
Proyecto Final de Estudios

Investigación y análisis de viabilidad de la implementación de un LLM como Enrutador de comportamiento de un Agente Virtual

Ingeniería Mecatrónica

Alumno

Valentin Lopez

valentinlopezandres@gmail.com

Director

Ing. Eric Sánchez

Resumen

La startup Hoomano estaba buscando una forma de hacer a su agente IA, Mojodex, más flexible y fácil de usar, por lo que, para comenzar, decidieron cambiar la forma en la que se decidía cómo el agente se iba a comportar, estudiando así, la posibilidad de pasar de una selección manual de dicho comportamiento a una automática hecha por el mismo agente IA.

El objetivo era colocar un LLM (Large Language Model) para que actuara de selector de dicho comportamiento o **enrutador de comportamiento**.

Dentro de este contexto, este documento presenta las técnicas y herramientas utilizadas como así el procedimiento que se llevó a cabo para acondicionar diferentes modelos y evaluarlos con el fin de determinar la viabilidad de dicha implementación. Dicho procedimiento de acondicionamiento y evaluación se llevó a cabo siguiendo dos posibles enfoques:

- Ajuste de los modelos pre-entrenados - **fine-tuning**
- Ingeniería de prompt sobre modelos pre-entrenados pero no ajustados - **prompt engineering**

Con respecto al primer enfoque (**fine-tuning**), para el proceso de entrenamiento mismo, se llevó a cabo una creación de datos de entrenamiento de forma sintética, debido a la ausencia de los mismos; pasando además por un proceso de validación manual para asegurar la calidad de estos datos sintetizados. La técnica de entrenamiento elegida fue **QLoRA**, debido a los beneficios que ofrece en ahorro de esfuerzo computacional y buen rendimiento en sus resultados.

Impulsado por los resultados del entrenamiento se intentó un segundo enfoque a través de un trabajo de **prompt engineering**, donde se investigaron las técnicas más extendidas y se realizó una evaluación comparativa con cada técnicas en diferentes modelos para determinar cual de ellas, y en que modelo, era la opción más adecuada para el caso de uso tratado (**enrutador de comportamiento**)

Como análisis complementario se realizó una comparativa entre modelos Open-Source y privados para determinar si es posible para este fin el uso de los primeros, en lugar de modelos privados. Las conclusiones de este análisis son de gran interés para ciertas empresas que optan por poseer ellos mismos los modelos de inferencias por razones de privacidad de datos.

Índice

1. Introducción	4
1.1. Objetivos	5
2. Herramientas y software utilizados	8
2.1. Agente IA - Mojodex	8
2.2. Amazon Web Service - Sage Maker	10
2.2.1. Entrenamiento	11
2.2.2. Inferencias	11
2.3. Hugging Face Hub	12
3. Enfoque de Fine-tuning	14
3.1. Métodos de Entrenamiento	14
3.2. Métodos PEFT	16
3.3. Dataset	17
3.4. Métodos de prompt engineering	18
3.5. Proceso de Validación	22
3.6. Entrenamiento	23
4. Enfoque de Prompt Engineering	29
4.1. Puntos de medición	29
4.2. Evaluación	31
5. Resultados	34
6. Conclusiones	38
7. Bibliografía	39

1. Introducción

Actualmente estamos atravesando una verdadera revolución entorno al campo de la Inteligencia Artificial, especialmente con los **Grandes Modelos de Lenguaje - (Large Language Models (LLMs) en ingles)**. Dichos modelos se están transformando con gran rapidez en una herramienta verdaderamente potente y versátil de la cual aun no se tiene un conocimiento completo acerca de todas sus posibles aplicaciones, ni sobre sus limites. Es un campo en plena investigación.

Uno de los casos de uso más extendido es el de los **Agentes Virtuales**. Sistemas automatizados que funcionan de manera proactiva para asistir a un usuario, a diferencia de los asistentes virtuales tradicionales que operan de manera reactiva. Si bien un LLM seria el eje central de un Agente de estas características, el modelo por si mismo no puede proveer una experiencia útil al usuario final. A fin de poder aplicarlos como un Agente viable es necesario encapsular estos modelos en un sistema que los utilice como motor de generación.

En el marco de este encapsulamiento, el sistema se encarga de solicitar la información necesaria al usuario, preprocesarla con el fin de que al ingresarla al LLM, el mismo se comporte de la forma esperada, y finalmente procesar la respuesta del modelo para adaptarla a un formato que el usuario pueda utilizar.

El estudio propuesto en este documento lo llevé a cabo en el seno de la startup Francesa [Hoomano](#) a fin de implementar esta solución en su producto principal, Mojodex, un Agente Virtual para profesionales. Hoomano se dedica a crear un Agente virtual orientado a grandes compañías. El agente es adaptativo, ya que Hoomano propone diferentes versiones del mismo agente, adaptado a cada usuario. Esto lo logra ofreciendo una selección de “**Tareas**” diferentes a cada usuario, que Mojodex es capaz de hacer, y las cuales se adaptan a las necesidades de dicho profesional. Por mencionar algunas, es capaz de: hacer resúmenes de reuniones, analizar documentos, generar emails de seguimiento, convertir ideas que el usuario proponga a un memo estructurado, entre otras.

1.1. Objetivos

Como se mencionó previamente, la forma en la que Mojodex funciona es haciendo que el usuario elija manualmente en una primera instancia, la tarea que se desea realizar, para luego pedirle que provea a Mojodex con la información respectiva a dicha tarea. Pero ¿qué pasaría si se pudiera simplemente empezar a hablar y Mojodex fuese capaz de intuir automáticamente lo que se está tratando de hacer dentro de sus posibles tareas? ¡Esa sería una característica muy interesante y útil! No solo ayudaría a ser más rápido la interacción, sino que también permitiría cambiar fácilmente entre tareas sin tener que regresar, hacer una nueva selección y empezar todo de nuevo.

