

# Introducción a los Sistemas inteligentes

Question Answering over a Document using a Large Language Model (LLM)

## Descripción del problema:



Muchos documentos tales como reglamentos y normativas pueden ser complejos de entender para el usuario común. Por lo que procesos que impliquen el uso de esta información pueden ser complejos y generar demasiadas dudas.

### **Solución Propuesta:**

Implementar un modelo de inteligencia artificial el cual pueda.

- Aprender sobre documentos.
- Responder preguntas respecto a los contextos documentales.

cscript type="text/javascript" src="/rlunal/re

LangChain + Chroma

Tener una interfaz de manejo sencillo o intuitivo.







Identificar los sitios web que poseen la información

Examinar el código base de cada uno de los sitios

Extraer el código como archivos **HTML** en bruto.

Extraer la información mediante código

Limpiar la información y ordenarla en uno o mas documentos

script type "text/javascript" src="/rluna <script type="text/javascript"> de pregrado y posgrado que ofrece

<script type="text/javascript" src="/rluna.</pre>

python

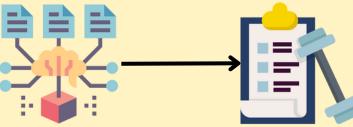
### Obtención del Modelo







Utilizamos AlpacaLoRA-7B ya que es uno de los pocos modelos con compatibilidad al español (traducción)



Descargamos el modelo pre-entrenado con el conjunto de datos

StandardAlpaca de la **U.Stanford** 

Descargamos los pesos de **Guanco**, ya que estan calibrados para procesamiento español y construimos el modelo.

- from transformers import LlamaTokenizer, LlamaForCa
- tokenizer = LlamaTokenizer.from\_pretrained(base\_mode

Downloading tokenizer.model: 100%

Downloading (...)cial\_tokens\_map.json: 100%

Downloading (...)okenizer\_config.json: 100%

The tokenizer class you load from this checkpoint is not The tokenizer class you load from this checkpoint is 'LL The class this function is called from is 'LlamaTokenizer

#### **Transformando Datos** 3



Beautiful oup Python

transformándolo en fragmentos pequeños con LangChain

Trabajamos el texto,

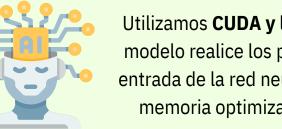
Asignamos un valor numérico a cada fragmento (tokenizar) Transformamos el valor token en un vector ndim real para la red neuronal del modelo

Creamos una DB Chroma con los fragmentos tokenizados en ella

### rocesamiento







Utilizamos CUDA y las GPU de colab para que el modelo realice los procesamientos al recibir una entrada de la red neuronal. Y tambien para liberar memoria optimizando el uso de los recursos.

### chunk\_size=100, chunk\_overlap=10, add start index = True documents = text\_splitter.split\_documents(documents) print(f'Cantidad de fragmentos generados: {len(documents)}\n')

text splitter = RecursiveCharacterTextSplitter

from langchain.text\_splitter import RecursiveCharacterTextSplitter

#### 6 **Metricas**



A continuación presentamos algunas de las métricas del modelo al procesar diferentes preguntas:

### Tiempo minimo de respuesta:

1.66 segundos

Tiempo maximo de respuesta:

35.69 segundos

Promedio de respuesta: 14.85 segundos

### **Uso y Consultas** LangChain + Chroma



Generamos una consulta para el modelo, hacemos uso de LangChain para enviar al modelo la consulta.



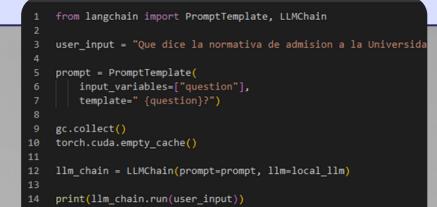
Con la **DB de Chroma**, consultamos todos los fragmentos que tengan cierta similitud a la pregunta que le pasamos al modelo. y lo convertimos en el contexto.



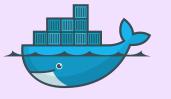
Consultamos al modelo entregando el contexto y la pregunta con la que también consultamos la DB Chroma. A partir de este punto el modelo sera capaz



de evaluar el contexto para responder.



### Despliegue



Podemos construir una imagen **de Docker** y desplegar los modelos en ambientes como GCCloud Run o AWS E3, creando subsistemas linux en los

cuales ejecutar el modelo.



Usando un**a Maquina** Virtual podemos exponer una interfaz web del modelo, esto facilitaría el uso de HTTP sobre el modelo.



**Notebook** 

Facultad de INGENIERÍA

Sede Bogotá

Departamento de Ingeniería de Sistemas y Computación