QUESTION ANSWERING OVER A DOCUMENT USING A LARGE LANGUAGE MODEL *(LLM)*

Ivan Yesid Sepulveda Paez
isepulveda@unal.edu.co

Redes de computadores
Octavio José Salcedo Parra

Universidad Nacional de Colombia, sede Bogotá.
Fecha: Noviembre 23 de 2023 - Semestre 2023 - II

RESUMEN

El documento propone un Modelo de Lenguaje de Máquina (LLM) para facilitar la comprensión de textos complejos en el ámbito académico. Utiliza la metodología CRISP-DM y busca mejorar la interacción con documentos extensos, permitiendo búsquedas contextuales y extracción de información relevante. Se detallan requisitos de usuario, dominio, funcionalidades y rendimiento. Además, describe el proceso de extracción de datos mediante Web Scraping para analizar títulos y párrafos en archivos HTML.

Palabras Clave: Modelo de Lenguaje de Máquina (LLM), Metodología CRISP-DM, Interacción con documentos, Búsquedas contextuales, Extracción de información, Requisitos de usuario, Web Scraping, Análisis de archivos HTML

Introducción

Algunos documentos y textos de diversa índole pueden ser bastante complejos de comprender para quien desea consultarlos. En ocasiones conocer esta información es necesario e importante, puesto que puede ser necesario para una gran cantidad de objetivos.

De esta manera, en muchas áreas es necesario para que se pueda tener un progreso sólido o un conocimiento pertinente, tales como en el ámbito investigativo, la academia, el mercado laboral y demás áreas relevantes del proceso.

Otro tema importante es el relacionado con el análisis de contextos o comprensión de un texto: la comprensión individual de un concepto o tema dentro del contexto de nuestro lenguaje está sujeto a criterios personales e incluso puede verse afectado en ocasiones por ambigüedades. Teniendo presente esta información permitirá organizarse para llevar a cabo un proceso de comprensión efectivo.

Ahora bien, la información que consultamos puede tener múltiples inconvenientes, aunque útil, puede no ser lo suficientemente explicativa, ser demasiado larga, demasiado corta en conocimientos o explicaciones, poco comprensible, etc;

- Información muy extensa y redundante,

- información puntual que está muy dispersa dentro de un texto, entre otros aspectos.

Se hace necesario un método de interacción más amigable del texto para con el usuario en algunos casos hasta debe ser personalizada, por lo cual el diseño procesador de texto que interactúe con el usuario puntualmente en los temas de interés del mismo es una excelente alternativa para solucionar esta problemática.

Para esto, se decide implementar la metodología CRISP-DM (Cross-Industry Standard Process for Data Mining) es un enfoque ampliamente utilizado para proyectos de minería de datos.

Objetivos

- Mejorar la toma de decisiones, identificar patrones o descubrir conocimiento oculto en los datos.

- Crear el MML para estudiantes, profesores, personal académico, personal directivo y en general cualquier persona que esté interesada en la adquisición de conocimiento de un texto de una manera más efectiva.

- Facilitar la interacción con los documentos, brindando acceso rápido y eficiente a la información relevante contenida en ellos.

- Proporcionar funcionalidades que permitan identificar información relevante para el usuario sin la necesidad de buscar en los documentos o leer una gran cantidad de contenido.

- Realizar búsquedas basadas en palabras clave utilizadas por los distintos usuarios, permitiendo al LLM hacer búsquedas basadas en el contexto del documento, resolver preguntas específicas y dar resultados relevantes.

- Extraer información enriquecida relacionado con la temática de interés, relación entre conceptos, categorización de temas, citas y referencia a los documentos donde se encuentran.

- Permitir la interacción del tipo texto a texto, es decir que a partir de un texto de entrada se pueda generar un texto de salida con la respectiva respuesta a la consulta.

- Proporcionar una interfaz intuitiva para que cualquier usuario pueda interactuar con el aplicativo y la disposición ordenada de todos los elementos interactivos.

Recopilación de Requisitos

En el proyecto la se identificaron los siguientes requisitos necesarios para, poder crear el MML

1. usuarios:
   - Estudiantes
   - Profesores y personal Académico
   - Lectores
   - Contratistas o entidades
2. Entendimiento del dominio y los documentos:

Dominio:

- El dominio se encuentra en el ámbito académico y de aprendizaje

- Se manejan diversos tipos de información y documentos relacionados, como teorías, normas y reglamentos.

- La aplicación tiene como objetivo facilitar la interacción con estos documentos, brindando acceso rápido y eficiente a la información relevante contenida en ellos.

- La aplicación debe proporcionar funcionalidades que permitan identificar información relevante para el usuario sin la necesidad de buscar en los documentos o leer todas las normativas

3. Funcionalidades deseadas:

- Las búsquedas basadas en palabras clave utilizadas por los distintos usuarios, permitirá al LLM hacer búsquedas basadas en el contexto del documento, resolver preguntas específicas y dar resultados relevantes.

- La extracción de información enriquecida como entidades mencionadas, relaciones entre conceptos, categorización de temas, citas y referencia a los documentos donde se encuentran la información.

4. Interacción con el LLM:

- La interacción que se pretende utilizar es del tipo texto a texto, es decir que a partir de un texto de entrada se pueda generar un texto de salida con la respectiva respuesta a la consulta

5. Requisitos de interfaz de usuario:

- Requiere como mínimo una interfaz para la entrada y salida de texto bien sea un text panelo un text area, dependerá de cómo queda configurado después del entrenamiento el LLM

- Interfaz intuitiva para cualquier usuario pueda interactuar con el aplicativo y la disposición ordenada de todos los elementos interactivos

6. Requisitos de rendimiento y escalabilidad:

Las características mínimas para que se pueda ejecutar el LLM en un servidor son un procesador de mínimo dos núcleos a 2Ghz, 4 Gb de memoria RAM y el almacenamiento dependerá de la cantidad de datos a procesar. teniendo en cuenta esas condiciones y que debe responder a múltiples consultas de manera diaria se optan por las siguientes opciones

- El modelo se desplegó en :

i.Google colab: Entorno de desarrollo desplegado en nube con la capacidad de ejecutar scripts de python y sus características en la capa gratuita son 12 Gb RAM, procesador intel Xeon CPU Intel Xeon a 2.20 GHz y 100Gb de disco duro

7. Validación y retroalimentación:

- Pruebas de usabilidad, rendimiento, alta disponibilidad, eficiencia del algoritmo, manejo de errores entre otras

- Pruebas de usuario, para determinar el correcto funcionamiento, validar el correcto funcionamiento del LLM

Análisis y Preparación de datos

Para poder realizar el análisis de datos se necesita primero hablar del método de extracción, mediante el cual podremos entender qué tipos de datos se

recibirán, en este caso se obtienen mediante el proceso de **Web Scraping;**

- El proceso de **Web Scrapping,** el cual consta en la extracción de contenidos y datos presentes en sitios web mediante el uso de software, este proceso se desarrolla de forma automática en la mayoría de casos permitiendo el procesado y extracción de datos.

- Esta herramienta se considera extremadamente útil para la recopilación de datos en línea, y en la actualidad se utiliza con fines investigativos, de mercado y desarrollo de inteligencia artificial.