Esta característica descrita previamente se llama “**routing**”, porque el sistema, basándose en las entradas que recibe por parte del usuario, dirige el comportamiento de Mojodex hacia la tarea adecuada. Es importante notar que este no es un problema de clasificación, ya que, la lista que contiene las tareas de Mojodex es dinámica, lo que significa que en cualquier momento podría añadirse una nueva tarea, o modificarse o eliminarse una tarea antigua.

El sistema debería ser capaz de seleccionar en todo momento el comportamiento adecuado, independientemente de las tareas contenidas en esta lista.

Por lo tanto, el objetivo de este estudio es determinar si es posible eliminar el menú donde se selecciona la tarea a realizar y colocar en su lugar un **enrutador o router** de comportamiento, el cual sería el encargado de seleccionar el comportamiento adecuado del agente a la tarea que el usuario está intentando realizar. La Figura 1 ilustra esta idea:

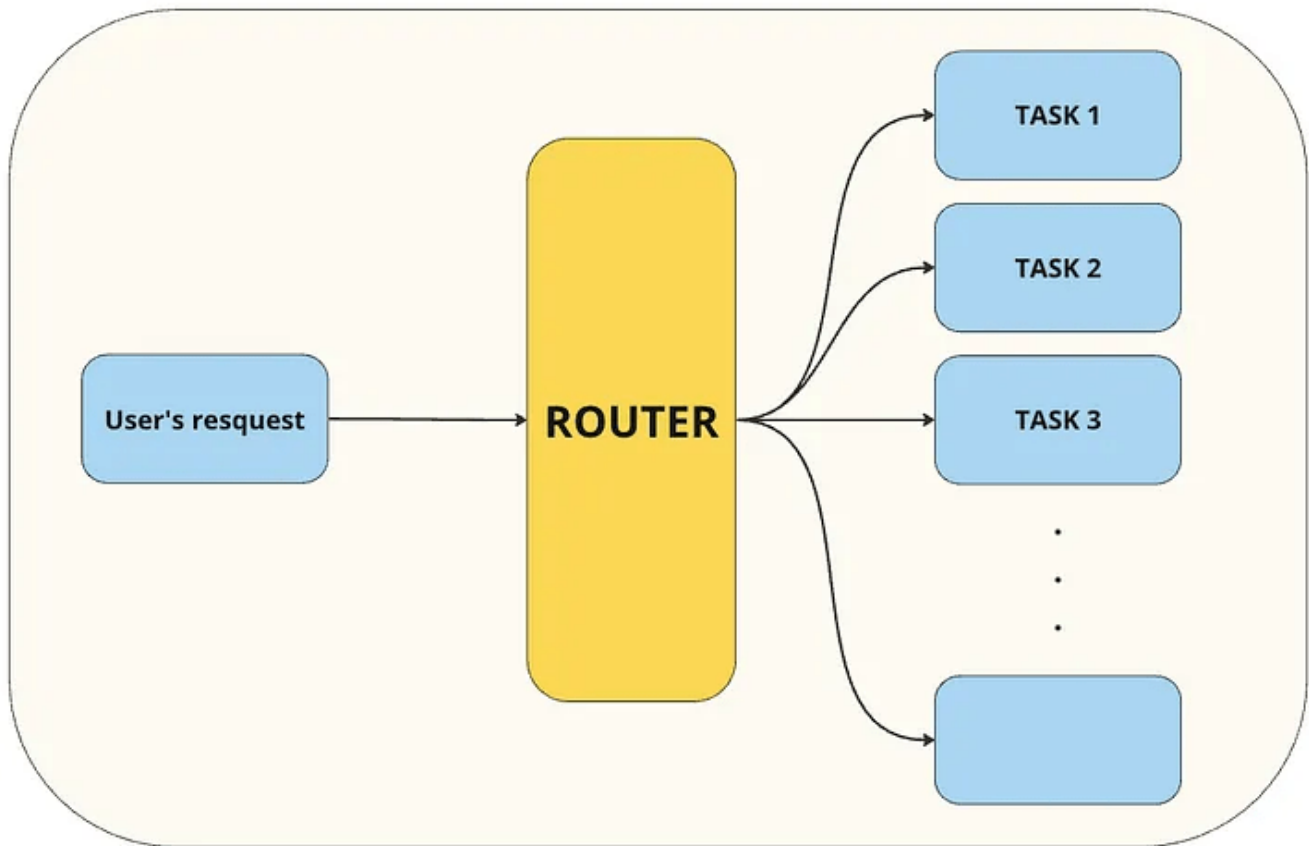


Figura 1: Concepto de “routing”

Por lo tanto se comprenden dos finalidades:

La primera, es determinar si es posible que un LLM actúe como enrutador para este fin. Para esto se realizó un estudio evaluando dos posibles caminos, como así, los beneficios y desventajas de cada uno:

- Realizar un entrenamiento o **fine-tuning**.
- Realizar un trabajo de **prompt engineering**.

La segunda, es evaluar si los modelos Open Source pueden tener, en el contexto de esta tarea, un rendimiento igual o superior a modelos privados (como por ejemplo el modelo GPT3.5 de la empresa OpenAI).

Es de interés la realización de este segundo análisis debido a cuestiones de privacidad empre-

sarial. Uno de los defectos de utilizar modelos privados como lo es GPT, es que el mismo se utiliza a través de una API de la propia empresa OpenAI. A pesar de que existan acuerdos de confidencialidad, la realidad es que la información sale del entorno de la empresa cliente para ser analizada en los servidores de OpenAI, lo que implica que la información puede potencialmente filtrarse o incluso ser usada para la mejora de estos modelos privados, lo cual, no es del agrado de muchas empresas.

2. Herramientas y software utilizados

A continuación se presentan las herramientas y entornos que se utilizarán durante el desarrollo:

2.1. Agente IA - Mojodex

Mojodex es un agente de IA diseñado tanto para grandes empresas como para individuos. Viene en versiones web y móvil (ver Figuras 2 y 3), ofreciendo una variedad de tareas predefinidas con el objetivo de optimizar la jornada laboral. Además, es capaz de ofrecer sugerencias en tiempo real a lo largo de la jornada y organizar los pasos a seguir luego de la finalización de ciertas actividades específicas a la ocupación del usuario.

