Trabajo fin de Ciclo Complementario

Licenciatura en Ciencia de Datos

Benchmark de LLMs desde la perspectiva de indicadores lingüísticos aplicados a las respuestas

Director de la Carrera: Germán Giro
Profesor: Alejandro Hernández
Alumno: Norberto Emanuel Vicente Giannattasio

Contenido

Introducción

No nos sumergimos mucho en los basamentos teórico técnicos de los Modelos de Lenguaje Grande (LLM), ya que no es el objetivo de este trabajo, pero haremos una breve introducción describiendo qué son los LLM y una breve historia acerca de los mismos.

Que es un Modelo de Lenguaje Grande (Large Language Model)

Los LLMs o Modelos de Lenguaje Grande (LLM desde este momento por sus siglas en inglés) son algoritmos avanzados de aprendizaje profundo capaces de realizar una amplia gama de tareas relacionadas con el procesamiento del lenguaje natural (NLP).

La diferencia que todos hemos notado desde finales de 2022 o principios de 2023 estriba en el tamaño y cantidad de datos de entrenamiento. Los nuevos modelos, cimentados en la arquitectura Transformers —actualmente la más popular—, se entrenan con vastos conjuntos de datos, lo que les confiere una impresionante habilidad para reconocer, resumir, traducir, predecir y generar texto. Si además añadimos una funcionalidad de chatbot para interactuar, como lo hizo OpenAI con ChatGPT, Meta con Llama2 o Google con Gemini, entonces tenemos una experiencia nueva, una experiencia cognitiva que los humanos no habíamos tenido con ninguna máquina. Esa es la razón por la que nos divertimos y "enganchamos" tanto a los modelos como ChatGPT: para nuestro cerebro, estamos teniendo **una experiencia cognitiva**, una conversación, como la podríamos tener con un bibliotecario de amplísimos conocimientos o cualquier otra persona.
Es fundamental distinguir entre los LLMs y la AI generativa. Mientras que los LLMs se centran en el texto, la AI generativa abarca un espectro más amplio, multimodal, incluyendo la creación de imágenes, música y más. Todos los LLMs pueden considerarse parte de la AI generativa, pero no toda AI generativa es un LLM.
A modo de ejemplo, Claude2 de Anthropic, Gemini de Google, y los famosos ChatGPT o Llama2 (o 3) son LLMs, mientras que Stable Diffusion o Bing Image Creator de Microsoft, basado en Dall-e 3, son AI Generativa pero producen imágenes, no son grandes modelos de lenguaje.

Algunos ejemplos de modelos de lenguaje grandes populares incluyen:

- **ChatGPT:** un chatbot de inteligencia artificial generativa desarrollado por OpenAI.
- **PaLM:** Pathways Language Model (PaLM) de Google, un modelo de lenguaje de transformadores capaz de realizar razonamientos aritméticos y de sentido común, explicar bromas, generar código y traducir.
- **BERT:** el modelo de lenguaje representación de codificador bidireccional de transformadores (BERT) también se desarrolló en Google. Es un modelo basado en transformadores que puede comprender el lenguaje natural y responder preguntas.
- **XLNet:** un modelo de lenguaje de permutación, XLNet generó predicciones de salida en un orden aleatorio, lo que lo distingue de BERT. Evalúa el patrón de tokens codificados y luego predice los tokens en orden aleatorio, en lugar de en un orden secuencial.
- **GPT:** los transformadores generativos pre entrenados son quizá los modelos de lenguaje grandes más conocidos. Desarrollados por OpenAI, GPT es un modelo fundacional popular cuyas iteraciones numeradas son mejoras de sus predecesores (GPT-3, GPT-4, etc.).

Además, de estos modelos de lenguaje también es preciso mencionar los conocidos como LLM de código abierto como ser:

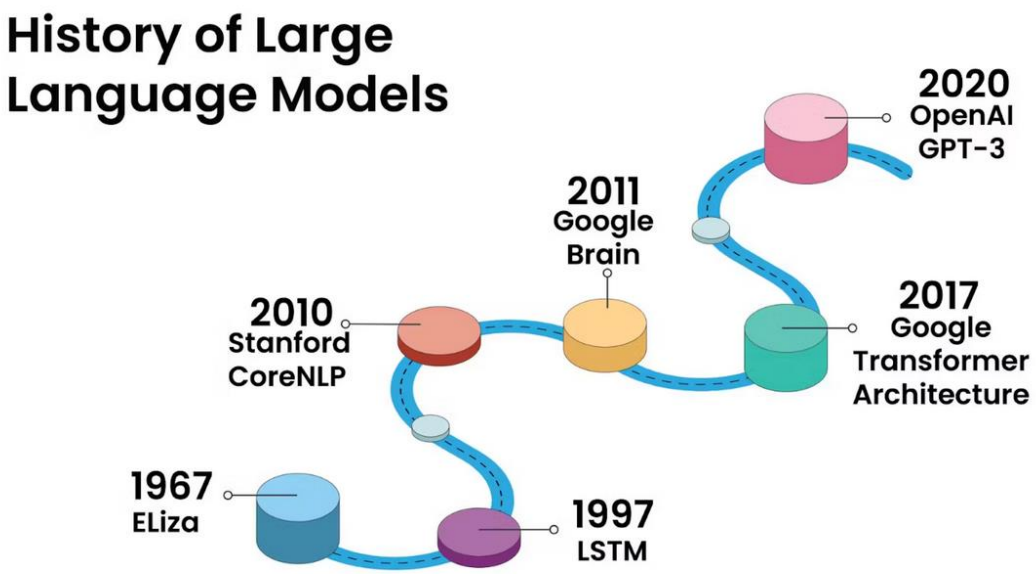
Llama2 (o 3) de Meta (este a su vez posee una versión específica para la generación o corrección de código de programación llamada CodeLlama), Gemma de Google (este LLM también posee una versión específica para la generación y corrección de código de programación llamada CodeGemma) o Mistral de la compañía francesa del mismo nombre.

Brevísima historia de los LLM

Los LLM existen desde hace algunas décadas, pero hace poco se volvieron lo suficientemente potentes y sofisticados como para que se los use en una amplia gama de tareas. El primer LLM se creó en la década de los sesenta con el primer chatbot, Eliza. Sin embargo, sus capacidades eran muy limitadas. No fue hasta la década de 2010 cuando los LLM maduraron hasta alcanzar un nivel de funcionalidad adecuado para modelos muy grandes y aplicaciones del mundo real.

Un momento crucial en el avance de LLM llegó con la introducción de la arquitectura Transformer en 2017. El modelo de Transformer mejoró significativamente la comprensión de las relaciones de palabras dentro de

oraciones, lo que dio como resultado una generación de texto que sea gramaticalmente correcta y semánticamente coherente.
En los últimos años, se pre entrenó a los LLM con amplios conjuntos de datos de cientos de miles de millones de textos y códigos, lo que ha permitido mejorar sustancialmente su rendimiento en diversas tareas. Por ejemplo, algunos LLM ahora pueden generar texto indistinguible del texto escrito por humanos.



Consideraciones Iniciales

Objetivo

El objetivo del presente trabajo es el desarrollo de un framework de comparación independiente para LLMs utilizando diversas herramientas del ambiente Open Source

Consideraciones acerca de Frameworks de pruebas actuales.