1. **Exploración de los datos.**

1.1 Extracción de datos: Se realiza el proceso de **"scraping"** en el cual se obtienen conjuntos de datos representados como archivos HTML, que contienen los contenidos de diferentes enlaces de interés, estos archivos son traídos desde la web como archivos mediante el siguiente código:

```
from os import path
from pathlib import PurePath

import os
import re
import requests

# Abrir el archivo 'urls.txt' en modo lectura
with open("urls.txt", "r") as fileLinks:
      # Leer todas las líneas del archivo y guardarlas en la lista 'urls'
      urls = fileLinks.readlines()

# Crear una lista 'pureUrls' para almacenar las URL limpias
pureUrls = []
for url in urls:
      # Eliminar los espacios en blanco iniciales y finales de cada URL y agregarla a 'pureUrls'
      pureUrls.append(url.strip())
      print(f"URL Obtenida: {url} \n")

# Crear el directorio donde se guardarán los archivos HTML
directory = "archivos_html"
os.makedirs(directory, exist_ok=True)
# Iterar sobre cada URL en 'pureUrls'
for url in pureUrls:
      # Obtener el nombre del archivo de la URL utilizando 'PurePath'
      file_name = PurePath(url).name
      # Limpiar el nombre del archivo eliminando los caracteres especiales
      clean_filename = re.sub(r"\W+", "", file_name)

      # Combinar el nombre limpio del directorio y el archivo con la extensión '.html' utilizando 'path.join'
      file_path = os.path.join(directory, clean_filename + ".html")

      print(f"file_name: {file_name} || file_path: {file_path}")
      text = ""

      try:
        # Realizar una solicitud GET a la URL
        response = requests.get(url)
        if response.status_code == 200:
            # Si la respuesta es exitosa (código 200), guardar el contenido en 'text'
            text = response.text
        else:
```

```
            # Si hay un error en la respuesta, guardar un mensaje de error en 'text'
            text = f"Error: {response.status_code} - {response.reason}"
        except (requests.exceptions.ConnectionError, requests.exceptions.Timeout) as exception:
            # Capturar excepciones de conexión o tiempo de espera y mostrar un mensaje de error
            print(f"Ha ocurrido un error: {exception}")

        # Escribir el contenido en el archivo especificado por 'file_path'
        with open(file_path, "w") as fileWriter:
            fileWriter.write(text)

        print(f"Archivo escrito con éxito en: {file_path}")
```

Bloque 1.1.1: Extracción de archivos HTML

1.2 Exploración de los datos: Se examina cada archivo HTML con el fin de identificar qué elementos y etiquetas HTML son las que contienen la información de identificación de la información; Particularmente se busca la siguiente información:

1. **C**ontenido de títulos, ya sea denotado por etiqueta tipo **header <H1>...<H6>**, o clases particulares del CSS**.**
2. **C**ontenidos tipo párrafo, denotados por la etiqueta **Paragraph <p>**

Todos estos elementos se encuentran principalmente en la clase body de los HTML por lo cual se traba a partir del body de los HTML, en este caso se utilizará como ejemplo un documento de texto acerca del reglamento de la universidad:

- **HTML obtenidos:** Carpeta de HTML

```
<body><META name=Generator content="Microsoft Word 97"><B><FONT face=Arial>

<P align=center>RESOLUCI&#65533;N 235 DE 2009</P>

<P align=center>"Por la cual se reglamenta la admisi&#65533;n de exalumnos de la Universidad Nacional de Colombia"</P>

<P align=center>LA VICERRECTORA ACAD&#65533;MICA DE LA UNIVERSIDAD NACIONAL DE COLOMBIA</P>

<P align=center>En uso de sus atribuciones legales y</P>
```

Bloque 1.2.1: Ejemplo de elementos HTML obtenidos

Además, debido a que se está trabajando en el conjunto de caracteres UTF-8 , se debe verificar donde existan elementos no reconocidos, los cuales corresponden a elementos tales como tildes, diéresis y virgulillas. Por lo cual, se realiza la extracción de los párrafos en un archivo de texto plano.

```
import os
import re
from bs4 import BeautifulSoup

# Directorio que contiene los archivos HTML
```

```
directory = "archivos_html"

# Obtener la lista de archivos en el directorio
file_list = os.listdir(directory)

# Diccionario para almacenar los resultados
parsed_dict = {}

# Iterar sobre los archivos en el directorio
for file_name in file_list:
    # Combinar la ruta del directorio con el nombre de archivo
    file_path = os.path.join(directory, file_name)

    # Verificar si el elemento en el directorio es un archivo
    if os.path.isfile(file_path):
        # Abrir el archivo y realizar las operaciones deseadas
        with open(file_path, 'r') as file:
            # Leer el contenido del archivo
            content = file.read()

            # Imprimir información del archivo actual
            print(f"Examinando el Archivo: {file_name}\n")

            # Crear el objeto BeautifulSoup
            soup = BeautifulSoup(content, "html.parser")

            # Encontrar todos los elementos <p>
            paragraphs = soup.find_all('p')

            # Obtener los textos de los párrafos
            paragraphs_text = [p.text for p in paragraphs]

            # Agregar la lista de párrafos al diccionario
            parsed_dict[file_name] = paragraphs_text

# Nombre del archivo de salida
output_file = "resultadoCrudo.txt"

# Guardar el diccionario en un archivo de texto
with open(output_file, "w") as file:
    for file_name, paragraphs in parsed_dict.items():
        file.write(f"Archivo: {file_name}\n")
        file.write("Contenido:\n")

        # Escribir cada párrafo en el archivo de salida
        for paragraph in paragraphs:
            # Eliminar caracteres no alfanuméricos utilizando expresiones regulares
            paragraph = re.sub(r"\W+", " ", paragraph)
            file.write(f"{paragraph}\n")

file.write("-----------------------------\n")

# Imprimir mensaje de confirmación
print(f"El diccionario se ha guardado en el archivo: '{output_file}'.")
```

Bloque 1.2.1: Extracción de párrafos

1.3 Limpieza de los datos: Después del proceso de exploración, se identifican los siguientes aspectos los cuales se deben trabajar sobre los datos para refinarlos y

generar un archivo con datos íntegros a nivel gramatical.

- **Enlace a los datos en crudo:** Datos en bruto

```
Archivo: docjspd_i101456.html
Contenido:

RESOLUCI□N 19 DE 2022
 28 de julio

 Por la cual se reglamenta la admisi□n a los programas curriculares de pregrado de la Universidad Nacional de Colombia

LA VICERRECTOR□A ACAD□MICA DE LA UNIVERSIDAD NACIONAL DE COLOMBIA
```