Este agente puede asistir en una variedad diversa de **tareas**. Como la preparación de minutas de reuniones, correos electrónicos de seguimiento, toma de notas sobre nuevas ideas (memos), redacción de publicaciones en LinkedIn, entre otras tareas. Todas las interacciones tienen lugar a través de una interfaz de chat, similar a ChatGPT, donde Mojodex solicita la información necesaria para la tarea elegida por el usuario.

Una diferencia a destacar en la interfaz de la aplicación móvil, es que todas las interacciones se realizan mediante voz para un uso más conveniente.

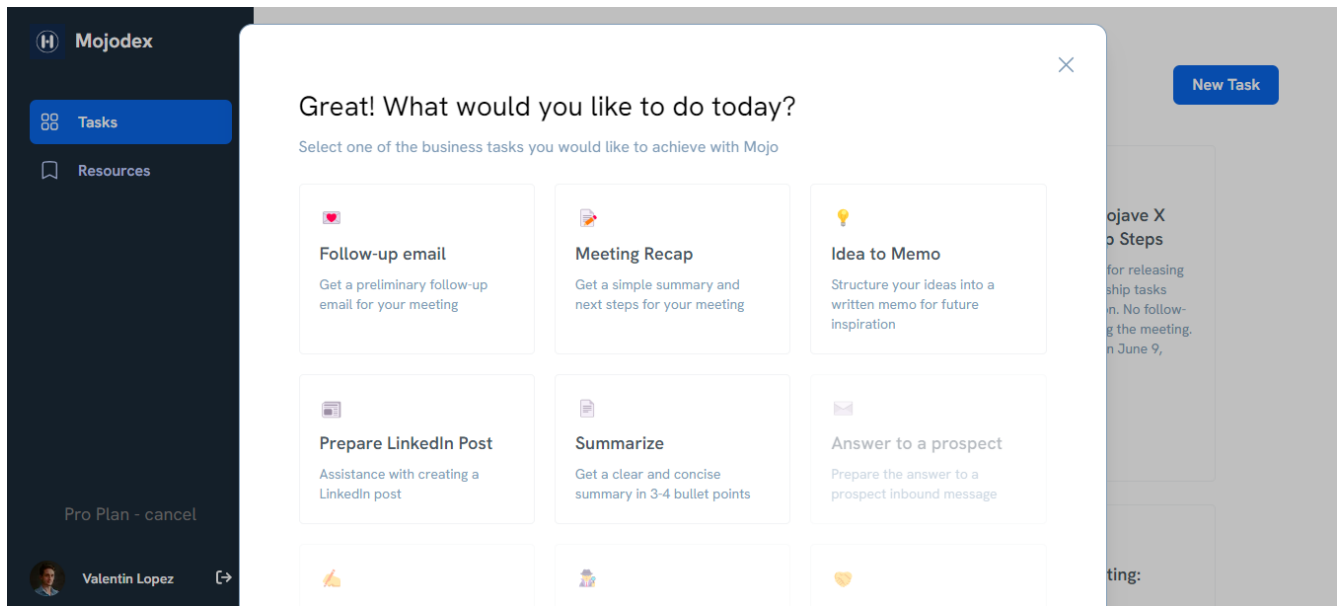


Figura 2: Mojodex versión web

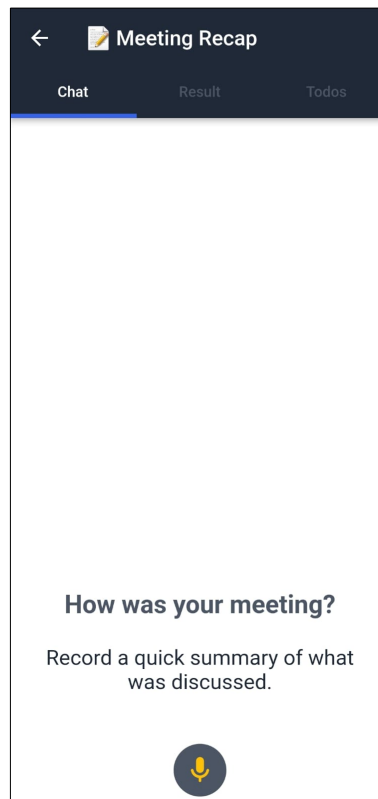


Figura 3: Mojodex versión móvil

2.2. Amazon Web Service - Sage Maker

Amazon Web Service (AWS) es una plataforma en línea bien conocida que ofrece soluciones de computación en la nube escalables y rentables. Uno de los principales productos de AWS es Sage Maker, un entorno completo diseñado para entrenar e implementar modelos de inteligencia artificial en la nube.

Sage Maker tiene varios espacios dedicados (ver Figura 4). Algunos de ellos permiten usar métodos de entrenamiento predefinidos, pero están limitados a solo algunos modelos. Los otros, y los que se usan en este trabajo, brindan más flexibilidad para hacer uso de un mayor número de modelos e implementar bucles de entrenamiento propios.