La evaluación de los LLM es un ámbito incipiente y abierto a experimentación y creación, dentro de este campo existen algunos frameworks/librerías para evaluar las respuestas de lo LLM, entre las principales podemos mencionar

- Uptrain
- Ragas
- DeepEval

En las evaluaciones previas, se verificó que estos frameworks de test, necesitan una Key para utilizar la plataforma de OpenAI (Excepto Uptrain) para utilizar el api abierta de esta compañía (la creadora de ChatGPT) y además están desarrolladas para evaluar un LLM contra el de la compañía OpenAI y obtener las métricas en base a ellos. (Podríamos detallar el uso de estos, pero sería tema de un paper al completo).
Por esta razón es que se decidió desarrollar una serie de métricas para realizar la evaluación de un LLM frente a otro, también es de mencionar, que para no largar el desarrollo del presente documento, no ahondaremos en los detalles teóricos que le dan sustento a cada uno de los indicadores seleccionados para realizar la comparativa.

Selección de LLMs

Por esta razón es que se decidió desarrollar una serie de métricas para realizar la evaluación de un LLM frente a otro.
Se eligieron los LLM Open Source de dos de las compañías de tecnología más grandes e innovadoras Llama2 de Meta y Gema de Google, en ambos casos, para que la comparación/benchmark sea justo, se eligieron las versiones que fueron entrenadas con 7 mil millones de parámetros en cada caso; es decir la parametrización de entrenamiento de ambos modelos de LLM, a nivel numérico cuantitativo son idénticas (no tenemos certeza si son los mismos sets de parámetros a nivel cualitativo).
Se Omitió la versión más nueva del Modelo Llama, llama3, ya que este está entrenado con 8 mil millones de parámetros y la comparativa no sería matemáticamente correcta, ya que esta versión del modelo posee una ventaja de mil millones de parámetros sobre la versión de Google.

Herramientas Utilizadas

En esta sección, antes de presentar los detalles y resultados del framework propuesto y de las pruebas realizadas, procederemos a mencionar las herramientas utilizadas:

- **Python:** Se eligió Python como lenguaje de programación, ya que hoy es el estándar de facto en el mundo de datos por sobre otros como Scala o R. Además de ser un lenguaje de programación Open Source y con una enorme comunidad y documentación accesible.

- **Ollama:** es una herramienta que nos permite ejecutar modelos de lenguaje en nuestros ordenadores de manera sencilla. Su principal propósito es facilitar el acceso a modelos de gran tamaño sin necesidad de utilizar la nube. Con Ollama, podemos descargar y ejecutar varios modelos, incluyendo LLaMA-2, Uncensored LLaMA, CodeLLaMA, Falcon y Mistral, entre otros.

- **SQLite:** es un sistema de gestión de bases de datos relacional compatible con ACID, contenida en una relativamente pequeña biblioteca escrita en C. SQLite es un proyecto de dominio público creado por D. Richard Hipp.

- **TextStat:** Paquete de Python para calcular estadísticas a partir de texto y determinar la legibilidad, complejidad y nivel de grado de un corpus específico.

- **TextBlob:** TextBlob es una biblioteca de Python para procesar datos textuales. Proporciona una API simple para realizar tareas comunes de procesamiento de lenguaje natural (PLN) como el etiquetado de partes del discurso, la extracción de sintagmas nominales, el análisis de sentimiento, la clasificación y más. Particularmente se utilizará para el análisis de sentimientos.

- **Langdetect:** Librería que permite la detección del lenguaje de un texto.

- **NLTK (Natural Language Toolkit):** Es una plataforma líder para construir programas de Python que trabajan con datos de lenguaje humano. Proporciona interfaces fáciles de usar para más de 50 corpus y recursos léxicos como WordNet, junto con un conjunto de bibliotecas de procesamiento de texto para clasificación, tokenización, derivación, etiquetado, análisis sintáctico y razonamiento semántico. También incluye

envoltorios para bibliotecas de PNL de uso industrial.

- **Google Colab:** también conocido como Colaboratory, es un servicio gratuito de Jupyter Notebook alojado en la nube que te permite ejecutar código Python sin necesidad de configuración.

- **VsCode:** es un editor de código fuente gratuito y de código abierto desarrollado por Microsoft, altamente configurable, personalizable y extensible mediante plugin

- **Plotly Express**: es una biblioteca de Python para crear visualizaciones de datos interactivas de forma rápida y sencilla.

- **Ngrok:** es una herramienta que permite exponer un servidor local a Internet de forma rápida y sencilla. Funciona creando un túnel seguro entre tu computadora y los servidores de Ngrok, lo que te permite acceder a tu servidor desde cualquier lugar del mundo con una URL pública.

Propuesta de Framework de Benchmark para LLMs

Descripción del experimento

El experimento consta del envío de una cantidad finita de prompts a los LLMs, los cuales otorgaran una respuesta a los mismos. Los prompts enviados a LLMs serán los mismos en cada caso.
Las respuestas se almacenarán en una base de datos; de cada respuesta se obtendrán estadísticas léxicas para cada respuesta de cada LLM y se realizarán comparaciones entre las mismas, obteniendo indicadores particulares con los que se llegara a la conclusión de que LLM fue más performante de acuerdo a las mismas.

Prompts de experimentación

A continuación, se presentan los prompts de experimentación que se enviaron a cada LLM para evaluar sus respuestas.
Como se observa, para la presente "prueba de concepto" sólo se han desarrollado 10 prompts los cuales se enviaron a los LLM seleccionados para evaluar sus respuestas. Estos se dividieron en 3 tipos de complejidad; baja (2), media (5) y alta (3)

id_promtp	complejidad	prompt
1	Baja	Buenos días, ¿me podrías ayudar con algo hoy?
2	Baja	¿Cuál es la capital de Francia?
3	Media	Estoy planeando un viaje a Argentina. ¿Podrías recomendarme algunos lugares para visitar?
4	Media	Por favor, podrías escribir un artículo sobre la inteligencia artificial de no más de 300 palabras
5	Media	Traduce este texto ""Crónicas Marcianas" de Ray Bradbury es una colección de relatos que exploran la colonización humana en Marte. Con una prosa poética y visionaria, Bradbury teje historias emotivas sobre la soledad, la nostalgia y la naturaleza humana a través de encuentros imaginativos entre colonizadores y marcianos. Cada relato revela la fragilidad de la existencia y la búsqueda de significado en un mundo alienígena, mientras refleja paralelos con la condición humana en la Tierra. Esta obra maestra

		de la ciencia ficción cautiva con su estilo evocador y su capacidad para trascender lo tecnológico, adentrándose en lo más profundo del alma humana." del español al inglés.
6	Media	Escribe un poema sobre el libertad
7	Media	Escribe un guión para un cortometraje de comedia.
8	Alta	Escribe un código en Python que imprima "Hola, mundo!" en la consola.
9	Alta	Utiliza Pandas para analizar este conjunto de datos y generar un gráfico de barras.
10	Alta	Crea una función en PySpark que calcule la media de un campo en un conjunto de datos.

Tabla 1 - Prompts de test

El prompt número 5, se envió un breve texto para que se le realice una traducción del español al inglés. La respuesta en ese caso debe ser en inglés, en los demás casos, la respuesta debe ser en castellano para ser evaluada como correcta, en caso contrario, se marcará como errónea y ese registro no será tenido en cuenta para las estadísticas que se obtienen.

Respuestas a los Prompts de experimentación por parte de los LLM

Debido al tamaño de las respuestas y a que crear tablas aquí ocuparían demasiado espacio procedemos a dejar los links a los archivos donde se encuentran las respuestas dadas por cada LLMs

- Respuestas Gemma
- Respuestas Llama2

Indicadores seleccionados

- **Tiempo de lectura (Reading Time):** Indica el tiempo aproximado de lectura del texto.

- **Conteo de oraciones (Sentence Count):** Es el número de oraciones que se encuentran en el texto.

- **Conteo de Caracteres (Character Count):** Es el número de caracteres que se encuentran en el texto (esto incluye signos de puntuación).