Bloque 1.3.1: Ejemplo del texto obtenido

Hacemos la limpieza de datos eliminado caracteres no unicode, se elimina cualquier separador o divisor obtenido en el paso anterior, y se genera un diccionario limpio de 7 listas donde cada lista posee las resoluciones, esto mediante el siguiente código:

```
import os
import unicodedata

def processer(fileName:str,divider:str):
      dictionary = {}
      with open(fileName,'r') as archive:
      lines = archive.readlines()

      i = 0
      while i < len(lines):
      line = lines[i].strip()
      if line.startswith("Archivo:"):
            key = line.split(":")[1].strip()
            dictionary[key] = []
            elif key is not None and not lines[i].startswith(divider) and not lines[i].startswith('Contenido:'):
            parragraph = lines[i].strip()
            if(parragraph and parragraph != divider):

dictionary[key].append(parragraph)

      i+=1

      return dictionary

def stringRegularizer(wordList:list):
      regularized = []
      for string in wordList:
            string = unicodedata.normalize('NFKD',string).encode('ASCII','ignore').decode('utf-8')
      string = string.lower().strip()
      string = string.title()

      regularized.append(string)

      regularizedSet = set(regularized)

      return list(regularizedSet)

def dictionaryCleaner(dictionary:dict):
      for key in dictionary:
      value = dictionary[key]
            new_value = stringRegularizer(wordList=value)
      dictionary[key] = new_value

      return dictionary

def saveDictionaryToFile(dictionary:dict, file_name:str):
      with open(file_name, 'w') as file:
            for key, values in dictionary.items():

file.write(f"Archivo:{key}\n")
            file.write(f"Contenido:\n\n")
```

```
            for value in values:
file.write(f"{value}\n")
            file.write("\n")
```

Bloque 1.3.2: Limpieza de los datos

Como resultado obtenemos por ejemplo:

```
file_name ="resultadoCrudo.txt"
divider = '------------------------------'

dictionary                                  =
processer(fileName=file_name,divider=divi
der)
dictionary                                  =
dictionaryCleaner(dictionary=dictionary)

for key in dictionary:
      print(f'(key:     {key},     value:
[{dictionary[key][:1]},...])\n')


output_file_name                            =
"resultadoProcesado.txt"
saveDictionaryToFile(dictionary,
output_file_name)
```

```
(key:     docjspd_i101456.html,     value:
[['Capitulo Iv'],...])

(key: docjspd_i103411.html, value: [['Que
En Sesion Extraordinaria Asincronica No
Presencial 01 De 2023 Realizada Entre El
13 Y El 16 De Enero De 2023 El Consejo
Superior Universitario Analizo La
Propuesta Presentada Por La Direccion
Nacional De Admisiones Y Decidio
Aprobarla'],...])
...
```

1.4 Características de los datos: En este conjunto de datos de ejemplo, lo presente es la información de las resoluciones, separada por los párrafos, en especial existen ciertos vínculos entre párrafos que poseen una relación de causalidad, o de obligación. Tales como los requisitos para admisión de estudiantes antiguos de la universidad, por eso, es que cada set de datos a pesar de estar procesado listo para las bases **Vector Store de Chroma** debe revisarse para la aplicación de los embeddings.

1.5 Posibles problemas y patrones interesantes: En este conjunto de datos el mayor problema es el encadenamiento de textos, la mayoría de párrafos de las resoluciones vienen en tuplas **(acción,condiciones)** por lo cual es de vital importancia preservarlas con el fin de mejorar las respuestas dadas por el LLM.

Modelado

Repositorio de GitHub: Repositorio

En esta fase seleccionamos y aplicamos las técnicas que consideramos adecuadas para el modelado de datos. Para obtener esas técnicas fueron necesarias 3 versiones. Expondremos el desarrollo de las mismas de forma consecutiva para evidenciar avances e inconvenientes. Por otro lado, diagramamos la realización del proyecto y su modelo en la siguiente imagen:

Figura 1: Flujo de construcción del modelo

A medida que avancemos, iremos describiendo paso a paso cada una de las etapas.

1.1 Seleccionando la técnica de modelado

Para seleccionar la técnica de modelado es necesario entender cómo se encuentran conformados los datos, en el caso de este proyecto los datos son una colección de documentos procesados en cadenas de texto.

Al requerir resultados fácilmente presentables y poco procesados debido al método de tratamiento que se genera, lo más indicado es hacer uso del modelado de temas. Con lo cual se pueden identificar los temas particulares de cada texto. **a pesar de que esta labor sea realizada por el modelo LLM.**

Respecto al modelo, este se encuentra “Prefabricado” por lo cual no se puede hablar sobre una técnica de modelado respecto a este.

Sin embargo, para el desarrollo del modelo se siguió un modelado iterativo, en el cual se trabajaron diferentes modelos en diferentes etapas en las cuales los mejores resultados fueron avanzados a etapas posteriores, es por esto que para el desarrollo se generaron 3 modelos de procesamiento completos de los cuales se utilizó el de mayor precisión y sencillez respecto a los demás. **(Modelos de prueba disponibles en el enlace a GitHub)**

1.1.1 Técnica de modelado

Para el texto, se tiene como opción trabajar El modelado de temas es una técnica de **Procesamiento del Lenguaje Natural (PNL)** que utiliza el reconocimiento de patrones y el aprendizaje automático para identificar los temas dentro de cada texto o documento que analiza, inferir grupos de temas a partir de los datos de texto en general y agrupar textos o documentos que contengan grupos temáticos similares.

En comparación con el análisis manual, el modelado de temas permite analizar rápidamente una gran colección de documentos de una sola vez. Por ejemplo, si necesita clasificar y organizar **500.000 documentos que contienen aproximadamente 750 palabras cada uno**, mediante el modelado temático, puede determinar que su colección de documentos contiene 12 grupos temáticos en total. A continuación, su modelo agrupa los documentos en función de sus grupos temáticos. ¿El resultado? En lugar de tener que procesar y analizar 375 millones de palabras (500.000 documentos X 750 palabras), puede basar su análisis en estos grupos temáticos. Esto reduce su análisis a 9.000 palabras (12 grupos temáticos X 750 palabras), que se analizan con mayor rapidez.

Debido a que no existen muchos modelos que funcionen con el lenguaje español, se debe trabajar con uno que tenga cierta compatibilidad , en este caso se trabajara con el modelo **Alpaca LoRA 7B** el cual es de los que tienen mayor compatibilidad:

```
[ ]  # Clonar el Repo
     ! git clone https://github.com/tloen/alpaca-lora.git
```

UNIVERSIDAD NACIONAL DE COLOMBIA

Figura 2: Clonación del repositorio de Alpaca

Alpaca LoRA 7B es un adaptador de baja calificación para **LLaMA-7b** ajustado en el conjunto de datos **Stanford Alpaca.** Es un modelo de lenguaje natural que se puede utilizar para tareas de procesamiento del lenguaje natural como la generación de texto y la traducción automática.

El modelo viene entrenando con los siguientes hiper parámetros:

- **Epochs:** 10 (cargar desde la mejor época)
- **Tamaño del lote:** 128;
- **Longitud de corte:** 512;
- **Tasa de aprendizaje:** 3e-4;

LLaMA-7b es un modelo de lenguaje natural basado en la arquitectura de transformador que se puede utilizar para tareas de procesamiento del lenguaje natural como la generación de texto y la traducción automática.

