



UNIVERSIDAD
NACIONAL
DE COLOMBIA

PROYECTO FINAL: Question Answering over a Document using a Large Language Model (*LLM*)

Asignatura:
Redes de Computadores

Profesor:
Jose Octavio Salcedo

Integrantes:
[Ivan Yesid Sepulveda Paez](#)

Fecha: noviembre 9 de 2023 - Semestre 2023 - II

COMPRENSIÓN EMPRESARIAL

Introducción

Se propone el diseño de un LLM capaz de interpretar texto y generar respuestas basadas en el contexto del mismo texto que se le ha ingresado. El objetivo es crear un procesador de texto más amigable con el usuario para el portal de admisiones de la Universidad Nacional. Para esto, se implementará la metodología CRISP-DM.

Objetivos

Este proyecto tiene como objetivo crear un LLM capaz de interpretar texto normativo y generar respuestas basadas en el contexto del mismo texto que se le ha ingresado. El LLM se utilizará para mejorar la toma de decisiones, identificar patrones o descubrir conocimiento oculto en los datos, a través del análisis de los documentos de la normativa.

El LLM se diseñará para distintos grupos de interés, como estudiantes, profesores, personal administrativo, aspirantes, contratistas, etc. Las funcionalidades del LLM permitirán identificar información relevante para el usuario sin la necesidad de buscar en los documentos o leer todas las normativas.

El LLM realizará búsquedas basadas en palabras clave utilizadas por los distintos usuarios. Estas búsquedas se basarán en el contexto del documento, lo que permitirá resolver preguntas específicas y dar resultados relevantes.

El LLM también extraerá información enriquecida como entidades mencionadas, relaciones entre conceptos, categorización de temas, citas y referencia a los documentos donde se encuentran las normativas.

El LLM permitirá la interacción del tipo texto a texto, es decir que a partir de un texto de entrada se podrá generar un texto de salida con la respectiva respuesta a la consulta.

El LLM se implementará con una interfaz intuitiva para que cualquier usuario pueda interactuar con el aplicativo. La interfaz incluirá una 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 de universidad
 - Profesores y personal Académico
 - Personas que estén interesadas de manera particular en un tema
2. Mejora de la accesibilidad: El LLM permitirá a los usuarios con discapacidades acceder a la información .
3. Mejora de la eficiencia:
 - El LLM permitirá a los usuarios identificar información relevante más rápidamente, lo que ahorrará tiempo y esfuerzo.
 - Mejora de la experiencia del usuario: El LLM permitirá a los usuarios interactuar con la información normativa de forma más natural e intuitiva.
4. Metodología

El proyecto se implementará utilizando la metodología CRISP-DM. Esta metodología consta de las siguientes fases:

Comprensión del negocio: En esta fase, se comprenderán los requisitos del negocio y se definirán los objetivos del proyecto.

Entendimiento de los datos: En esta fase, se recopilarn y analizarán los datos que se utilizarán para entrenar el LLM.

Preparación de los datos: En esta fase, se prepararán los datos para el entrenamiento del LLM.

Entrenamiento del modelo: En esta fase, se entrenará el LLM utilizando los datos preparados.

Evaluación del modelo: En esta fase, se evaluará el rendimiento del LLM.

Despliegue del modelo: En esta fase, se desplegará el LLM en producción.

- Este proyecto tiene como objetivo crear un LLM capaz de interpretar texto y generar respuestas basadas en el contexto del mismo texto que se le ha ingresado. El LLM se utilizará para facilitar la interacción con documentos de texto, brindando acceso rápido y eficiente a la información relevante contenida en ellos.

5. Entendimiento del dominio y los documentos

- El dominio del proyecto es el texto en general. Los documentos que se manejarán serán de diversos tipos, como artículos, libros, informes, etc. La aplicación tendrá 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 leerlos todos.