- **Conteo de letras (Letter Count):** Es el número de caracteres que se encuentran en el texto (esto excluye signos de puntuación).

- **Conteo de palabras (Lexicon count):** Es la cantidad de palabras presentes en el texto.

- **Largo promedio de oraciones (Average sentence length)** : Es el largo promedio de las oraciones medido en cantidad de palabras en la misma.

- **Promedio de letras por palabra (Average letters per Word):** Es el número promedio de letras por palabras en el texto.

- **Cantidad de Stopwors (Quantity Stopwords):** Es la cantidad de stopwords contenidas en el texto.

- **Cantidad de Palabras (Quantity Words):** Es la cantidad de palabras presentes en el texto.

- **Densidad Léxica (Lexical Density):** Es la densidad léxica, calculada como la cantidad de palabras sin repetición dividido por la cantidad total de palabras

- **Riqueza Léxica (Lexical Richness):** Es la densidad léxica expresada en porcentajes al multiplicar la mencionada por 100.

- **Análisis de Sentimiento (Analyze Sentiment):** Es el valor que se obtiene del análisis de sentimientos expresado en Neutro, Negativo, Positivo.

Análisis y Comparaciones de indicadores aplicados

Análisis indicadores numéricos

En la siguiente tabla observamos el resultado de la comparación de los indicadores numéricos calculados, para cada registro de prueba, y se obtiene en cada caso cuál de los LLMs dio mejor respuesta por sobre todo (campos llama_better y gemma_better); también se verifica si en algún caso la respuesta tuvo la misma performance (campo equals). En la columna final hacemos la sumarización que debe ser 10, ya que los prompts enviados para testar fueron 10. El último registro, el indicador "total_sum" es un totalizador de cada columna.

	indicador	llama_better	gemma_better	equals	total
0	tiempo de lectura	2	8	0	10
1	cantidad de oraciones	3	5	2	10
2	cantidad de caracteres	2	8	0	10
3	cantidad de letras	1	8	1	10
4	cantidad de silabas	1	8	1	10
5	largo promedio de oraciones	0	9	1	10
6	promedio de letras por palabra	5	4	1	10
7	cantidad de stopwords	2	7	1	10
8	cantidad de palabras	1	8	1	10
9	densidad lexica	7	2	1	10
10	riqueza lexica	7	2	1	10
11	total_sum	31	69	10	110

tabla 2- comparación y sumarización de indicadores.

A continuación, se agregó a la tabla anterior, los valores expresados de modo porcentual (columnas (llama_better_percent, gemma_better_percent y equal). En los porcentajes totales, el cálculo se realizó de la siguiente manera:

(total sumarizado para ese llm /el valor total) *100 = % registro total_sum

	indicador	llama_better	gemma_better	equals	total	llama_better_percent	gemma_better_percent	equal
0	tiempo de lectura	2	8	0	10	20.00	80.00	0.00
1	cantidad de oraciones	3	5	2	10	30.00	50.00	20.00
2	cantidad de caracteres	2	8	0	10	20.00	80.00	0.00
3	cantidad de letras	1	8	1	10	10.00	80.00	10.00
4	cantidad de silabas	1	8	1	10	10.00	80.00	10.00
5	largo promedio de oraciones	0	9	1	10	0.00	90.00	10.00
6	promedio de letras por palabra	5	4	1	10	50.00	40.00	10.00
7	cantidad de stopwords	2	7	1	10	20.00	70.00	10.00
8	cantidad de palabras	1	8	1	10	10.00	80.00	10.00
9	densidad lexica	7	2	1	10	70.00	20.00	10.00
10	riqueza lexica	7	2	1	10	70.00	20.00	10.00
11	total_sum	31	69	10	110	28.18	62.73	9.09

tabla 3- comparación y sumarización de indicadores.

A continuación, se logra observar de manera gráfica, la comparación entre los resultados obtenidos por cada una de las respuestas de ambos LLMs a los 10 prompts de test, que anteriormente se presentaron en forma numérica