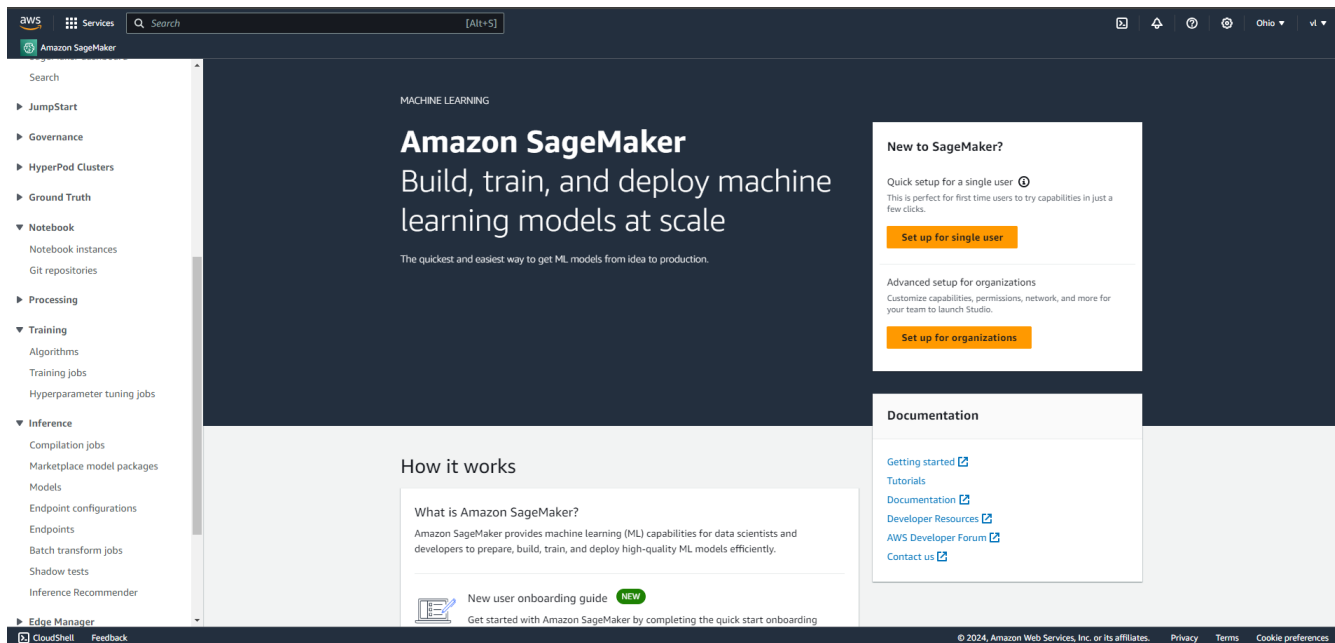


Figura 4: Vista general de la consola de AWS Sage Maker

Todos los modelos mencionados en este documento provinieron de Hugging Face Hub, así como los frameworks utilizados para construir los bucles de entrenamiento, entrenar los modelos en Sage Maker y posteriormente desplegarlos para su uso.

Como se explicó, Sage Maker tiene diferentes espacios dedicados:

2.2.1. Entrenamiento

Para propósitos de entrenamiento, Amazon Sage Maker tiene varios servicios disponibles. Uno que es de relevancia mencionar es el que se muestra en la Figura 4 como **Trabajos de Entrenamiento**. Un Trabajo de Entrenamiento es un servicio construido para cargar un modelo, los datos de entrenamiento y un bucle de entrenamiento personalizado de una manera compacta y sencilla.

Este servicio se puede lanzar mediante la propia interfaz de AWS o mediante un script escrito en una **Jupyter Notebook**. En este proyecto, se utilizó la opción del script debido a su conveniencia y simplicidad para cambiar las configuraciones del entrenamiento.

2.2.2. Inferencias

Para hacer uso de un modelo, es decir, realizar inferencias, es necesario realizar tres configuraciones en Sage Maker, la cuales se encuentran dentro del espacio llamado “Inferencias” (ver Figura 4):

- **Crear un modelo**

Un **modelo** en Sage Maker es un espacio donde se cargan los **artefactos** del modelo que se quiere usar. Los artefactos son todos los requerimientos para que el modelo funcione, principalmente se trata de un archivo .zip que contiene los **pesos** o parámetros del modelo y algunos scripts que describen como se pre-procesa o post-procesa la información, antes y después respectivamente, que sea tratada por el modelo mismo.

- **Crear la configuración del endpoint**

Cuando se habla de “desplegar el modelo”, se esta haciendo referencia a crear un **endpoint**. Un endpoint es un punto de acceso que corre en un servidor dentro de AWS que permite que el modelo sea llamado desde un script externo de forma de poder realizar inferencias.

La **configuración** de dicho endpoint es el espacio en Sage Maker donde se asocia el “modelo” creado previamente, para luego poder referenciarlo y utilizarlo desde el endpoint

mismo. También se define el tipo de hardware o **instancia** a usar y el número de dichas instancias, entre otras posibles configuraciones.

- **Crear el endpoint**

Una vez se tiene el modelo creado y asociado en la configuración del endpoint, simplemente se crea y activa el endpoint para su uso. La creación del endpoint mismo se realiza desde un espacio diferente en Sage Maker, ya que es posible crear múltiples endpoints usando la misma configuración.

2.3. Hugging Face Hub

Hugging Face es un gran repositorio en línea donde se pueden encontrar una gran cantidad de modelos para una amplia variedad de propósitos diferentes. Este trabajo se basó en la familia de **Grandes Modelos de Lenguaje (LLMs)** de Llama 2, Falcon. Familia de modelos que en la fecha de realización del presente proyecto lideraban la lista de los mas utilizados y con mejor calidad de resultados.

Las razones por las que elegí trabajar con el entorno y herramientas de Hugging Face fueron principalmente dos:

- **La biblioteca Transformers**

La biblioteca **Transformers** es una biblioteca de Python para interactuar con modelos alojados en Hugging Face Hub o localmente en una computadora. Esta biblioteca no solo proporciona la flexibilidad de trabajar con modelos creados y pre-entrenados en diferentes frameworks como Tensorflow o PyTorch, sino que también permite la creación de bucles de entrenamiento personalizados.

Además, permite la integración de otra de las bibliotecas construidas por el equipo de Hugging Face, la cual sirve para implementar métodos de **cuantización** y **Parameter-Efficient Fine-Tuning (PEFT)**, que fueron elementos clave para este proyecto.

- **El Kit de Herramientas de Inferencia de Hugging Face para SageMaker**

Gracias a este Kit desarrollado por Hugging Face en colaboración con Amazon, es posible integrar las ventajas de las bibliotecas de Hugging Face previamente mencionadas con los recursos disponibles en Sage Maker de una manera realmente intuitiva y sencilla.

