IIC3103 - Taller de Integración
Departamento Ciencia de la Computación
Escuela de Ingeniería
Pontificia Universidad Católica

Enunciado Tarea 3

Objetivo

El objetivo de esta tarea es desarrollar una aplicación web que permita a los usuarios obtener explicaciones detalladas de películas, utilizando técnicas de **Retrieval-Augmented Generation** (RAG) y **Modelos de Lenguaje de Gran Tamaño** (LLMs). Se deberá implementar un proceso que incluya la el procesamiento de guiones de películas, la construcción de un **sistema RAG** para recuperar información relevante, la interacción con una **API de un LLM** proporcionada por el equipo docente, y la creación de una **interfaz web** para la interacción con los usuarios.

Trabajo a realizar

En esta tarea, deberán desarrollar una **aplicación web que funcione como un chatbot, permitiendo a los usuarios hacer consultas sobre películas** y recibir explicaciones detalladas. Para lograr esto, deberán:

- Realizar descargas de guiones de películas desde el sitio IMSDB (Internet Movie Script Database), respetando las políticas del sitio.
- Procesar y almacenar los guiones obtenidos, manejando posibles errores y asegurando la calidad de los datos.
- Generar una base de datos vectorial de fragmentos de guiones.
- Implementar un sistema RAG para recuperar fragmentos relevantes de los guiones en respuesta a las consultas de los usuarios.
- Interactuar con una API de un LLM proporcionada por el equipo docente, enviando el contexto recuperado y recibiendo respuestas generadas.

Luego, cada uno deberá crear una interfaz web intuitiva que permita a los usuarios interactuar con el chatbot, realizar consultas y visualizar respuestas.

Índice

Desarrollo de la tarea

Selección de guiones de películas

Deberán descargar guiones de películas desde el sitio IMSDB (https://imsdb.com/). Para esto, deberán:

- Identificar un conjunto de películas cuyos guiones estén disponibles en el sitio.
- Almacenar los guiones obtenidos en un formato adecuado (por ejemplo, archivos de texto plano) para su posterior procesamiento.
- Manejar posibles obstáculos técnicos, como estructuras HTML complejas o contenido dinámico.

Deberán seleccionar al menos 10 películas e inscribirlas en este link. No se podrán repetir más de 4 de las 10 películas inscritas en el formulario entre personas.

Adicionalmente, el Frontend deberá señalar claramente las películas seleccionadas con tal de poder hacer consultas de ellas.

Procesamiento y Almacenamiento de Guiones

En primer lugar, se deben descargar los guiones de las 10 películas seleccionadas. Esto se puede realizar manualmente.

Luego, cada uno deberá limpiar y preprocesar el texto, eliminando caracteres especiales, metadatos irrelevantes, etiquetas HTML u otro contenido no deseado, y asegurando un formato consistente para poder ser utilizado en los pasos siguientes.

Implementación de RAG

Usar como referencia este ejemplo: https://github.com/alanezz/rag-example

Dividir los guiones en fragmentos manejables (por ejemplo, escenas o párrafos) utilizando herramientas como LangChain Text Splitters.

Generación de Embeddings

Utilizar modelo “nomic-embed-text" v1.5 para convertir los fragmentos de texto en vectores. Este modelo está disponible en el servidor de la tarea. Para generar un vector de un texto, se debe hacer la siguiente llamada:

URL	POST tormenta.ing.puc.cl/api/embed

Cuerpo	`{`
	`  "model": "nomic-embed-text",`
	`  "input": "texto a vectorizar"`
	`}`
	
	El valor para model es fijo. Cualquier otro valor no funcionará.
	En el campo input, se debe insertar el texto a vectorizar

Respuesta	`{`
	`  "model":"nomic-embed-text",`
	`  "embeddings":[[...]],`
	`  "total_duration":1,`
	`  "load_duration":1,`
	`  "prompt_eval_count":1`
	`}`
	
	En el campo “embeddings” se devuelve el vector correspondiente. Este vector tiene una dimensionalidad de 768.
	Los campos total_duration, load_duration y prompt_eval_count contienen información de la ejecución realizada.

Más información sobre Nomic en:

- https://www.nomic.ai/blog/posts/nomic-embed-text-v1
- https://ollama.com/library/nomic-embed-text
- https://huggingface.co/nomic-ai/nomic-embed-text-v1.5

Almacenar en una base de datos vectorial

Crear una colección en una base de datos vectorial y guardar los embeddings generados. Esto le permitirá hacer generación mejorada por aumentación (RAG).

Recuperación de Información

Implementar funciones que, dada una consulta, realicen una búsqueda de similitud para recuperar los fragmentos más relevantes.

Asegurarse de que el contexto enviado al LLM cumple con los límites de longitud establecidos.

Interacción con la API del LLM

Configuración de la API

El equipo docente gestionará el modelo LLM en un servidor separado, accesible mediante una API RESTful.

La API está alojada en el servidor tormenta.ing.puc.cl. Este servidor utiliza Ollama para servir un modelo LLM basado en Llama3.2, de 3.21B parámetros.

Para interactuar con la API, se pueden usar los siguientes métodos:

- Generate a completion: genera una respuesta a un *prompt* con el modelo provisto.
- Generate a chat completion: genera el siguiente mensaje de un chat con el modelo provisto.