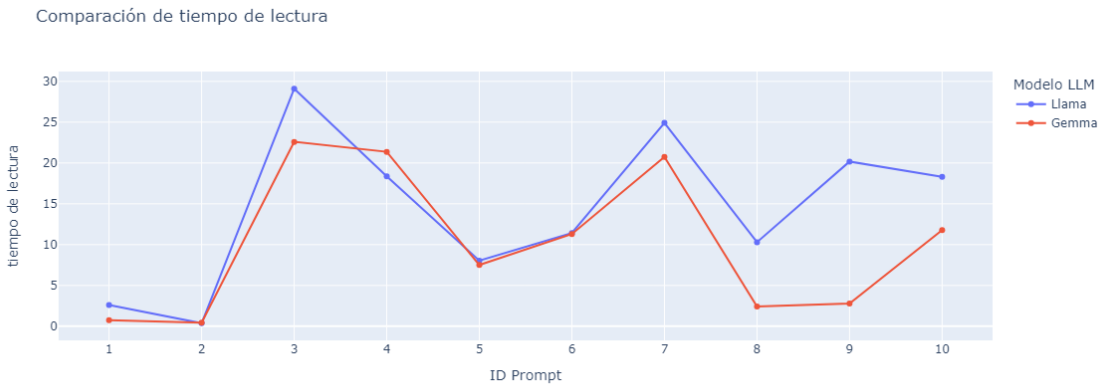


Fig. 1 - Comparación tiempo de lectura

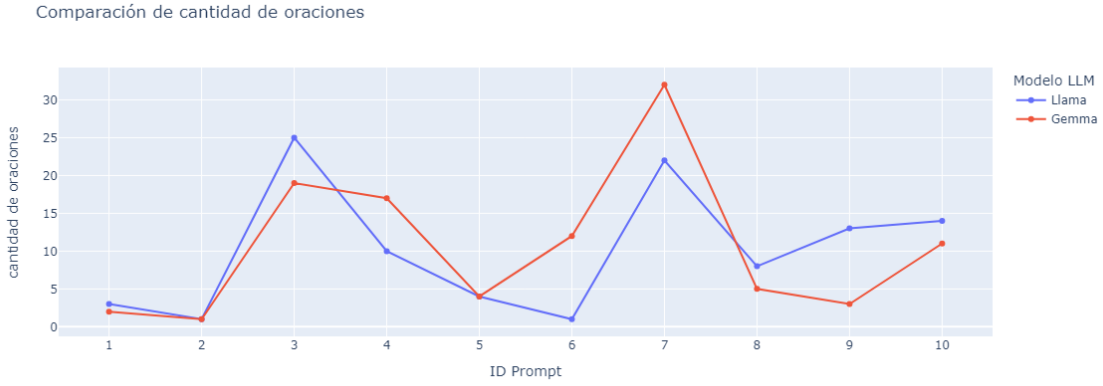


Fig. 2 - Comparación cantidad de oraciones

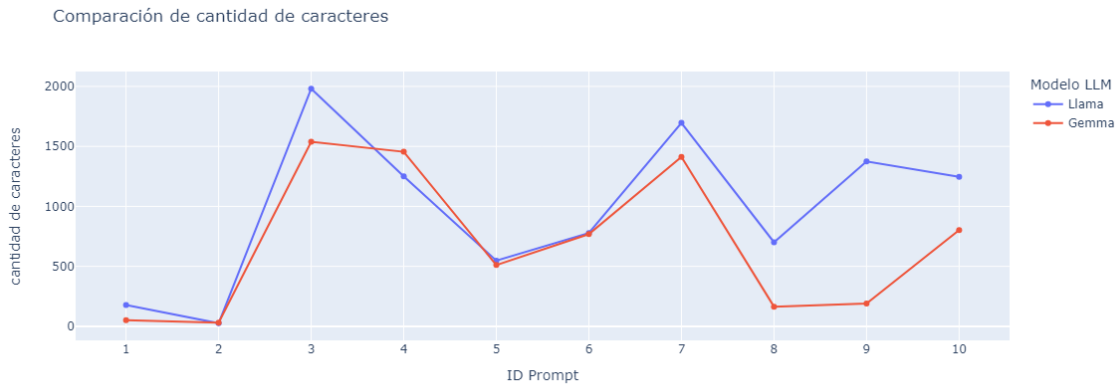


Fig. 3 - Comparación cantidad de caracteres

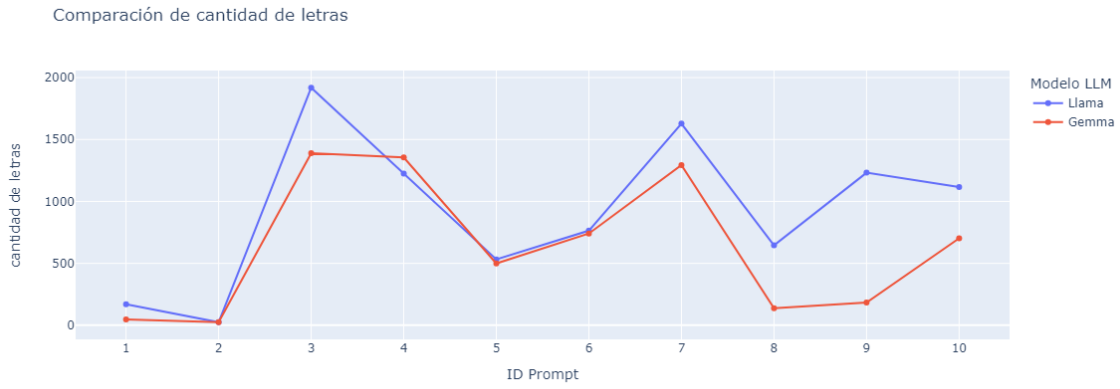


Fig. 4 - Comparación cantidad de letras

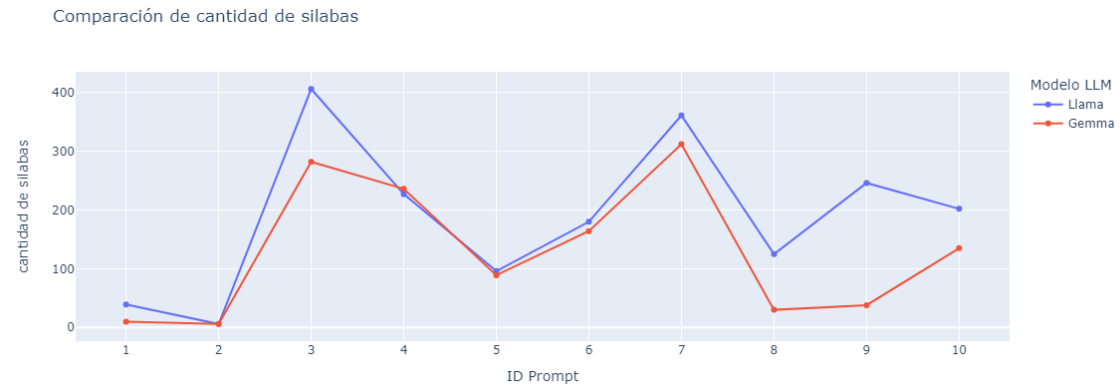


Fig. 5 - Comparación cantidad de sílabas

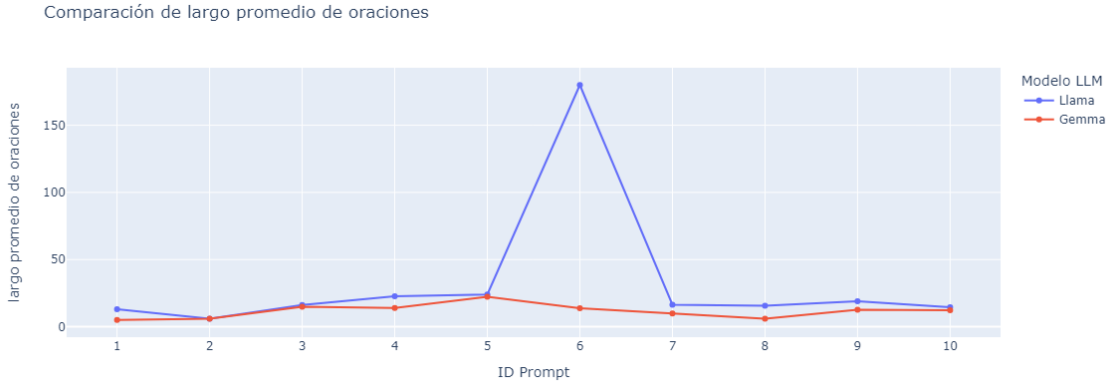


Fig. 6 - Comparación largo promedio de oraciones

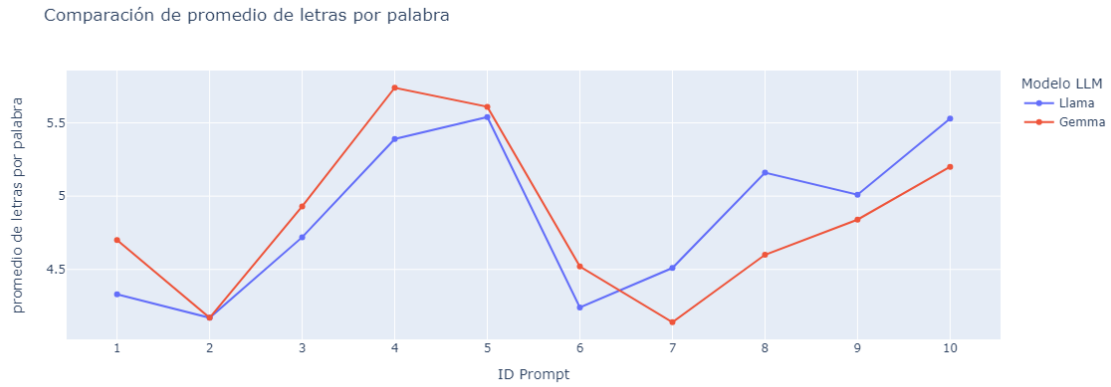


Fig. 7 - Comparación promedio de letras por palabra

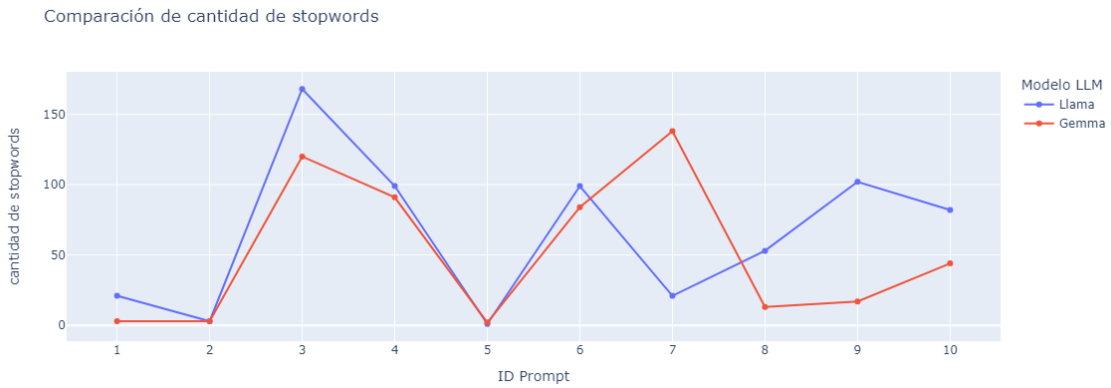


Fig. 8 - Comparación cantidad de stopwords