En la siguiente previsualización de Hugging Face Hub se puede ver el repertorio variado de modelos para diferentes fines que se encuentran disponibles.

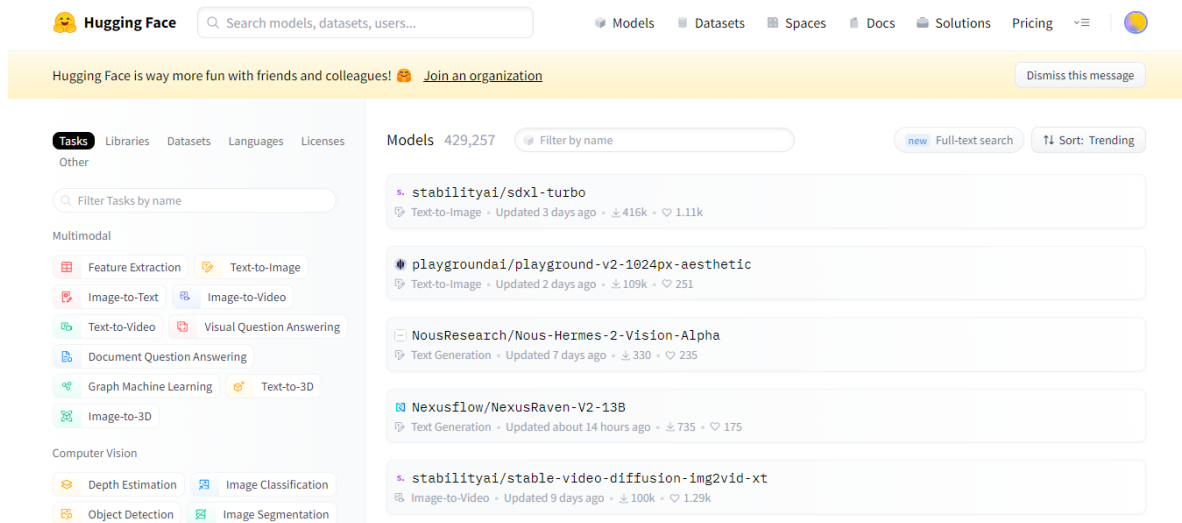


Figura 5: Vista general del Hugging Face Hub

3. Enfoque de Fine-tuning

Como se mencionó, se siguieron dos caminos para evaluar a los modelos. Se comenzará exponiendo el desarrollo completo para el enfoque de **Fine-tuning** y luego se explorará el enfoque de **Prompt Engineering**.

Se comienza por dar algunas definiciones importantes:

3.1. Métodos de Entrenamiento

- ¿Que es entrenamiento?

Un modelo IA es lo que se conoce como una red neuronal. Estos sistemas intentan emular en cierta forma como se comporta el cerebro humano, y esencialmente permiten encontrar, a través de una serie de cálculos matemáticos, relaciones realmente complejas entre datos de salida y datos de entrada.

Un ejemplo de esto podría ser la clasificación de una palabra como “buena” o “mala”. Si nos detenemos a pensar un momento, esto es algo realmente sencillo e intuitivo para el cerebro humano, pero para una computadora que trabaja bajo 1s y 0s, no existe una relación matemática normalizada que defina la connotación de una palabra. Por lo que, el problema aquí es como encontrar la relación entre la palabra y su connotación. Una forma de lograr esto, es de la misma forma que una persona lo aprende, basándose en muchos ejemplos. Lo que se hace es alimentar a la red neuronal con muchas palabras y por cada palabra decirle la connotación de la misma.

Una red neuronal esta compuesta por neuronas. Cada una de estas neuronas posee un parámetro denominado **peso**, el cual, no es más que un número que se utiliza para realizar una operación matemática sobre la palabra y obtener un resultado. A lo largo de este proceso de aprendizaje, estos pesos se actualizan hasta que se obtienen valores que en conjunto producen que cuando se ingresa un palabra se obtenga como resultado la con-

notación correcta.

Cuando se habla de **entrenar un modelo**, se esta haciendo referencia al proceso por el cual se somete a una red neuronal para actualizar los valores de los **pesos** de sus neuronas y así obtener los resultados o comportamiento del sistema deseado.

Esto evidentemente es una gran simplificación de lo que realmente sucede pero sirve para exponer lo que significa un **entrenamiento**. Esto no es solo para palabras, sino que también se puede entender para, análisis de imágenes, grabaciones de audio, etc.

○ ¿Que es un LLM?

LLM significa **Large Language Model** o **Gran Modelo de Lenguaje**. Estos modelos son redes neuronales que funcionan bajo una arquitectura conocida como **Transformers** que fue introducida en 2017 en el paper titulado “**Attention is All You Need**” [9]. Y se denominan “Grandes” porque poseen alrededor de miles de millones de parámetros o **pesos**. A modo ilustrativo se expone un esquema de la arquitectura mencionada:

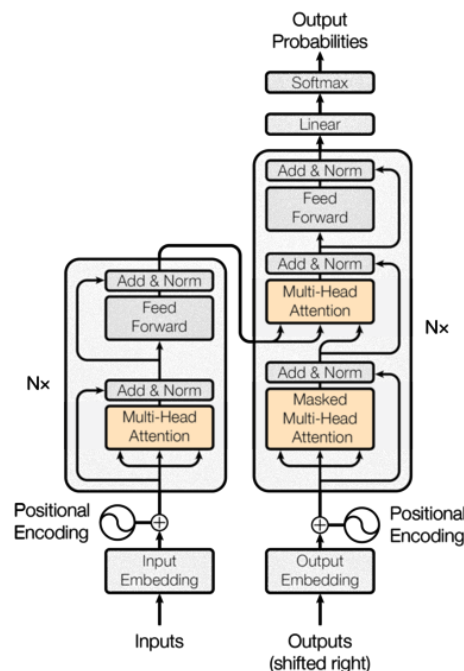


Figura 6: Arquitectura “Transformer”

- ¿Que es Fine-tuning?