LoRA es un adaptador de baja calificación para LLaMA-7b ajustado en el conjunto de datos Stanford Alpaca. Un adaptador de baja clasificación es un tipo de adaptador que se utiliza para ajustar un modelo de lenguaje pre-entrenado a un conjunto de datos específico..

```
[ ] #Listar los contenidos
    %ls

    alpaca_data_cleaned_archive.json  generate.py
    alpaca_data_gpt4.json             lengths.ipynb
    alpaca_data.json                  LICENSE
    DATA_LICENSE                      pyproject.toml
    docker-compose.yml                README.md
    Dockerfile                        requirements.txt
    export_hf_checkpoint.py           templates/
    export_state_dict_checkpoint.py   utils/
    finetune.py
```

Figuras 3 y 4: Trabajo con Alpaca LoRA-7B en binarios

```
[ ]  # Definimos el modelo base, en este caso LLaMA 7B (AlpacaLoRA-7B-Hf)
     # y el archivo de pesos de guanaco.

     base_model_path = 'decapoda-research/llama-7b-hf'
     weights_path = "plncmm/guanaco-lora-7b"
```

Figura 5: Trabajo con la interfaz de Hugging Faces

```
from peft import PeftModel

base_model = LlamaForCausalLM.from_pretrained(
    base_model_path,
    load_in_8bit=True,
    device_map='auto',
)

model = PeftModel.from_pretrained(
    base_model,
    weights_path,
)
```

Figura 6: Construcción del modelo Peft Con Hugging Faces

Uno de los criterios más importantes que influyeron en la selección del modelo fue la capacidad de procesamiento de lenguaje español **(Más avanzada que otros lenguajes)**, además de que su uso es bastante intuitivo ya sea mediante manipulación de binarios cómo mediante el **PipeLine de Hugging Faces**.

Guanaco-LoRA es un modelo entrenado a partir de LLaMA y Alpaca, diseñado con la finalidad de entender el idioma Español. Este modelo fue entrenado a partir de la traducción del conjunto de datos de alpaca. En este caso, se utilizan los pesos de Guanaco para poder computar y mejorar las predicciones de la red neuronal.

```python
from langchain.llms import HuggingFacePipeline

pipe = pipeline(
    "text-generation",
    model=base_model,
    tokenizer = tokenizer,
    max_length = 256,
    temperature = 0.1,
    top_p = 0.75,
    top_k = 40,
    repetition_penalty=1.1,
    max_new_tokens = 256,
)

local_llm = HuggingFacePipeline(pipeline=pipe)
```

Figura 7: Ajuste de parámetros del modelo.

1.2 Generación del diseño de prueba

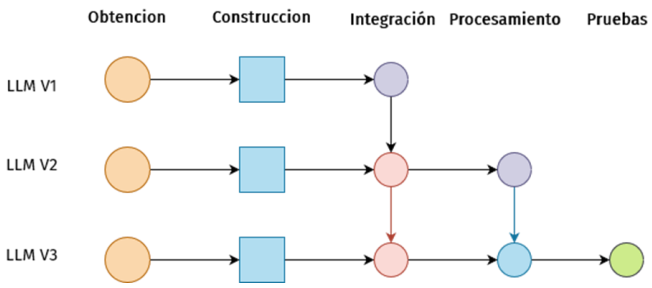


Figura 8: Generación iterativa del diseño.

A continuación se presentan las iteraciones que se realizaron para generar el modelo de prueba, se describen los elementos principales de cada iteración y las causales de finalización o descarte de los mismos:

LLM V1: Alpaca LoRA 7b

● Generado a partir de los binarios disponibles en los repositorios de Alpaca.

- Pre-Entrenado con Stanford Alpaca.
- Calibrado con Pesos base.
- Desplegado en interfaz web con puerto.
- **Razones de Descarte:** Debido a que se desplegaba como una interfaz web, comunicar al modelo usando LangChain o Chroma era más complejo debido a que se requiere usar un paquete de requisitos.

Este modelo corresponde a la primer iteración con alpaca, **(no se menciona a Groovy debido a su imprecisión)**; Este permitió sembrar las bases del procesamiento y ofreció uno de los mayores avances en el rendimiento a nivel máquina del modelo debido a que permite el ajuste con GPU de Cuda lo cual permitió que los siguientes modelos pudieran correr en las máquinas de colab sin colapsarlas.

LLM V2: AlpacaLoRA 7b + Guanaco + Hugging Faces

- Generado a partir de los binarios disponibles en los repositorios de Alpaca LoRA 7B, Guanaco en Hugging Faces.
- Pre-Entrenado con Stanford Alpaca.
- Calibrado con los pesos de Guanaco que es un conjunto de pesos y un dataset basado en Stanford Alpaca traducido al español.
- Desplegado en máquina.
- **Razones de Descarte:** A pesar de ser el modelo de mayor avance a nivel de construcción, no pudo integrarse a Chroma, pero sí a Langchain lo cual permitió comunicar peticiones al modelo pero obviando el contexto.

Este modelo es una reconstrucción usando Hugging Faces, lo cual simplificó procesos

tales como la obtención del modelo, los conjuntos de pesos. Debido a que corre en máquina, comunicar este modelo haciendo uso de LangChain.

En este modelo se realizaron las primeras pruebas contextuales, implementando el método de separación recursiva sobre una parte del conjunto de datos del contexto.

LLM V3: Alpaca LoRA 7b + Guanaco + Hugging Faces + LangChain y Chroma

- Generado a partir de los binarios disponibles en los repositorios de Alpaca LoRA 7B, Guanaco en Hugging Faces.
- Pre-Entrenado con Stanford Alpaca.
- Calibrado con los pesos de Guanaco que es un conjunto de pesos y un dataset basado en Stanford Alpaca traducido al español.
- Desplegado en máquina.
- Integró con éxito a LangChain y ChromaDB.

Este es el último modelo el cual surgió como una mutación al modelo V2, en este se utilizó PEFT para ajustar y construir el modelo.

También aumentó el número de hiperparametros para el control de respuestas, se implementó el Garbage Collector para liberar espacio en la GPU haciendo viable el procesamiento.

Se migró a una cadena de preguntas y respuestas de langchain para que el modelo pudiera generar mejores resultados.

1.3 Construcción del modelo

1.3.1 Tokenización e incrustación

Para la tokenización se utilizó **Llama Tokenizer,** con el fin de realizar la tokenización de una entrada, que en este contexto sería un texto. Durante este proceso, se divide el texto en secciones y se les asigna un valor numérico que puede representar su relevancia. Es importante destacar que el tokenizador utilizado está previamente entrenado para funcionar con el modelo LlaMA.

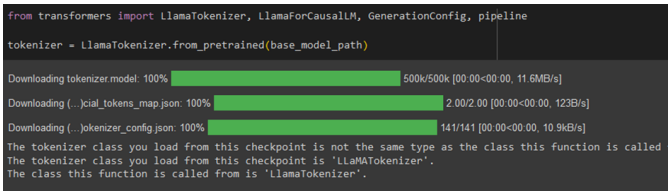


Figura 9: Descarga de los Tokenizadores

PETF, creación y entrenamiento del modelo

PETF, derivado de las siglas de **Parameter Efficient Fine Tuning** es una técnica utilizada en el procesamiento del lenguaje natural (NPL) para mejorar el rendimiento de los modelos de lenguaje pre entrenados en tareas específicas. Este se utiliza para abordar los problemas de la afinación de los modelos de lenguajes pre entrenados. Además, se centra en la eficiencia de los parámetros y utiliza menos parámetros entrenables para lograr un rendimiento comparable.