6. Funcionalidades deseadas

- Las funcionalidades deseadas para el LLM son las siguientes:
- Búsquedas basadas en palabras clave: El LLM debe permitir realizar búsquedas basadas en palabras clave utilizadas por los distintos usuarios. Estas búsquedas se basarán en el contexto del documento, lo que permitirá resolver preguntas específicas y dar resultados relevantes.
- Extracción de información enriquecida: El LLM debe permitir extraer información enriquecida de los documentos, como entidades mencionadas, relaciones entre conceptos, categorización de temas, citas y referencia a los documentos donde se encuentran.

7. Interacción con el LLM

- La interacción que se pretende utilizar con el LLM es del tipo texto a texto. Esto significa que a partir de un texto de entrada se podrá generar un texto de salida con la respectiva respuesta a la consulta.

8. Requisitos de interfaz de usuario

- La interfaz de usuario del LLM debe cumplir con los siguientes requisitos:

- Interfaz para la entrada y salida de texto: La interfaz debe permitir ingresar y visualizar texto.
 - Intuición: La interfaz debe ser intuitiva para que cualquier usuario pueda interactuar con ella.
 - Orden: Los elementos de la interfaz deben estar dispuestos de manera ordenada para facilitar su uso.
 - Beneficios del proyecto
-
- El proyecto de creación de un LLM para la interpretación de texto tendrá los siguientes beneficios:
 - Mejora de la accesibilidad: El LLM permitirá a los usuarios con discapacidades acceder a la información contenida en los documentos.
9. Mejora de la eficiencia: El LLM permitirá a los usuarios identificar información relevante más rápidamente, lo que ahorrará tiempo y esfuerzo.
- Mejora de la experiencia del usuario: El LLM permitirá a los usuarios interactuar con la información de forma más natural e intuitiva.
10. Metodología
- El proyecto se implementará utilizando la metodología CRISP-DM. Esta metodología consta de las siguientes fases:
 - Comprensión del negocio: En esta fase, se comprenderán los requisitos del negocio y se definirán los objetivos del proyecto.
 - Entendimiento de los datos: En esta fase, se recopilarán y analizarán los datos que se utilizarán para entrenar el LLM.
 - Preparación de los datos: En esta fase, se prepararán los datos para el entrenamiento del LLM.
 - Entrenamiento del modelo: En esta fase, se entrenará el LLM utilizando los datos preparados.
 - Evaluación del modelo: En esta fase, se evaluará el rendimiento del LLM.
 - Despliegue del modelo: En esta fase, se desplegará el LLM en producción.

11. 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 desplegará 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

12. 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 Scrapping**;

- 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.

Con base en los enlaces identificados en la página de admisiones de la universidad, se extraen un total de 7 enlaces en los cuales se recopilan todas las normativas que deben tenerse en cuenta al momento de iniciar el proceso de admisión.

1. Exploración de los datos.

1.1 Extracción de datos: Se realiza el proceso de “**scraping**” en el cual se obtienen 7 conjuntos de datos representados como archivos HTML, que contienen los contenidos de los diferentes enlaces de las normativas, estos archivos son traídos desde la web como archivos mediante el siguiente código:

- **Urls disponibles en:** [Enlaces en url](#)

```
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 las normativas; Particularmente se busca la siguiente información:

1. Contenido de títulos, ya sea denotado por etiqueta tipo **header** **<H1>...<H6>**, o clases particulares del CSS.
2. Contenidos 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:

- **HTML obtenidos:** [Carpeta de HTML](#)

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

<P align=center>RESOLUCIÓN 235 DE 2009</P>

<P align=center>"Por la cual se reglamenta la admisión de exalumnos de la
Universidad Nacional de Colombia"</P>

<P align=center>LA VICERRECTORA ACADÉMICA 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:

RESOLUCIÓN 19 DE 2022

28 de julio

Por la cual se reglamenta la admisión a los programas curriculares de pregrado de la Universidad Nacional de Colombia

LA VICERRECTORÍA ACADÉMICA DE LA UNIVERSIDAD NACIONAL DE COLOMBIA

❓ **Bloque 1.3.1:** Ejemplo del texto obtenido

Hacemos la limpieza de datos eliminando 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 = processor(fileName=file_name,divider=divider)
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: [['Capítulo 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, 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:

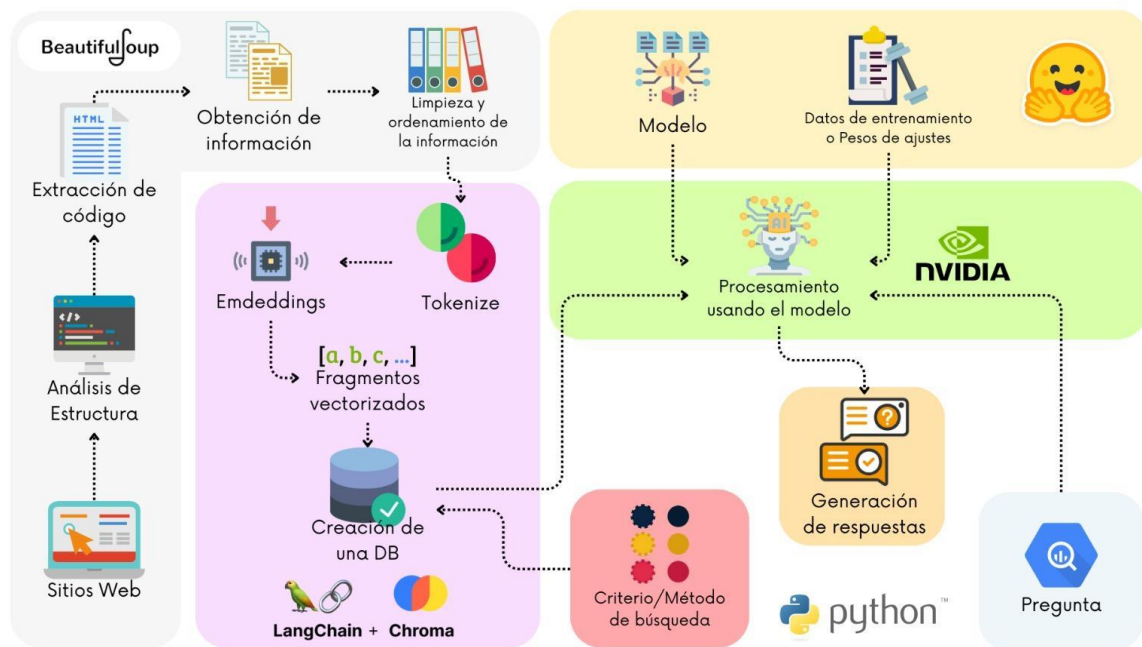


Figura 1: Flujo de construcción del modelo

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
```

Figura 2: Clonación del repositorio de Alpaca

Alpaca LoRA 7B es un adaptador de baja clasificación 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 entrenado 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 clasificación 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

```
[ ] 1 # Definimos el modelo base, en este caso LLaMA 7B (AlpacaLoRA-7B-Hf)
    2 # y el archivo de pesos de guanaco.
    3
    4 base_model_path = 'decapoda-research/llama-7b-hf'
    5 weights_path = "plncmm/guanaco-lora-7b"
```

Figura 5: Trabajo con la interfaz de Hugging Faces

```
1 from peft import PeftModel
2
3 base_model = LlamaForCausalLM.from_pretrained(
4     base_model_path,
5     load_in_8bit=True,
6     device_map='auto',
7 )
8
9 model = PeftModel.from_pretrained(
10     base_model,
11     weights_path,
12 )
```

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.

```
1 from langchain.llms import HuggingFacePipeline
2
3 pipe = pipeline(
4     "text-generation",
5     model=base_model,
6     tokenizer = tokenizer,
7     max_length = 256,
8     temperature = 0.1,
9     top_p = 0.75,
10    top_k = 40,
11    repetition_penalty=1.1,
12    max_new_tokens = 256,
13 )
14
15 local_llm = HuggingFacePipeline(pipeline=pipe)
```

Figura 7: Ajuste de parámetros del modelo.