Es importante notar que, hoy en día, cuando alguien menciona “entrenar un LLM”, en realidad está hablando sobre realizar un proceso de **Fine-tuning** en un modelo pre-entrenado. Como se expuso, un LLM posee miles de millones de parámetros, en particular los modelos analizados en este proyecto contienen de 7 mil millones a 70 mil millones de parámetros. Por lo que es intuitivo darse cuenta que crear un modelo y entrenarlo desde 0 implica un costo computacional y económico realmente elevado. Además para que se obtenga un modelo útil, es decir que el modelo tenga un conocimiento suficientemente basto y general para que se lo pueda usar para múltiples casos de uso, se debe contar con grandes cantidades de información de calidad y variada para entrenarlo. Por lo que, este proceso, es solo viable si es hecho por grandes compañía o instituciones con acceso a hardware específico, información y financiamiento.

En la practica lo que se hace es usar modelos que ya hayan sido entrenados y se realiza un proceso **fine-tuning**. Esto también es un entrenamiento, pero implica personalizar el modelo original para un caso de uso específico utilizando un nuevo conjunto de datos. Este proceso incluye modificaciones leves a los parámetros del LLM, aprovechando el entrenamiento previo. Es por eso que este proceso es realmente más económico y rápido que comenzar desde cero.

Las familias de modelos que se mencionaron previamente en este documento como lo modelos que evaluaron y que fueron obtenidos en Hugging Face Hub son modelos pre-entrenados.

3.2. Métodos PEFT

Si bien, realizar un proceso de fine-tuning, es mas económico que un entrenamiento completo, sigue siendo un proceso caro y demoroso, por lo que, lo que se suele usar y lo que se utilizó en este proyecto son un conjunto de métodos llamados **PEFT**.

Los métodos **PEFT** o **Parameter-Efficient Fine-Tuning**, permiten realizar un ajuste sobre un pequeño número de parámetros extras que se agregan al modelo, en lugar de ajustar el total de parámetros del mismo. Esto disminuye significativamente los costos computacionales y de

almacenamiento, mientras que proporcionan un rendimiento comparable a un modelo en el que se ajustaron todos sus parámetros. Esto lo hace más accesible para entrenar y almacenar LLMs en hardware de menor capacidad.

El método PEFT más conocido y el que se utilizó en este proyecto es:

- **Low-Rank Adapters - LoRA**

Para hacer que el ajuste sea más eficiente, el enfoque de LoRA es representar las actualizaciones de pesos con dos matrices más pequeñas (llamadas matrices de actualización) a través de la descomposición de bajo rango. Estas nuevas matrices pueden ser entrenadas para adaptarse a los nuevos datos manteniendo bajo el número total de cambios. La matriz de pesos original permanece congelada y no recibe más ajustes. Para producir los resultados finales, se combinan tanto los pesos originales como los adaptados.

Además de la disminución en costos y tiempo por la disminución de parámetros a ajustar, este método permite flexibilidad, ya que, es posible solo guardar los adaptadores entrenados para este fin. Si se quiere utilizar el modelo para otra finalidad, solo se deben cargar los adaptadores entrenados para ese otro caso de uso. De esta forma se pueden guardar los adaptadores para cada utilización que se requiera almacenando el modelo fuente una sola vez.

3.3. Dataset

Para poder aplicar fine-tuning, primero se necesitó obtener un conjunto de datos que fuese representativo para el caso de uso del **enrutador**. Hoomano, startup para la cual se hizo esta evaluación, tenía acceso a numerosas conversaciones con sus usuarios actuales que podrían haber servido para realizar el entrenamiento. Pero debido sus políticas de privacidad, se prohibía la utilización de esta información para tales fines. Una solución potencial que se discutió implicaba involucrar a nuevos usuarios y solicitar explícitamente su permiso para utilizar sus datos para construir este conjunto de datos. Desafortunadamente, debido a limitaciones de tiempo, esto quedó como un plan potencial para el futuro.

Como alternativa, se tomó otro camino y se creó un conjunto de datos sintético. La idea fue

utilizar otro LLM para generar conversaciones completas entre un **usuario** y su **agente de IA**, etiquetando cada conversación con una tarea de una lista de tareas posibles para Mojodex creada colaborativamente con el equipo de Hoomano.

Esta tarea se llevó a cabo elaborando en primer lugar un **prompt** para que el LLM se comporte como el Agente de IA y el usuario, autocontestandose y genere así, conversaciones en un formato específico que emule una interacción real entre un usuario y su agente de IA. Con este fin se investigó todas las técnicas más conocidas de **prompt engineering**.

3.4. Métodos de prompt engineering

Realizar **prompt engineering** consiste en crear instrucciones claras y precisas para la IA. Se trata de dar las palabras correctas para que la IA entienda lo que queremos y responda con precisión. Es importante usar las mejores palabras y estructura para evitar ambigüedades.

Algunos de los métodos mas conocidos son:

- **Few shots**

El enfoque **Few shots** consiste en dar ejemplos al modelos para lograr un aprendizaje contextual (ver Figura 7). Los ejemplos sirven como condicionantes para ejemplos posteriores donde nos gustaría que el modelo genere una respuesta.

Prompt:

A "whatpu" is a small, furry animal native to Tanzania. An example of a sentence that uses the word whatpu is:
We were traveling in Africa and we saw these very cute whatpus.
To do a "farduddle" means to jump up and down really fast. An example of a sentence that uses the word farduddle is:

Output:

When we won the game, we all started to farduddle in celebration.