Para nuestro modelo usamos **PeftModel**, el cual es una clase de modelo de **HuggingFace** que abarca varios métodos **PETF**

La secuencia que seguimos fue la siguiente:

- Cargamos el modelo y pasamos como base un modelo de lenguaje Causal de LLaMA el cual nos **permite predecir cuál será el siguiente token con base en los que hay actualmente.**

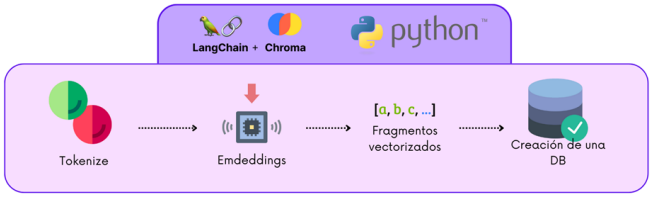
- Para reducir el tamaño del modelo y hacerlo más eficiente para correr en **Colab** lo cargamos en 8 bits
- Finalmente, pasamos los pesos con los cuales queremos que se entrene la red neuronal del modelo, a pesar que puede que las capas subyacentes cambian de pesos, es importante dar estos pesos ya que nos permitirá enfocar el modelo a la finalidad la cual en este punto es el entendimiento del proceso en español.

```
from peft import PeftModel

base_model = LlamaForCausalLM.from_pretrained(
    base_model_path,
    load_in_8bit=True,
    device_map='auto',
)

model = PeftModel.from_pretrained(
    base_model,
    weights_path,
)
```

```
Downloading (…)l-00008-of-00033.bin: 100% 405M/405M [00:03<00:00, 107MB/s]
Downloading (…)l-00009-of-00033.bin: 100% 405M/405M [00:01<00:00, 226MB/s]
```

Figura 10: Descarga de requisitos y Ajuste del modelo con los pesos.

1.3.2 Comunicación



- **ChromaDB** es una base de datos de incrustación de código abierto y nativa de IA .
  - Es la forma más rápida de crear aplicaciones de Python o JavaScript con memoria utilizando modelos de lenguaje grandes (LLM) .
  - La API principal consta de solo 4 funciones.
    - Configurar Chroma en memoria.
    - Crear una colección.
    - Agregar documentos a la colección y consultar.
    - Buscar resultados similares a lo que se consulta.

Aunque Chroma no es la única base de datos que permite esto, hay alternativas como **FAISS y Vectra.** Al ser un requerimiento, se eligió sobre otras bases.

Para este proyecto, se aprovecha la característica de **búsquedas por similaridad de Chroma**, lo cual permite cargar documentos, PDF y otros archivos. Sobre los cuales se pueden buscar resultados basados en el criterio de búsqueda.

- LangChain es un marco para desarrollar aplicaciones impulsadas por modelos de lenguaje. Este nos permite colocar un conjunto de métodos diseñados para conectarse e interactuar con modelos.
  - Posee ventajas como la percepción del contexto lo que le permite a los modelos interactuar con el ambiente.

- Cuenta con formas de conectar a diferentes fuentes de datos.

- Estas funcionalidades, y las múltiples integraciones de LangChain nos permiten brinda el contexto obtenido de los documentos para qu el modelo lo procese.
- Creamos un pipeline con el cual podemos ajustar y configurar parámetros del modelo, entr los más importantes tenemos:
- Comportamiento del modelo.
- Modelo a utilizar y tokenizador d procesamiento textual.
- Penalidad por repetición en la generación d respuestas.
- Máximo de tokens que pueden Generars para las respuestas.
- Longitud máxima de la entrada que procesa el Modelo.

Dadas estas características, pasamos todo esto a nuestro modelo local, puesto que al estar ya pre entrenado posee una forma de validar las respuestas que genera **(previa imagen).**

```
from langchain.llms import HuggingFacePipeline

pipe = pipeline(
    "text-generation",
    model=base_model,
    tokenizer = tokenizer,
    max_length = 256,
    temperature = 0.1,
    top_p = 0.75,
    top_k = 40,
    repetition_penalty=1.1,
    max_new_tokens = 256,
)

local_llm = HuggingFacePipeline(pipeline=pipe)
```

Figura 11: Ajuste hiper parámetros.

Acto seguido, hacemos uso de la función TextLoader para cargar el documento que contiene las respuestas en una variable para procesarla. Esto, debido a que el procesamiento debe hacerse sobre fragmentos del texto, ya que al habilitar la GPU en el ambiente, se pueden llegar a **ubicar gigas de memoria para el procesamiento sobrecargando el modelo y el entorno.**

```
from langchain.document_loaders import TextLoader
loader = TextLoader("/content/resultado.txt")
documents = loader.load()
```

Figura 12: Importación del documento.

Para modificar el documento en fragmentos, utilizamos un splitter de caracteres recursivo que iteramos sobre un conjunto de separadores. Para nuestro modelo, lo separamos en 100 caracteres o menos, utilizando len de Python

```
from langchain.text_splitter import RecursiveCharacterTextSplitter

text_splitter = RecursiveCharacterTextSplitter(
    separators=["\n",":",","],
    chunk_size=256,
    chunk_overlap=10,
    length_function = len,
    add_start_index = True
    )

documents = text_splitter.split_documents(documents)

print(f'Cantidad de fragmentos generados: {len(documents)}\n')
```

```
Cantidad de fragmentos generados: 864
```

Figura 13: Construcción de fragmentos

Ahora, desde LangChain creamos una cadena para la creación y respuesta de preguntas al modelo, particularmente preguntas de contexto

Figura 14: Creamos la cadena de consultas de LangChain y los Embeddings.

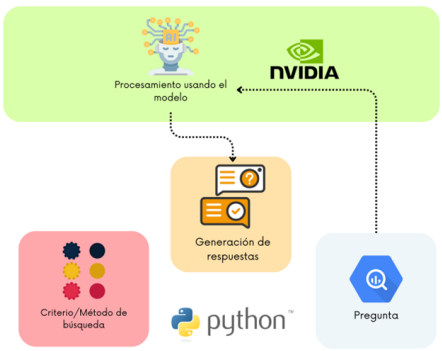
Como último paso en lo relacionado a la comunicación del modelo, llamamos un generador de Embeddings ajustado al modelo que estamos utilizando.

Cabe recordar que con los embeddings estamos tomando los fragmentos generados con Chroma y les estamos representando como **vectores de valores Reales de n-dimensiones** con el cual capturamos la semántica de cada fragmento.

```
from langchain.vectorstores import Chroma
vectorstore = Chroma.from_documents(documents, embeddings)
```

Figura 15: Creación de los Embeddings

1.3.3 Procesamiento, administración de memoria y consultas



Haciendo uso de **Nvidia CUDA** agilizamos el tiempo de respuesta y paralelismos parte del mismo en los núcleos de la GPU. También se hace uso del **Garbage Collector** para liberar almacenamiento de forma estática.

Ahora bien, se verifica que se obtengan resultados relacionados al contexto de la pregunta. Esto se realiza mediante una consulta de **"Similaridad"** a la base de vectores Chroma.

Figura 16: Consulta de contexto a la base de datos chroma.

En este punto hacemos el llamado al modelo haciendo uso de los resultados que obtuvimos de la búsqueda con Chroma y de la pregunta sobre la cual queremos una respuesta.

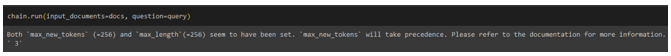


Figura 17: Consulta al modelo usando el contexto

Estando aquí, podemos hacer queries o consultas de prueba, sin contexto para medir la fiabilidad del modelo entrenado

```
from langchain import PromptTemplate, LLMChain

from langchain import PromptTemplate, LLMChain

user_input = input("Haz una pregunta: ")

prompt = PromptTemplate(
    input_variables=["question"],
    template=" {question}?")

gc.collect()
torch.cuda.empty_cache()

llm_chain = LLMChain(prompt=prompt, llm=local_llm)

print(llm_chain.run(user_input))
```

Figura 18: Consultas Extra al modelo fuera del contexto.