1.2 Generación del diseño de prueba

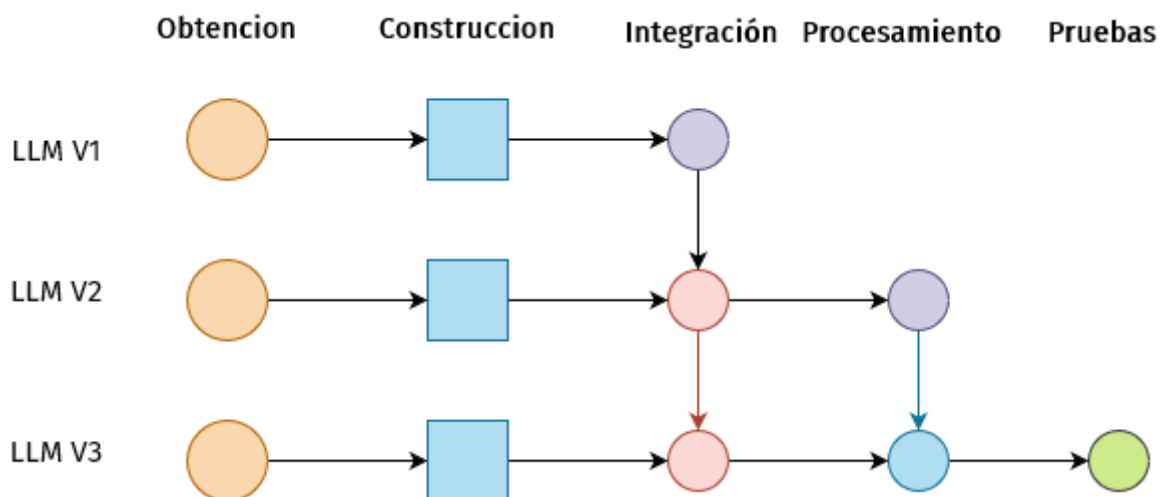


Figura 8: Generación iterativa del diseño.

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

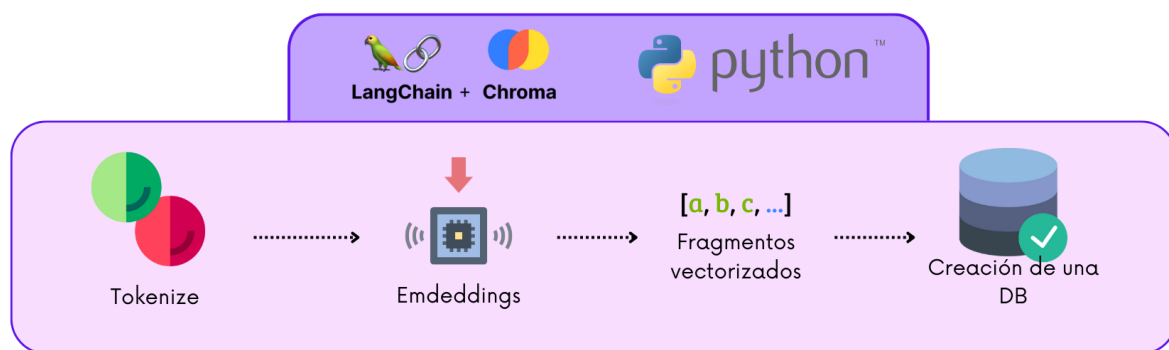
- Generado a partir de los binarios disponibles en los repositorios de Alpaca LoRA 7B, Guanaco en Hugging Faces.
- Pre-Entrenado con Stanford Alpaca.
- Calibrado con los pesos de Guanaco que es un conjunto de pesos y un dataset basado en Stanford Alpaca traducido al español.
- Desplegado en máquina.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

1.3.2 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 similitud 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 brindar el contexto obtenido de los documentos para que el modelo lo procese.
- Creamos un pipeline con el cual podemos ajustar y configurar parámetros del modelo, entre los más importantes tenemos:
 - Comportamiento del modelo.
 - Modelo a utilizar y tokenizador de procesamiento textual.
 - Penalidad por repetición en la generación de respuestas.
 - Máximo de tokens que pueden Generarse para las respuestas.
 - Longitud máxima de la entrada que procesará 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

```
from langchain.chains.question_answering import load_qa_chain

chain = load_qa_chain(local_llm, chain_type="stuff")

from langchain.embeddings import HuggingFaceEmbeddings

embeddings = HuggingFaceEmbeddings()

Downloading (...)a8e1d/.gitattributes: 100% 1.18k/1.18k [00:00<00:00, 66.8kB/s]
Downloading (...)_Pooling/config.json: 100% 190/190 [00:00<00:00, 9.00kB/s]
```