Figura 7: Ejemplo de prompt con Few shots

Chain of Thoughts

Chain of Thoughts (CoT) o **cadena de pensamientos** permite capacidades de razonamiento complejas a través de pasos de razonamiento intermedios (ver Figura 8). Se puede combinar con el enfoque de **Few shot** para obtener mejores resultados en tareas más complejas que requieren razonamiento antes de responder. [5]

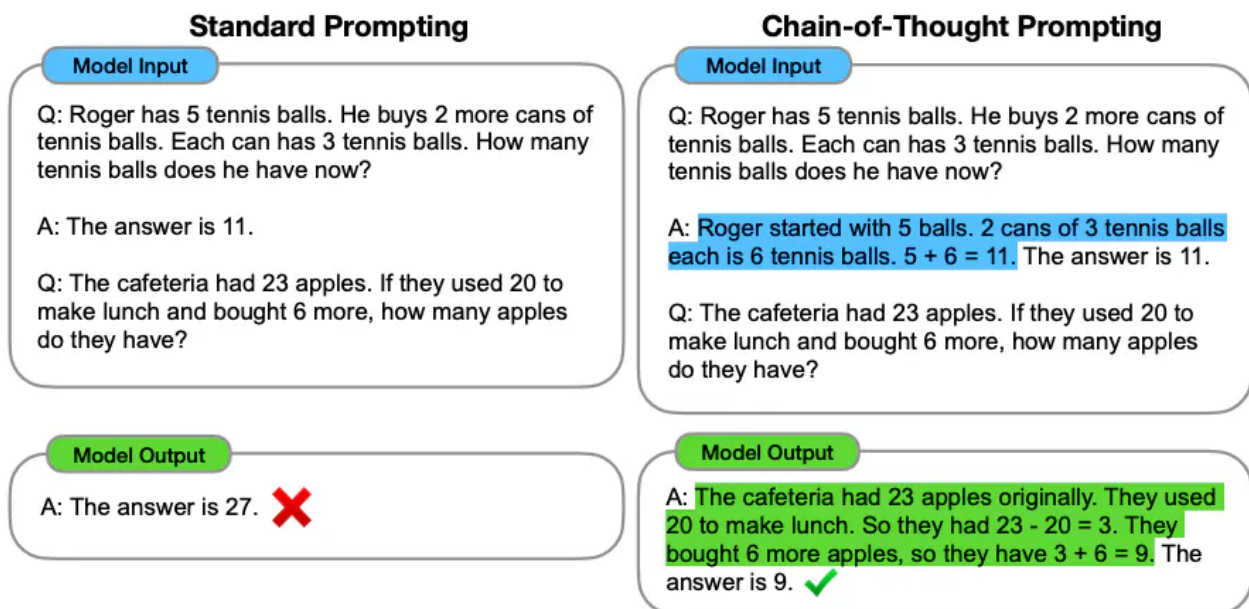


Figura 8: Ejemplo de prompt con cadena de pensamientos

Tree of Thoughts

Tree of Thoughts (ToT) o **Árbol de Pensamientos**, genera un árbol de pensamientos, donde dichos pensamientos representan secuencias de lenguaje coherentes que sirven como pasos intermedios para resolver un problema (ver Figura 9). Este enfoque permite que el LMM autoevalúe el progreso de los pensamientos intermedios hacia la resolución del problema a través de un proceso de razonamiento. La capacidad del LMM para generar y evaluar pensamientos se combina luego con algoritmos de búsqueda (por ejemplo, breadth-first search o depth-first search) para permitir la exploración sistemática de pensamientos anticipándose a posibles futuros pasos y retrocediendo en la exploración del espacio de búsqueda cuando se una solución incorrecta [6]

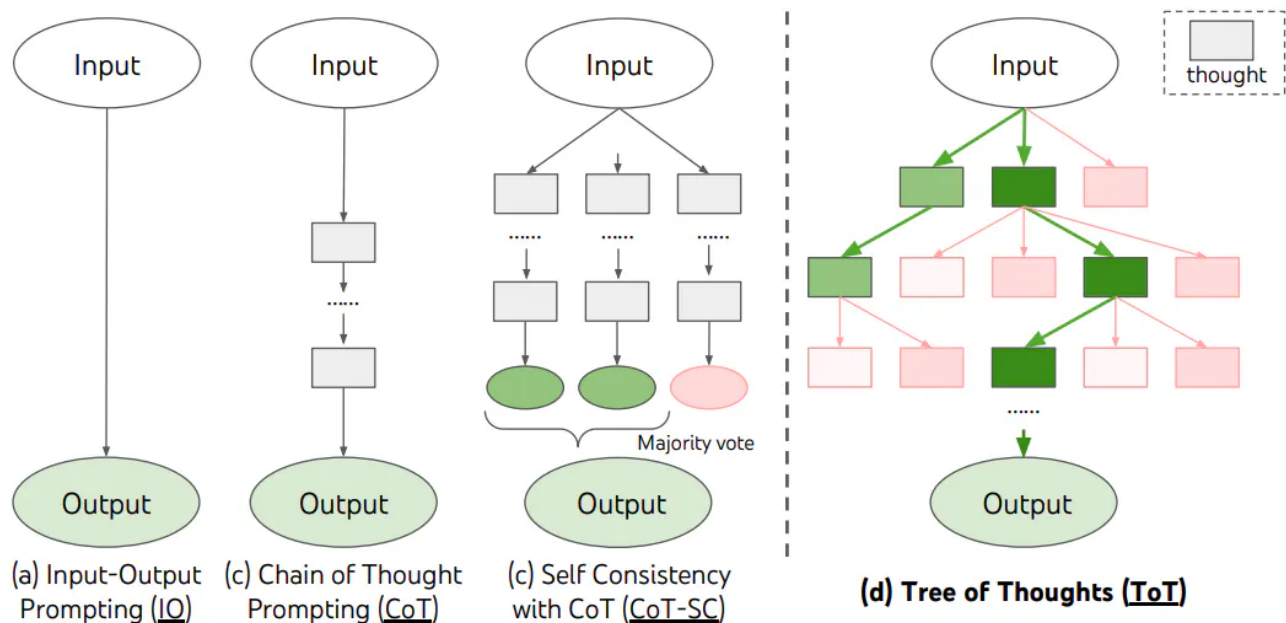


Figura 9: Diagrama Árbol de Pensamientos vs. otros enfoques

Después de algunas pruebas se determinó que de entre todas estas técnicas, la que funcionaba mejor para este caso de uso, era la técnica de **Few shots**.