Ollama también expone una API basada en la API de OpenAI. Esta API es equivalente a los servicios anteriores, pero puede servir para compatibilidad de librerías u otros componentes.

Detalles técnicos del modelo

- Nombre modelo: integra-LLM
  - NOTA: Este modelo se debe usar en todas las llamadas a la API. El servidor no tiene otros modelos implementados.
- Algoritmo base: Llama3.2
- Parámetros: 3.21B
- Configuraciones específicas (documentación)
  - Temperatura (temperature): 6
  - Ventana de contexto (num_ctx): 2048 tokens
  - Ventana de repetición (repeat_last_n) : 10
  - Límite de opciones de siguiente token (top_k): 18

Restricciones de la API

La API cuenta con un rate-limit de 10 request/seg. Se debe respetar este límite para asegurar un correcto funcionamiento para todo el curso.

Todas las llamadas a la API están sujetas a un límite de tiempo de 120 segundos. En este sentido, si el cálculo del resultado excede este tiempo, se deberá reducir el contexto o los prompt correspondientes para garantizar la capacidad de completar los cálculos necesarios en ese plazo de tiempo determinado.

Integración de la API en el Backend

Implementar llamadas a la API del LLM desde el backend de la aplicación para generar respuestas basadas en el contexto proporcionado por RAG.

Manejo de Respuestas y Errores

Procesar las respuestas del LLM y manejar posibles errores o tiempos de espera de la API.

Implementar mecanismos de reintento o mensajes de error informativos para el usuario.

Flujo de interacción

1. Consulta del Usuario: El usuario realiza una pregunta sobre una película.
2. RAG Recupera Contexto: El sistema RAG busca y recupera fragmentos relevantes de los guiones almacenados.
3. Envío a la API del LLM: El contexto recuperado se envía junto con la consulta a la API del LLM gestionado por los docentes.
4. Generación de Respuesta: El LLM procesa el contexto y la consulta para generar una respuesta coherente y detallada.
5. Respuesta al Usuario: La aplicación muestra la respuesta generada al usuario en el frontend.

Desarrollo del Frontend

Respuesta al Usuario: La aplicación muestra la respuesta generada al usuario en el frontend.

Diseño de la Interfaz

Crear una interfaz tipo chat donde los usuarios puedan ingresar sus consultas. Mostrar las respuestas generadas por el LLM junto con las imágenes obtenidas de las APIs.

Tecnologías Sugeridas

- Frontend: React.js, Vue.js o Angular.
- Backend: Flask o FastAPI para manejar las solicitudes y comunicarse con el RAG y la API del LLM.

Funcionalidades Clave

- Campo de entrada para consultas.
- Área de visualización de respuestas con texto e imágenes (Afiches e información de películas)

- Manejo de estados de carga y errores.
- Diseño amigable para el usuario.

Versionamiento del código

El sitio y todo su código fuente deberá estar versionado en un repositorio en GitHub, creado en el classroom del curso:

- Link para creación de repositorio: https://classroom.github.com/a/O9GgJ1PR

El repositorio deberá contener:

- El código fuente del frontend
- El código fuente del backend
- Guiones de películas procesados

Entregables

Cada alumno deberá entregar, mediante un formulario publicado en el sitio del curso, las siguientes *url's*:

- URL sitio web.
- URL del repositorio a GitHub.

La evaluación de la tarea se realizará bajo una rúbrica a publicar. Se evaluará la completitud del sitio generado y la correctitud de la integración solicitada.

Fecha de entrega

La tarea debe entregarse antes del 7 de noviembre antes de las 18:00, y deberá estar desplegada y disponible durante el periodo de corrección (2 semanas).

Requisitos mínimos

Las tareas que no cumplan con las siguientes condiciones no serán corregidas y serán evaluados con la nota mínima:

- La página web deberá ser pública, accesible desde cualquier dispositivo conectado a internet.
- El código deberá estar versionado en su totalidad en un repositorio Git.

- El sitio implementado debe corresponder al código entregado. Para la revisión, más de una tarea serán corridas localmente por los ayudantes para comprobar el cumplimiento de este punto.
- En caso de que el código entregado no represente fielmente al sitio entregado, se calificará la tarea con la nota mínima.

Penalizaciones

Se descontarán 0,2 puntos de la nota de la tarea por cada hora de atraso en la entrega, contados a partir de la fecha estipulada en el punto anterior.

Cualquier intento de copia, plagio o acto deshonesto en el desarrollo de la tarea, será penalizado con nota 1,1 de acuerdo con la política de integridad académica del DCC.

Versiones del documento

- **Versión 1:**
  - 21/10/2024

Anexos

3Blue1Brown: How large language models work, a visual intro to transformers

Video que muestra cómo funciona un modelo de LLM. Explica cómo se entrena, como se generan los embeddings y por qué se usan relaciones vectoriales para esto, y otros conceptos como la temperatura del modelo.

Link: https://www.youtube.com/watch?v=wjZofJX0v4M

RAG + Langchain Python Project: Easy AI/Chat For Your Docs

Video que muestra una demo de un proyecto usando RAG para mejorar los resultados de consultas a un LLM, utilizando Python y Langchain.

En el video, usa como ejemplo la documentación de AWS, vectorizándola usando Embeddings y luego usando una BD vectorial para encontrar el mejor contexto relacionado con el prompt.

Link: https://www.youtube.com/watch?v=tcqEUSNCn8I