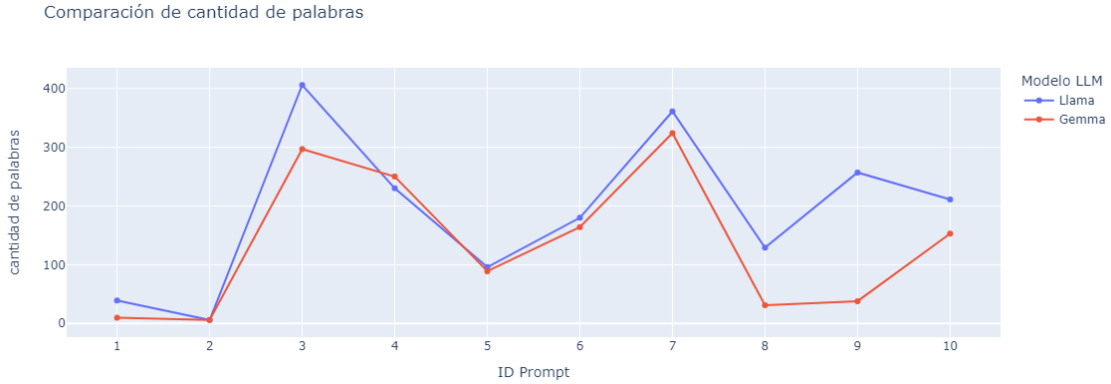


Fig. 9 - Comparación cantidad de palabras

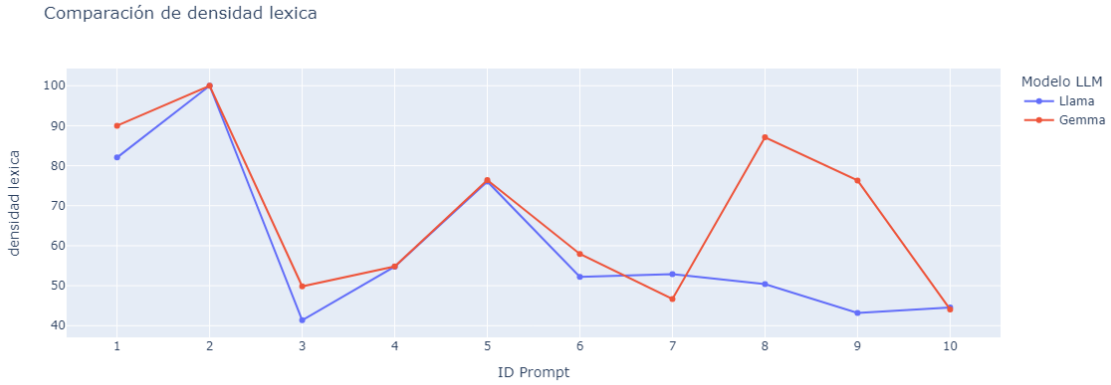


Fig. 10 - Comparación densidad léxica

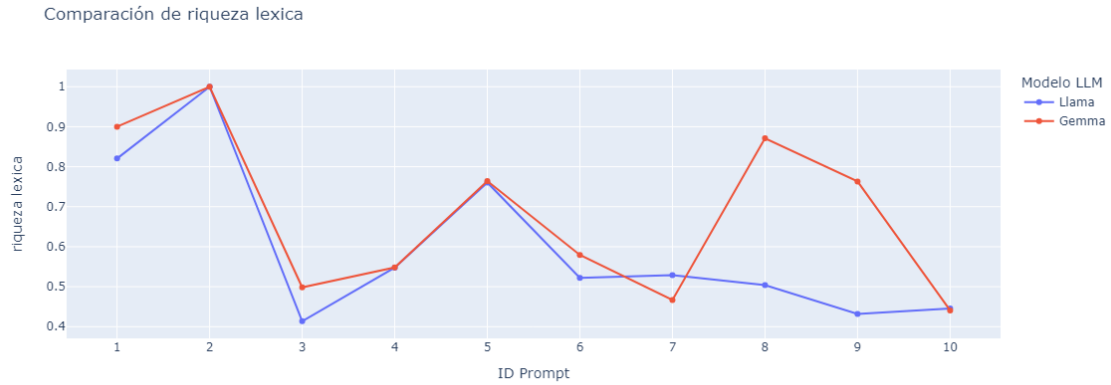


Fig. 11 - Comparación riqueza léxica

Análisis Sentimientos

En esta sección evaluamos, mediante el análisis de sentimientos, la "cordialidad" de cada respuesta de los LLMs

	modelo llm	analisis sentimiento	count
0	llama2	Neutral	5
1	llama2	Positive	3
2	llama2	Negative	2

tabla 4 - análisis de sentimiento de llama2

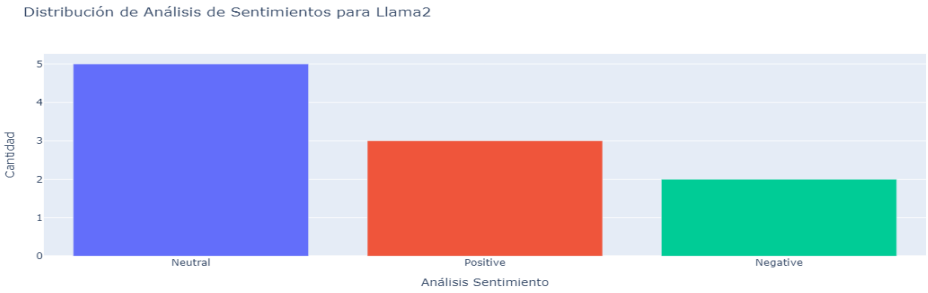


Fig. 12 - Distribución del análisis de sentimiento para llama2

	modelo llm	analisis sentimiento	count
0	gemma	Neutral	6
1	gemma	Negative	3
2	gemma	Positive	1

