrado?

a. La arquitectura de **Agente con Herramientas** es significativamente más fiable y flexible que un enfoque de un solo prompt. Permite separar la lógica de negocio (búsquedas en Python) de la generación de lenguaje (la tarea del LLM).

b. La técnica de **RAG** es fundamental para que el bot responda preguntas sobre datos específicos. Sin la capacidad de recuperar información del CSV, el LLM "alucinaría" o inventaría respuestas.
c. La **optimización de los parámetros del LLM** es crucial para el rendimiento en tiempo real. Un equilibrio entre el tamaño del contexto, los tokens máximos y la temperatura es clave para obtener respuestas rápidas y de calidad.
d. La **calidad del Prompt** es el factor más importante. Un prompt claro, con reglas estrictas y ejemplos (Chain of Thought), guía al modelo para que se comporte como se espera y evita la generación de texto no deseado.

CONCLUSIONES

Se ha construido con éxito un chatbot funcional y accesible vía web que resuelve una necesidad empresarial de descubrimiento y recomendación de contenido. El uso de una arquitectura de Agente con herramientas especializadas y el paradigma RAG demostró ser la solución más robusta para manejar una conversación fluida y responder preguntas basadas en datos específicos. Se cumplió el objetivo de aplicar Programación Orientada a Objetos para una estructura de código limpia y escalable. El proyecto integra con éxito cadenas de LangChain, mensajes de sistema y una Cadena de Pensamiento explícita, logrando un asistente de IA inteligente y eficiente.

RECOMENDACIONES

Para futuras mejoras, se podría implementar una base de datos vectorial (como FAISS o ChromaDB) para la herramienta de búsqueda de detalles. Esto permitiría búsquedas semánticas más avanzadas en las descripciones de las películas, en lugar de depender únicamente de la coincidencia de títulos, acercándose aún más a un sistema RAG puro y permitiendo preguntas como "¿qué películas tratan sobre viajes en el tiempo?". Adicionalmente, se podría explorar el uso de modelos LLM aún más pequeños y optimizados (quantizados a 2 o 3 bits) para reducir aún más el tiempo de respuesta en hardware con menos recursos.

WEBGRAFÍA y BIBLIOGRAFÍA

LangChain. (2024). *Agents Documentation*. Recuperado de https://python.langchain.com/docs/modules/agents/

Streamlit. (2024). *Streamlit Documentation*. Recuperado de https://docs.streamlit.io/

Hugging Face. (s.f.). *QuantFactory/Meta-Llama-3-8B-Instruct-GGUF-v2*. Recuperado de https://huggingface.co/QuantFactory/Meta-Llama-3-8B-Instruct-GGUF-v2

Kaggle. (2021). *Netflix Movies and TV Shows Dataset*. Recuperado de https://www.kaggle.com/datasets/shivamb/netflix-shows