1.3.4 Parámetros de configuración

Utilizando la función **pipeline** de la biblioteca de **LangChain** , particularmente el pipeline de **Hugging Faces** para crear un pipeline de generación de texto. El modelo y el tokenizador utilizados en el pipeline se especifican con los argumentos **modelo y tokenizador**, respectivamente. Los siguientes argumentos son parámetros para controlar la generación de texto:

- **max_length:** La longitud máxima del texto generado.

- **temperature:** Controla la aleatoriedad del texto generado. Un valor más bajo da como resultado un texto más conservador.

- **top_p:** Utiliza el muestreo núcleo (nucleus sampling) para limitar la distribución de tokens a un subconjunto más pequeño. Un valor más bajo da como resultado un texto más conservador.

- **top_k:** Utiliza el muestreo superior-k para limitar la distribución de tokens a los k tokens más probables. Un valor más bajo da como resultado un texto más conservador.

- **repetition_penalty**: Penaliza las repeticiones en el texto generado. Un valor mayor que 1.0 desalienta las repeticiones.

- **max_new_tokens:** El número máximo de tokens que se generarán en una sola llamada al pipeline.

Evaluación

1.1 Evaluación de resultados

Para la ejecución de las pruebas se plantearon 10 preguntas respecto al documento de resolución de la universidad nacional de colombia, procesado por el LLM version_3, es el que permite ejecutar pruebas y partir de sus resultados se hicieron los siguientes descubrimientos sobre el modelo seleccionado:

UNIVERSIDAD NACIONAL DE COLOMBIA

Identificador	Pregunta
1	¿Qué dice la normativa de admisión a la Universidad Nacional de Colombia?
2	Cuál es la duración máxima para completar el proceso de admisión según la Resolución 037 de 2010?
3	¿Cuál es el proceso mediante el cual se adquiere la calidad de estudiante en la Universidad Nacional de Colombia según el Acuerdo?
4	¿Qué se requiere para mantener el estado de admitido en un programa de pregrado según el Acuerdo?"
5	¿Qué se reglamenta en la Resolución 19 de 2022?
6	¿Cuáles son los pasos a seguir para solicitar la admisión en la Universidad según el reglamento vigente?
7	¿Cuál fue el motivo por el cual se suspendió la aplicación de la prueba de admisión?
8	¿Cuáles son los requisitos específicos para la admisión en la Facultad de Ingeniería según la normativa vigente?
9	¿Cuántos cupos se asignan para la admisión en la Facultad de Ingeniería según la normativa vigente?
10	¿Cuál es la fecha límite para presentar la solicitud de admisión según la Resolución Rectoría 235 de 2009?

Identificador	Respuesta
1	La normativa de admisión a la Universidad Nacional de Colombia establece los requisitos para ser admitido en el programa de pregrado de la universidad y define el proceso de admisión.

2	1 año
3	El proceso para obtener la calidad de estudiante en la Universidad Nacional de Colombia se regula por el Acuerdo 008 de 2008. Según este acuerdo, los estudiantes deben completar un proceso de admisión y luego pasar un examen de admisión para ser admitidos a la universidad.
4	Para mantener el estado de admitido en un programa de pregrado según el Acuerdo, los estudiantes deben cumplir con los siguientes requisitos:\n1. Completar la etapa inicial del programa.\n2. Realizar bien en el examen de admisión.\n3. Responder a las necesidades específicas del programa curricular al que se haya admitido.\n4. Responda a las necesidades de disponibilidad de programas que se puedan ofrecer en la sede de presencia nacional .\n5. Ser aprobado por el reglamento transitorio de admisión al programa de pregrado de la Sede de Paz para el segundo periodo académico de 2022.
5	La Resolución 19 de 2022 establece los artículos 9, 10 y 11 de la Resolución 002 de 2014.
6	\n1. Completar todos los requisitos de admisión y prestar pruebas de idioma. \n2. Enviar una solicitud formal de admisión al departamento de admisión. \n3. Responder las preguntas de admisión y proporcionar cualquier documentación adicional que se pida. \n4. Recibir una respuesta de admisión del departamento de admisión.
7	La prueba de admisión se suspendió debido a que los estudiantes no estaban preparados para pasarla.
8	Los requisitos específicos para la admisión en la Facultad de Ingeniería según la normativa vigente incluyen un examen de ingreso, una prueba de inglés y un examen de matemáticas. Además, se espera que los candidatos tengan un puntaje mínimo de 1200 en el SAT o un puntaje mínimo de 30 en el ACT.

9	Según el artículo 1 del acuerdo, se establecen dos por ciento (2%) de los cupos disponibles para cada carrera que ofrece la universidad en su sede y secundarios para ser llenados por miembros de comunidades indígenas
10	La fecha límite para presentar la solicitud de admisión según la Resolución Rectoría 235 de 2009 es el 15 de mayo del año en curso.

Evaluación de calidad de generación de texto: Para el estudio de la calidad de las respuestas de las preguntas, se procede hacer un análisis cualitativo respecto a la coherencia del mensaje, y si en efecto usa el contexto del documento.

- Se observa que las preguntas **1,2,3,4,6,10** eventualmente usan el contexto de la información brindada al modelo de aprendizaje y citan correctamente el documento.

- El caso **5** cita los artículos pero no explica que reglamenta la resolución, el caso **7,8** la respuesta es incorrecta, pues la suspensión fue el covid 19, posiblemente mezcló el contexto con lo ya aprendido del modelo.

- Y la respuesta **9** está incompleta, sólo explica la admisión en un determinado caso.

Considerando las limitaciones de hardware y el entrenamiento del modelo, podemos decir que responde bastante bien, **siempre que la pregunta esté correctamente formulada**, en cambio una pregunta ambigua o con un contexto amplio, el modelo comete errores en su respuesta.

Prueba de rendimiento: Para ejecutar pruebas de rendimiento, se plantearon 10 preguntas con características diferentes,a algunas muy específicas respecto al contenido, otras más generales y otras donde debe comparar o analizar desde múltiples fuentes

Pregunta	**Tiempo de ejecución (segundos)**
1	11.07
2	1.66
3	20.06
4	35.69
5	10.64
6	18.65

7	6.09
8	22.62
9	12.84
10	9.2

- **Valor máximo:** 35.69 s
- **Valor mínimo:** 1.66 s
- **Tiempo promedio respuesta:** 14.85 s

Como se puede observar algunas preguntan demoran menos tiempo en ser procesadas, por ejemplo la **pregunta 4 preguntaba una lista de requerimientos, computacionalmente demora más en responder y es la respuesta con más caracteres generados,** en cambio una pregunta con un contexto claro demora menos tiempo en generar respuesta.

Respecto al funcionamiento general es lento ya que el hardware brindado para desplegar y hacer un modelo de aprendizaje es bastante limitado, ya que se utilizó la capa gratuita de colab.