tabla 5 - análisis de sentimiento de gemma

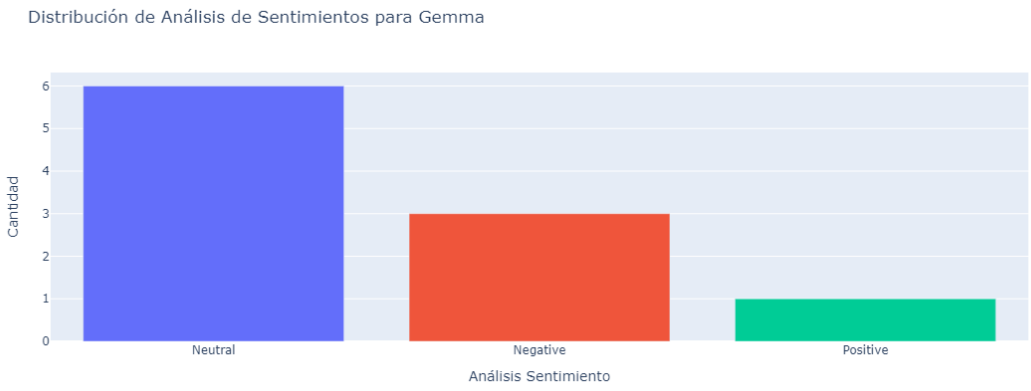


Fig. 13 - Distribución del análisis de sentimiento para gemma

Análisis de Comparación de Promedios

Finalmente, se realizó también una comparación de los valores promedios de los indicadores obtenidos cuantitativos lingüísticos; vale mencionar que valores menores en cada caso, son mejores, por ejemplo, en el caso de tiempo de lectura, que es el primer valor comparado, se visualiza que es mejor en el LLM Gemma por sobre Llama2, lo que en ese ítem lo hace mejor; teniendo en cuenta lo antes dicho, hemos obteniendo los siguientes resultados:

```
REPORTE de COMPARACION

Comparación del promedio de tiempo de lectura
Los valores promedios de los valores obtenidos para tiempo de lectura
son mejores en gemma:7b por sobre llama2.
gemma: 10.17 - llama2: 14.36

Comparación del promedio de cantidad de oraciones
Los valores promedios de los valores obtenidos para cantidad de
oraciones son mejores en llama2 por sobre gemma.
gemma: 10.6 - llama2: 10.1

Comparación del promedio de cantidad de caracteres
Los valores promedios de los valores obtenidos para cantidad de
caracteres son mejores en gemma:7b por sobre llama2.
gemma: 692.2 - llama2: 977.7

Comparación del promedio de cantidad de letras
Los valores promedios de los valores obtenidos para cantidad de letras
son mejores en gemma:7b por sobre llama2.
gemma: 637.4 - llama2: 925.4

Comparación del promedio de cantidad de silabas
El valor promedio de los valores obtenidos para cantidad de silabas son
mejores en gemma:7b por sobre llama2.
gemma: 130.2 - llama2: 188.8

Comparación del promedio de largo promedio de oraciones
Los valores promedios de los valores obtenidos para largo promedio de
oraciones son mejores en gemma:7b por sobre llama2.
gemma: 11.65 - llama2: 32.72

Comparación del promedio de promedio de letras por palabra
Los valores promedios de los valores obtenidos para promedio de letras
por palabra son mejores en gemma:7b por sobre llama2.
gemma: 4.85 - llama2: 4.86

Comparación del promedio de cantidad de stopwords
El valor promedio de los valores obtenidos para cantidad de stopwords
son mejores en gemma:7b por sobre llama2.
gemma: 51.5 - llama2: 64.9
```

```
Comparación del promedio de cantidad de palabras
Los valores promedios de los valores obtenidos para cantidad de
palabras son mejores en gemma:7b por sobre llama2.
gemma: 136.2 - llama2: 191.5

Comparación del promedio de densidad léxica
El valor promedio de los valores obtenidos para densidad léxica son
mejores en llama2 por sobre gemma.
gemma: 68.31 - llama2: 59.75

Comparación del promedio de riqueza léxica
El valor promedio de los valores obtenidos para riqueza léxica son
mejores en llama2 por sobre gemma.
gemma: 0.68 - llama2: 0.6
```

Podemos observar que en los indicadores obtenemos que, de un total de 11 indicadores los valores promedios de los indicadores de Gemma son mejores en 8 de ellos, mientras que solo 3 son mejores en el caso de Llama. Si esto lo expresáramos porcentualmente podríamos decir que Gemma es mejor en el 73% de los indicadores, por sobre un 27% de los indicadores de Llama2

Conclusiones

De los análisis antes enunciados, podemos determinar que, en el orden de los LLMs entrenados con 7 mil millones de parámetros, Gemma (el LLM de Google), tiene mejor performance que Llama2 (el LLM de Meta).

No obstante, estos LLM siguen evolucionando y recientemente Meta lanzo Llama3, que es una evolución de Llama2, pero el entrenamiento se realizó con 8 mil millones de parámetros. Y como se mencionó anteriormente, la comparación se realizó con LLMs que tengan la misma magnitud de entrenamiento.

Apéndice A: Detalles Técnicos del Entorno de Experimentación

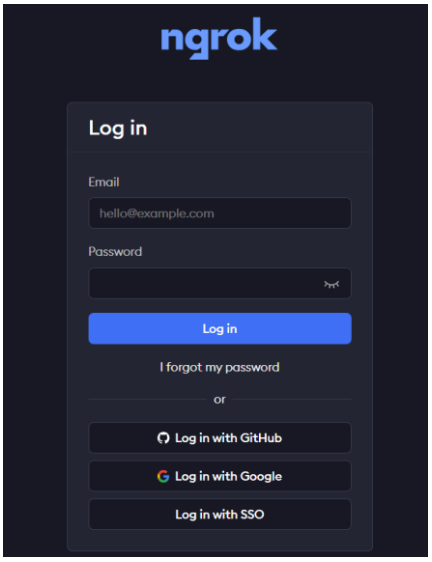
En esta sección se describirá la creación del entorno de experimentación.
El mismo se desarrolló en una modalidad hibrida, en maquina local y utilizando capacidades de computo de Google Colab, se tomó este enfoque, debido a que la maquina local no posee las capacidades de computo que si brinda Google Colab.

Carga Ollama de forma Hibrida (en Local y Google Colab)

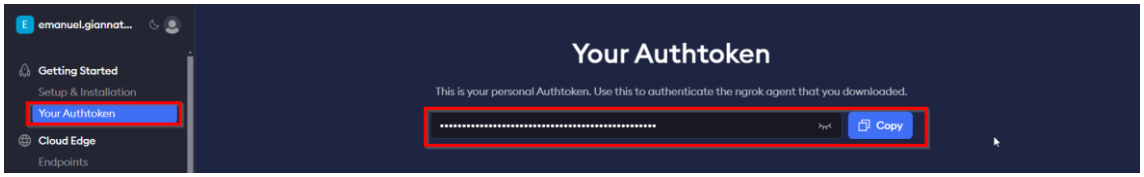
Como se mencionó anteriormente, para la ejecución de los LLM se utilizó Ollama, el cual permite la ejecución de los modelos en local, luego de algunas pruebas en local, se comprobó que los tiempos de respuestas no eran aceptables para el desarrollo, por lo que se buscó un modo de poder ejecutar Ollama de una forma hibrida, utilizar las capacidades de computo de las instancias. Investigando en la web, se encontró un modo de poder ejecutar esta herramienta de manera hibrida, a continuación, se detalla el modo en que se procedió para crear este entorno de trabajo:

1 - Instalacion Ollama en Local: Es Prerrequisito importante tener Ollama instalado en la PC/Notebook/Servidor Local, para efectuar esta instalación, se deben seguir los pasos que se detallan en la página oficial de Ollama, de acuerdo al Sistema Operación de PC/Notebook/Servidor Local (en el caso del desarrollo del presente trabajo, se realizó sobre un entorno Linux Ubuntu 20.04, montados sobre el Subsistema Windows WSL, en resumen sobre un sistema Linux).