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

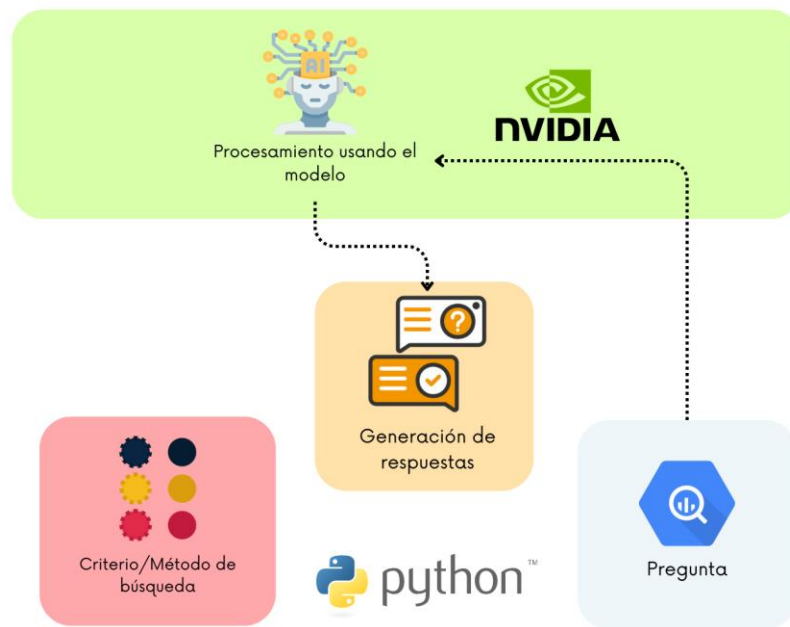
Como último paso en lo relacionado a la comunicación del modelo, llamamos un generador de Embeddings ajustado al modelo que estamos utilizando.

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 “**Similitud**” a la base de vectores Chroma.

```
docs= vectorstore.search(query,"similarity")
print(docs)
[Document(page_content='Artículo 1 A Partir Del Presente Acuerdo Se Contempla Un Cupo Equivalente Al 2 De Los Cupos Establecidos Para Cada Carrera En Su Sede Bogotá Y Seccionales Pa
```

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.

```
chain.run(input_documents=docs, question=query)
Both 'max_new_tokens' (=256) and 'max_length' (=256) seem to have been set. 'max_new_tokens' will take precedence. Please refer to the documentation for more information.
' 3'
```

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:

Evaluación

1.3 Características.

Realizando esta implementación obtenemos primordialmente las siguientes características:

- Se puede usar la interfaz de **Hugging Faces** 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

22

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.

1. 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 **Hugging Faces** 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.

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 características:

- 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.

Referencias

- **1. Google Colaboratory. (n.d.).** Retrieved from https://colab.research.google.com/drive/1ZnIAqDkxVwnEIAICrzFK2oucemrSfzh6#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_qgtVqSmOmCTuhHs7IHiYB5M5_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/recur_sive_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--Vmldzo0MjE3NjYz [Web page]. (n.d.).** Wandb.ai pshar053/Alpaca-Lora-7B-FineTuning/reports/Fine-Tuning-Alpaca--Vmldzo0MjE3NjYz [Web page]. Wandb.ai pshar053/Alpaca-Lora-7B-FineTuning/reports/Fine-Tuning-Alpaca--Vmldzo0MjE3NjYz. Retrieved from <https://wandb.ai/pshar053/Alpaca-Lora-7B-FineTuning/reports/Fine-Tuning-Alpaca--Vmldzo0MjE3NjYz>
- **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>