1.2 Revisión de proceso

Durante el proceso de carga de librerías, selección del modelo, entrenamiento, y la validación de las preguntas se usaron diferentes metodologías para estimar la precisión del modelo, el rendimiento y la calidad de las respuestas según el contexto, y cómo esto afecta el desempeño del algoritmo.

Despliegue

El proceso de despliegue consta en la producción y puesta en línea del proyecto. Sin embargo, dependiendo del método de construcción del modelo se puede ofrecer dos alternativas principales de construcción.

1. **Construcción haciendo uso de docker.**
2. **Construcción mediante API.**

A continuación expondremos en qué consta cada una de las alternativas:

1. **Construcción mediante docker:**

1.1 Fundamentos: Docker es una plataforma de software que permite crear, probar e implementar aplicaciones rápidamente.

- Docker empaqueta software en unidades estandarizadas llamadas contenedores que incluyen todo lo necesario para que el software se ejecute, incluidas bibliotecas, herramientas de sistema, código y tiempo de ejecución.
- Con Docker, puede implementar y ajustar la escala de aplicaciones rápidamente en cualquier entorno con la certeza de saber que su código se ejecutará.

1.2 Construcción

Los contenedores de docker poseen internamente una infraestructura similar a una máquina virtual, particularmente el factor de poseer un subsistema linux. Lo cual representa una ventaja migratoria al momento de llevar código desde Google Collaboratory. Se proponen los siguiente pasos.

1.2.1 Construcción de una imagen docker.

- La construcción de la imagen docker debe incluir todas aquellas librerías que son necesarias para la correcta ejecución del proyecto (**Tensor Flow, Py Torch,Chroma,Langchain,etc).**

- Deberá instalarse un ambiente virtual (Venv) de **python 3.9** o superior para maximizar la compatibilidad con las librerías.

- Debe tenerse **git y wget** instaladas para el control dinámico de repositorios en caso de ser necesarios ya sea para la obtención de datos, o de código que permita hacer Scraping.

- Es vital que las máquinas virtuales sobre las cuales se construya la imagen posean una **interfaz de red abierta**, ya que puede permitirse el uso de paralelización para el despliegue y ejecución del modelo, a demás que se puede integrar una interfaz rest que permita el uso de solicitudes HTTP para interactuar con el modelo.

- Debe contarse con una arquitectura de procesadores conectados en paralelo, preferiblemente usando un esquema maestro-esclavo para paralelizar el procesamiento en estos (En caso de no contar con GPUs dedicadas) usando **OpenMPI.**

```
FROM python:3.11
RUN apt-get update && apt-get install -y wget
RUN pip install fire gradio transformers
git+https://github.com/huggingface/peft.git
sentencepiece accelerate bitsandbytes
langchain sentence_transformers chromadb
xformers
EXPOSE <port_number>
CMD ["python", "your_script.py"]
```

Bloque 1.2.2: Ejemplo 1 de imagen docker para el proyecto.

```
FROM nvidia/cuda:11.8.0-devel-ubuntu22.04
ARG DEBIAN_FRONTEND=noninteractive
RUN apt-get update && apt-get install -y \
      git \
      curl \
      software-properties-common \
      &&                  add-apt-repository
ppa:deadsnakes/ppa \
      && apt install -y python3.10 \
      && rm -rf /var/lib/apt/lists/*
WORKDIR /workspace
COPY requirements.txt requirements.txt
RUN                 curl                -sS
https://bootstrap.pypa.io/get-pip.py        |
python3.10 \
      &&  python3.10  -m  pip  install  -r
requirements.txt \
      && python3.10 -m pip install numpy
--pre  torch  --force-reinstall  --index-url
https://download.pytorch.org/whl/nightly/cu118
COPY . .
ENTRYPOINT [ "python3.10"]

version: '3'

services:
  alpaca-lora:
      build:
      context: ./
      dockerfile: Dockerfile
      args:
       BUILDKIT_INLINE_CACHE: "0"
      image: alpaca-lora
      shm_size: '64gb'
      command:    generate.py    --load_8bit
--base_model    $BASE_MODEL    --lora_weights
'tloen/alpaca-lora-7b'
```

```
      restart: unless-stopped
      volumes:
      -  alpaca-lora:/root/.cache  #  Location
downloaded weights will be stored
      ports:
      - 7860:7860
      deploy:
      resources:
       reservations:
      devices:
            - driver: nvidia
            count: all
            capabilities: [ gpu ]
volumes:
  alpaca-lora:
      name: alpaca-lora
```

Bloque 1.2.3: Ejemplo 2 de imagen docker y dockerfile para el proyecto.

```
docker build -t your_image_name .

docker run -d --name your_container_name -p
<host_port>:<container_port> your_image_name

docker network create your_network_name

docker run -d --name your_container_name
--network your_network_name -p
<host_port>:<container_port> your_image_name
```

Bloque 1.2.4: Comandos de docker para la creación de la red interna

1.3 Características.

Realizando esta implementación obtenemos primordialmente las siguientes caracteristicas:

- Se puede usar la interfaz de **Hugging Face**s junto con las ventajas de **LangChain** y **ChromaDB.**
- Se puede desplegar sobre Kubernetes en plataformas de **computación** en la nube tales como **Google Cloud usando Cloud Run** y el **artifact Registry** o **Amazon Web Services** con múltiples instancias de **Elastic Compute ECS.**
- Una implementación de estas puede integrarse con repositorios locales sobre los cuales trabajar en mejoras del código e implementación continua.
- Se da la ventaja del escalamiento vertical y horizontal.

1.4 Desventajas

Entre las principales desventajas tenemos la dificultad en la misma. Ya que estas arquitecturas de paralelización sobre red son complejas de implementar y mantener.

En caso de facilitar el acceso al modelo, se debe crear una interfaz web de tipo Rest que conecte al modelo junto con otra interfaz sobre la cual se puedan enviar las consultas.

2. Construcción mediante API:

2.1 Fundamentos: Existe una implementación del modelo **Alpaca LoRA** que puede hacerse con los binarios del modelo, esta se comporta de igual manera a la implementación de **HuggingFaces** sin embargo al ejecutarse expone una interfaz de red al estar compilada, esta interfaz posee una web que viene incluida con el modelo sobre el cual se pueden hacer las consultas.

UNIVERSIDAD NACIONAL DE COLOMBIA

2.2 Construcción

Este método posee la ventaja de exponer un API de forma automática, sin embargo requiere correr sobre máquinas potentes a nivel de procesamiento, ya que no se puede hacer uso de otras máquinas virtuales sin tener que reescribir parte del despliegue del modelo.

2.3 Características.

Realizando esta implementación obtenemos primordialmente las siguientes caracteristicas:

- Despliegue automático de una interfaz de red.
- Exposición de un API Rest.
- Interfaz de usuario.

2.4 Desventajas

Entre las principales desventajas tenemos la arquitectura de este despliegue. Ya que requerimos tener paralelismo interno potenciado por una GPU lo cual puede representar un costo alto.

Además esta interfaz es incompatible en principio con lo que es Chroma y LangChain, lo cual impide la automatización del proceso de consultas al modelo.

Conclusiones:

Respuestas del Modelo:

El modelo Alpaca LoRA responde bien a preguntas específicas, pero lucha con preguntas ambiguas o amplias.

Tiempos de respuesta varían según la complejidad de la pregunta.