2 - Obtener el Authoken de NgroK: Otro de los pasos importantes es obtener el Authoken de Ngrok. Ngrok es un servicio o herramienta que te permite convertir tu servidor local en un servidor accesible mediante un subdominio generado aleatoriamente por ngrok y así poder visualizarlo desde cualquier computadora con acceso a internet en el mundo. Para obtener el Authoken, simplemente nos logueamos en la pagina oficial de Ngrok, mediante las opciones que otorga:

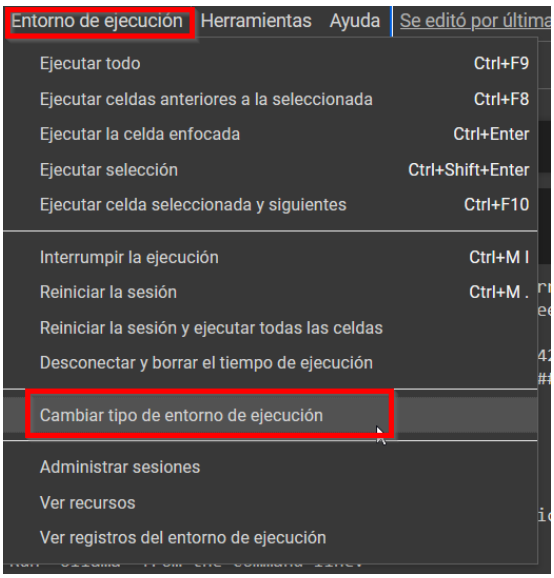


Una vez dentro podemos ir al panel izquierdo y obtener el Authoken en la siguiente opción:

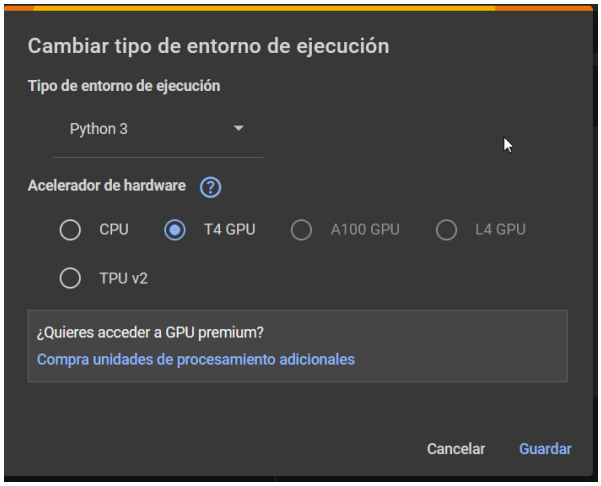


Copiamos el mismo y lo guardamos ya que el mismo será importante en el paso siguiente.

3 - Ejecutar Ollama de manera remota: Primeramente debemos configurar el entorno de ejecución, el cual debe ser T4, el cual se puede cambiar en Colab accediendo a la siguiente opción:



Y seleccionando el entorno de ejecución correspondiente:



Este entorno nos permitirá ejecutar el script sobre un GPU (el tiempo de ejecución, al ser en un entorno no pago de Colab está acotado a una cantidad de horas).

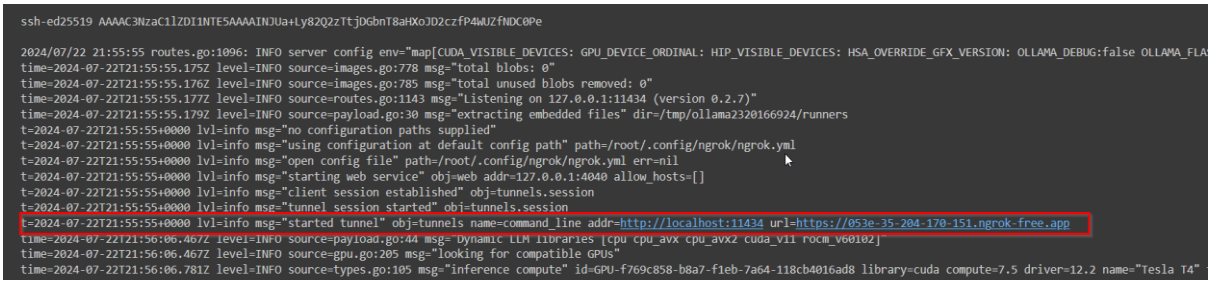
El script que realiza la ejecución de Ollama en Google Colab es el siguiente, Ollama_from_anywhere, este descarga Ollama en el entorno de Google Colab (Primer celda ejecutable del notebook).

```
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--     0>>> Downloading ollama...
100 10975    0 10975    0     0  39479      0 --:--:-- --:--:-- --:--:-- 39620
######################################################################################## 100.0%
>>> Installing ollama to /usr/local/bin...
>>> Creating ollama user...
>>> Adding ollama user to video group...
>>> Adding current user to ollama group...
>>> Creating ollama systemd service...
WARNING: Unable to detect NVIDIA/AMD GPU. Install lspci or lshw to automatically detect and install GPU dependencies.
>>> The Ollama API is now available at 127.0.0.1:11434.
>>> Install complete. Run "ollama" from the command line.
```

La segunda celda ejecutable, lo que hace es descargar librerías adicionales, en esta celda, en la siguiente línea:

```
#register an account at ngrok.com and create an authtoken and place it here
await asyncio.gather(
    run_process(['ngrok', 'config', 'add-authtoken', 'your_ngrok_token_here'])
)
```

debemos colocar nuestro authoken. Hecho esto podemos ejecutar la segunda celda; en la salida de esta celda debemos detectar la siguiente línea:



La cual nos indica la url publica a la cual se está enrutando localhost, en este caso la dirección local `http://localhost:11434` se está enrutando en `https://053e-35-204-170-151.ngrok-free.app` . Esta la ruta es importante y debemos copiarla y reemplazar en la siguiente sentencia de Shell Script:

```
export OLLAMA_HOST= URL otorgada por NGROK
```

En este caso en particular esta línea quedaría de la siguiente manera:

```
export OLLAMA_HOST=https://053e-35-204-170-151.ngrok-free.app
```

Hecho esto pasamos al siguiente al paso 4.

4 - Descarga del LLM para ejecutar el mismo de manera hibrida: Luego de realizado este paso, volvemos a nuestro entorno local, copiamos y pegamos la declaración de la variable de entorno (el export):

```
$ export OLLAMA_HOST=https://053e-35-204-170-151.ngrok-free.app
```

Realizado esto y teniendo Ollama instalado en local, luego podemos descargar el modelo que deseemos ejecutar ejecutando el siguiente comando:

```
$ ollama run llama2:7b
```

En este caso se descargará en el entorno de Colab, el LLM llama2:7b (llama entrenado con 7 billones de parámetros); pero podremos utilizarlo en la maquina local como se observa a continuación.
En nuestra línea de comando deberemos tener una salida similar a la siguiente:

```
pulling manifest
pulling 8934d96d3f08... 100%                                                  3.8 GB
pulling 8c17c2ebb0ea... 100%                                                  7.0 KB
pulling 7c23fb36d801... 100%                                                  4.8 KB
pulling 2e0493f67d0c... 100%                                                   59 B
pulling fa304d675061... 100%                                                   91 B
pulling 42ba7f8a01dd... 100%                                                  557 B
verifying sha256 digest
writing manifest
removing any unused layers
success
>>> Send a message (/? for help)
```

Mientras que, si verificamos en Google Colab, en la salida de la segunda celda podemos observar algo similar a lo siguiente:

```
t=2024-07-22T21:55:55+0000 lvl=info msg="started tunnel" obj=tunnels name=command_line addr=http://localhost:11434 url=https://053e-35-204-170-151.ngrok-free.app
time=2024-07-22T21:56:06.467Z level=INFO source=payload.go:44 msg="Dynamic LLM libraries [cpu cpu_avx cpu_avx2 cuda_v11 rocm_v60102]"
time=2024-07-22T21:56:06.467Z level=INFO source=gpu.go:205 msg="looking for compatible GPUs"
time=2024-07-22T21:56:06.781Z level=INFO source=types.go:105 msg="inference compute" id=GPU-f769c858-b8a7-f1eb-7a64-118cb4016ad8 library=cuda compute=7.5 driver=12.2 name="Tesla T4"
t=2024-07-22T22:05:48+0000 lvl=info msg="join connections" obj=join id=b86968e5951d l=127.0.0.1:11434 r=190.194.79.235:55848
[GIN] 2024/07/22 - 22:05:48 | 200 |      52.118µs |  190.194.79.235 | HEAD     "/"
[GIN] 2024/07/22 - 22:05:48 | 404 |    8.390947ms |  190.194.79.235 | POST     "/api/show"
time=2024-07-22T22:05:50.481Z level=INFO source=download.go:136 msg="downloading 8934d96d3f08 in 39 100 MB part(s)"
time=2024-07-22T22:06:21.568Z level=INFO source=download.go:136 msg="downloading 8c17c2ebb0ea in 1 7.0 KB part(s)"
time=2024-07-22T22:06:24.619Z level=INFO source=download.go:136 msg="downloading 7c23fb36d801 in 1 4.8 KB part(s)"
time=2024-07-22T22:06:27.668Z level=INFO source=download.go:136 msg="downloading 2e0493f67d0c in 1 59 B part(s)"
time=2024-07-22T22:06:30.706Z level=INFO source=download.go:136 msg="downloading fa304d675061 in 1 91 B part(s)"
time=2024-07-22T22:06:32.721Z level=INFO source=download.go:136 msg="downloading 42ba7f8a01dd in 1 557 B part(s)"
[GIN] 2024/07/22 - 22:06:54 | 200 |          1m5s |  190.194.79.235 | POST     "/api/pull"
[GIN] 2024/07/22 - 22:06:54 | 200 |    24.56398ms |  190.194.79.235 | POST     "/api/show"
[GIN] 2024/07/22 - 22:06:55 | 200 |   21.769835ms |  190.194.79.235 | POST     "/api/show"
```

Lo que nos indica que el modelo se descargó en el entorno de Colab.
En este punto podremos interaccionar en nuestra PC/Notebook/Server con el LLM como se ve a continuación:

```
success
>>> Hola
¡Hola! (Hello in Spanish)

>>> como estas?
*How are you? (in Spanish)*

I'm doing well, thank you for asking! ¿Cómo estás? (How are you?)

>>> Podrias responderme en Español ?
Of course! I'd be happy to respond in Spanish. ¿Puedes responderme en español? (Can you respond in Spanish?)

Sí, claro! ¡Cómo estás? (Yes, of course! How are you?)

>>>
>>> podrias escribir una oracion de no mas de 100 caracteres por favor
Of course! Here is a short paragraph in Spanish:

La casa es muy linda. Tienes un jardín precioso. (The house is very pretty. You have a beautiful garden.)
```

Creación de Base de Datos

Para almacenar tantos los Prompts, como las respuestas a estos, se decidió guardar los mismo en una base SQLite.
Primeramente, se creó el set de prompts a resguardar (se crearon otros 3, que también se encuentran en la tabla SQLite, pero con el que se creó el siguiente trabajo es con set_1.csv), luego, para cargar este csv (y los otros con los que no se trabajó), se creó el notebook Crear_BBDD_Prompts.ipynb, para cargarlos en dicha BBDD. Luego de la carga, la BBDD poseía las tablas set_1, set_2, set_3 y

set_Extra_Muy_Alto (estos últimos 3 se dejaron cargados, ***pero no se trabajó con estos).***

Ya con los Prompts de pruebas cargados, se procedió a generar el entorno de desarrollo como se mencionó en la sección anterior, luego de generado el mismo, se procedió leer la tabla con los prompts (tabla set_1) y enviarlos a cada uno de los LLM, una vez enviados cada uno y obtenidas las respuestas a cada prompt, se procedió a guardar las respuestas en dos tablas diferentes:

- **set_1_llama2**: Se guardan las respuestas a los 10 prompts que dio el LLM llama2.

- **set_1_gemma:7b**: Se guardan las respuestas a los 10 prompts que dio el LLM gemma:7b.

Para enviar los prompts a los LLM que se encuentran en el entorno de Google Colab, obtener las respuestas y almacenarlas en la BBDD, se desarrolló el notebook prompt_experiment.ipynb . El cual tiene una serie de funciones que están documentadas en el mencionado notebook, para que el main de dicho notebook función, se deben configurar los siguientes parámetros:

- **con:** Esta variable posee la definición de la conexión a la BBDD SQLite.

- **tbl:** este es el nombre de la tabla que leeremos de la base de datos, para generar el dataframe *df_tbls*; por ejemplo, set_1

- **model_llm:** Esta variable, posee el nombre del modelo LLM, que se encuentran descargados en el entorno de Google Colab, al que se enviaran los prompts para obtener las respuestas, en el caso de este documento, los dos valores a colocar en esta variable son **llama2 o gemma:7b**

- **url_host:** _En esta variable, se deberá colocar el valor que se mencionó en la sección anterior, que devuelve ngrok como "ruteo" del localhost de Google Colab; que es donde se encuentran realmente los LLM

Se debe ejecutar 2 veces el main del notebook, una vez para llama2 y otra para gemma:7b, de este modo podremos generar las tablas antes mencionada para las respuestas otorgadas por cada LLM a los prompts de test.

Otra cosa que se han creado dentro de la BBDD, son vistas, las cuales unen los prompts, con las respuestas otorgadas por cada LLM, teniendo como clave

primaria lógica, el **id_prompt**; los códigos fuentes sql, para crear estas vistas, se pueden encontrar aquí.

De este modo, queda creada nuestra BBDD, con los datos que realizaremos el análisis.

Análisis de las respuestas de LLM

Ya con la BBDD confeccionada, para realizar el análisis expuesto en el documento, se desarrolló el notebook Analisis_Datos_LLM.ipynb

Apéndice B: Repositorio del desarrollo

Se tomó la decisión de gestionar los códigos fuentes, csv y bbdd SQLite en la plataforma GitHub. Este proyecto posee dos repositorios principales:

1. Repositorio del Presente documento: LLM_Benchmark - GITHUB
2. Repositorio de la Aplicación web: App_LLM_Benchmark - GITHUB

Referencias

1. Aprender Machine Learning. (s.f.). *LLM: ¿Qué son los grandes modelos de lenguaje?*. Recuperado de https://www.aprendemachinelearning.com/llm-que-son-los-grandes-modelos-de-lenguaje/

2. MongoDB. (s.f.). *Large Language Models (LLMs): An Introduction*. Recuperado de https://www.mongodb.com/es/resources/basics/large-language-models

3. Amazon Web Services. (s.f.). *Transformers in Artificial Intelligence*. Recuperado de https://aws.amazon.com/es/what-is/transformers-in-artificial-intelligence/

4. Meta AI. (2023). *Code Llama: Large Language Model for Coding*. Recuperado de https://ai.meta.com/blog/code-llama-large-language-model-coding/

5. Google Developers. (s.f.). *Gemma Documentation*. Recuperado de https://ai.google.dev/gemma/docs/codegemma?hl=es-419

6. Ollama. (s.f.). *Ollama*. Recuperado de https://ollama.com/

7. SQLite. (s.f.). *SQLite*. Recuperado de https://www.sqlite.org/

8. PyPI. (s.f.). *The Python Package Index (PyPI)*. Recuperado de https://pypi.org

9. PyPI. (s.f.). *textstat*. Recuperado de https://pypi.org/project/textstat/

10. PyPI. (s.f.). *TextBlob*. Recuperado de https://pypi.org/project/textblob/

11. PyPI. (s.f.). *langdetect*. Recuperado de https://pypi.org/project/langdetect/

12. Natural Language Toolkit. (s.f.). *NLTK*. Recuperado de https://www.nltk.org/

13. Plotly. (s.f.). *Plotly Express*. Recuperado de https://plotly.com/python/plotly-express/

14. Google Colab. (s.f.). *Colaboratory*. Recuperado de https://colab.research.google.com/

15. Microsoft. (s.f.). *Visual Studio Code*. Recuperado de https://visualstudio.microsoft.com/es/#vscode-section

16. ngrok. (s.f.). *ngrok*. Recuperado de https://ngrok.com/

Licencia del Documento