Implementación con Docker y API:

Docker ofrece flexibilidad y control del entorno, mientras que la API simplifica la exposición del modelo pero puede ser costosa.

Ambos enfoques tienen ventajas y desventajas en términos de complejidad y compatibilidad.

Ventajas y Desafíos:

El modelo demuestra entendimiento contextual pero tiene limitaciones.

La complejidad en las arquitecturas de paralelización y la compatibilidad con herramientas adicionales son desafíos.

En síntesis, Alpaca LoRA es sólido con preguntas específicas, pero tiene problemas con las ambiguas. La elección entre Docker y API depende de necesidades específicas. La complejidad y la compatibilidad son desafíos importantes.

Código y Repositorios

- **Github:** https://github.com/bjportelac/UP-0001-MainCodeAndData

UNIVERSIDAD NACIONAL DE COLOMBIA

Referencias

- **1. Google Colaboratory. (n.d.).** Retrieved from https://colab.research.google.com/drive/1ZnlAqDkxVwnElAICrzFK2oucemrSfzh6#scrollTo=N2h9ARy23XM_

- **2. Google Colaboratory. (n.d.).** Retrieved from https://colab.research.google.com/drive/161iuKadZkDEB-623rGOXwQZ2Hqv4gN54?usp=sharing#scrollTo=-xSH6H3DbfU3

- **3. Google Colaboratory. (n.d.).** Retrieved from https://colab.research.google.com/drive/1BEZ_qgtVqSmOmCTuhHs7lHiYB5M5_myg?usp=sharing

- **4. Google Colaboratory. (n.d.).** Retrieved from https://colab.research.google.com/drive/1NoHaZbKTgIKmdAkAGwPPHFLdxb3GScDc?usp=sharing#scrollTo=8oz8qJSjJNbV

- **5. Google Colaboratory. (n.d.).** Retrieved from https://colab.research.google.com/drive/115ba3EFCT0PvyXzFNv9E18QnKiyyjsm5?usp=sharing#scrollTo=-YZzdNkvm8E2

- **6. Google Colaboratory. (n.d.).** Retrieved from https://colab.research.google.com/drive/1NoHaZbKTgIKmdAkAGwPPHFLdxb3GScDc?usp=sharing#scrollTo=mtmwoYgZkDlc

- **7.Langchain Python API Documentation. (n.d.). Retrieved from** https://python.langchain.com/en/latest/

- **8.Langchain Python API Documentation. (n.d.).** Retrieved from https://python.langchain.com/en/latest/modules/indexes/vectorstores/getting_started.html

- **9. Langchain Python API Documentation. (**n.d.). Retrieved from https://python.langchain.com/en/latest/modules/indexes/text_splitters/examples/recursive_text_splitter.html

- **10. Chroma Usage Guide. (n.d.).** Retrieved from https://docs.trychroma.com/usage-guide

- **11. Stack Overflow. (2021, April 28).** Mismatched tensor size error when generating text with beam search huggingface [Online forum post]. Retrieved from https://stackoverflow.com/questions/67221901/mismatched-tensor-size-error-when-generating-text-with-beam-search-huggingface

- **12.Langchain Python API Documentation. (n.d.).** Retrieved from https://python.langchain.com/en/latest/modules/indexes/retrievers/examples/chroma_self_query.html

- **13.Artificialis. (n.d.). Crafting an engaging chatbot: Harnessing the power of Alpaca and Langchain** [Blog post]. Retrieved from https://medium.com/artificialis/crafting-an-engaging-chatbot-harnessing-the-power-of-alpaca-and-langchain-66a51cc9d6de

- **14.tloen/alpaca-lora [Computer software]. (n.d.). GitHub.** Retrieved from https://github.com/tloen/alpaca-lora

- **15.ML Expert. (n.d.). Alpaca Fine Tuning [Blog post].** Retrieved from https://www.mlexpert.io/machine-learning/tutorials/alpaca-fine-tuning

- **16. Langchain Blog. (n.d.). Langchain Chroma [Blog post].** Retrieved from https://blog.langchain.dev/langchain-chroma/

- **17. Towards Data Science. (n.d.). 4 ways of question answering in Langchain [Blog post].** Retrieved from https://towardsdatascience.com/4-ways-of-question-answering-in-langchain-188c6707cc5a

- **18. Hugging Face – On a mission to solve NLP, one commit at a time.. (n.d.).** Document Question Answering [Web page]. Retrieved from https://huggingface.co/tasks/document-question-answering

- 19.Wandb.ai pshar053/Alpaca-Lora-7B-FineTuning/reports/Fine-Tuning-Alpaca--VmlIdzo0MjE3NjYz [Web page]. (n.d.). Wandb.ai pshar053/Alpaca-Lora-7B-FineTuning/reports/Fine-Tuning-Alpaca--VmlIdzo0MjE3NjYz [Web page]. Wandb.ai pshar053/Alpaca-Lora-7B-FineTuning/reports/Fine-Tuning-Alpaca--VmlIdzo0MjE3NjYz. Retrieved from https://wandb.ai/pshar053/Alpaca-Lora-7B-FineTuning/reports/Fine-Tuning-Alpaca--VmlIdzo0MjE3NjYz

- **20. Artificial Corner. (n.d.). GPT4All is the local ChatGPT for your documents and it is free [Blog post].** Retrieved from https://artificialcorner.com/gpt4all-is-the-local-chatgpt-for-your-documents-and-it-is-free-df1016bc335

- **21. Vaclav Kosar. (n.d.). Tokenization in Machine Learning Explained [Blog post].** Retrieved from https://vaclavkosar.com/ml/Tokenization-in-Machine-Learning-Explained

- **22. Vitalflux.com. (n.d.). LLM Chain OpenAI ChatGPT Python Example [Web page].** Retrieved from https://vitalflux.com/llm-chain-openai-chatgpt-python-example/

- **23. linonetwo/langchain-alpaca [Computer software]. (n.d.). GitHub.** Retrieved from https://github.com/linonetwo/langchain-alpaca

- **24. hwchase17/chroma-langchain [Computer software]. (n.d.).** GitHub. Retrieved from https://github.com/hwchase17/chroma-langchain/blob/master/persistent-qa.ipynb

- **25. Técnica de PNL:** El modelado de temas es la clave para obtener .... https://bing.com/search?q=tecnica+modelado+datos+texto Con acceso 19/6/2023.

- **26. ¿Qué es el modelado de datos?** | IBM. https://www.ibm.com/mx-es/topics/data-modeling Con acceso 19/6/2023.

- **27. ¿Qué es el modelado de datos? | Definición, importancia y tipos - SAP**. https://www.sap.com/latinamerica/products/technology-platform/datasphere/what-is-data-modeling.html

- **28.** 2015 ACM/IEEE 18th International Conference on Model Driven Engineering Languages and Systems, MODELS 2015 - Proceedings (scimagojr.com)

UNIVERSIDAD
NACIONAL
DE COLOMBIA

HTTPS://WWW.SCIMAGOJR.COM/JOURNALSEARCH.PHP?
Q=21100453509&TIP=SID&CLEAN=0

- 29. AISB 2011: LEARNING LANGUAGE MODELS FROM MULTILINGUAL CORPORA (SCIMAGOJR.COM)

HTTPS://WWW.SCIMAGOJR.COM/JOURNALSEARCH.PHP?
Q=21100205946&TIP=SID&CLEAN=0