En este punto se evaluaron varios modelos como posibles candidatos para generar las conversaciones. Entre estos, **GPT4** fue el que más respetó las instrucciones escritas y devolvió el tipo de formato que se necesitaba, así se terminó utilizando este modelo para generar este conjunto de datos. A continuación se puede visualizar el prompt utilizado:

You are a conversation generator.

You will be provided the name of a sales task. You will generate a conversation between a salesman user and their assistant that will lead to run this task.

[EXAMPLES]

TASK

answer_to_prospect : The user wants to prepare the answer to a prospect inbound message.

CONVERSATION

User: Hey, I got a message from a prospect. What should I answer?

Assistant: Hi, I'm your assistant. I can help you with that. What's the message ?

User: They are asking for a discount.

TASK

meeting_minutes : The user needs assistance to prepare a meeting minutes

CONVERSATION

User: I've just finished a meeting.

Assistant: How did it go?

User: It was great. We agreed on the next steps.

Assistant: What are the next steps?

User: We will send them a proposal.

Assistant: When will you send it?

User: Tomorrow.

TASK

send_information_email_to_client : The user wants to send information to a client by email

CONVERSATION

User: I've just finished a meeting.

Assistant: How was it?

User: Quite exhausting. We had to discuss a lot of details, many of which I

didn't know. But finally I think I managed to answer all the questions.

Assistant: Congratulations for that.

User: I need to inform my client John about the new product.

[END of EXAMPLES]

The target task is in [TASK] section.

Generate a conversation between a salesman user and their assistant that will lead to run this task.

No talk , just conversation .

[TASK]

{{task}} : {{task_description}}

Se creó un conjunto de datos de 2000 conversaciones. Una vez generados, se necesitó pasarlos por un proceso de validación manual antes de comenzar efectivamente el entrenamiento.

3.5. Proceso de Validación

Dado que el hecho de que conjunto de datos se creó sintéticamente, no se podía garantizar la calidad de todas las conversaciones. Así se creó una interfaz simple utilizando Python - PyQt5 con el fin de realizar una validación manual de dichas conversaciones (ver Figura 10). Debido a la gran cantidad de conversaciones a validar, este trabajo se hizo en conjunto con el equipo de Hoomano para dividir la carga y completar el proceso de validación más rápido.

La interfaz fue realmente sencilla. La conversación se mostraba a la izquierda, mientras que la tarea asociada o etiqueta se mostraba a la derecha. En la parte inferior, se contaban con diferentes botones que permitían la navegación (confirmar, retroceder, alterar la etiqueta, marcar la conversación para una revisión posterior o marcarla para eliminarla del conjunto de datos final).

1 out of 25

User: Hello
2. ### Assistant: Hello! How can I help you today?
3. ### User: We are planning a meeting next week to discuss our marketing strategy and I need your help to prepare for it. Would you be able to assist me with that?
4. ### Assistant: Of course! I can definitely help you with that. Let me start by asking you some questions about the meeting and what you want to accomplish.
5. ### User: The meeting is scheduled for next Monday at 15:00 at our company's headquarters. I would like to discuss how we can improve our marketing strategy and increase our visibility in the market.
6. ### Assistant: Great! Can you tell me more about your current marketing strategy and what you have tried so far?
7. ### User: Sure! We've primarily relied on social media ads and email campaigns. We have a good following on social media, but it doesn't seem to be converting to new customers.
8. ### Assistant: Hmm, that's interesting. Have you tried targeting specific demographics or interests?
9. ### User: Not yet, but that's definitely something we should consider.
10. ### Assistant: Agreed! It's also important to understand your competition and what they are doing to attract customers. Have you researched your competitors' marketing strategies?
11. ### User: Not in depth, but I do have some ideas based on what I've noticed from them on social media and their website.
12. ### Assistant: That's a good start! Can you tell me more about what you've observed?
13. ### User: Well, they seem to be using a lot of videos and images to showcase their products and services. They also have a strong presence on Instagram, which seems to be

⇒

Conduct competitor analysis

OKAY

CHANGE

BACK

TO BE CHECKED

MARK AS WRONG

Figura 10: Vista de la interfaz de validación

3.6. Entrenamiento

○ QLoRA

Para el entrenamiento, se utilizó el método QLoRA. QLoRA significa **Quantization and Low Range Adapters**[3] y es una variación del método LoRA presentado anteriormente.

Entrenar un modelo es costoso en memoria. Es por eso que en casos como este es una buena opción considerar la **cuantización**. Normalmente, los pesos de un modelo se representan numéricamente en 32 bits, cuantificar un modelo significa representar estos pesos en otra representación numérica que ocupe menos memoria. Podría ser una representación de 8 bits o una representación de 4 bits (conocida como QLoRA de 4 bits).

Como observación importante: solo los pesos del modelo se convierten, pero los pesos del adaptador LoRA permanecen en representación de 32 bits.

Es fácil darse cuenta de que el precio a pagar por ahorrar memoria es perder precisión, pero este no es exactamente el caso. Dado que estamos ajustando los pesos del adaptador LoRA, el valor obtenido después del entrenamiento tiene en cuenta esta pérdida de precisión, por lo que al final estos adaptadores entrenados alcanzan un valor que compensa la aproximación realizada durante la cuantización en los pesos del modelo.

○ **Modelos entrenados**

Como se mencionó se evaluaron modelos de la familia Llama de [Meta](#) y la familia de modelos Falcon del [Instituto de Innovación Tecnológica \(TII\)](#). Ambos son modelos de código abierto.

Por razones de coste, se eligieron solo dos de ellos para ser entrenados:

- ◇ Llama 2 7B
- ◇ Falcon 7B

7B significa que esa versión del modelo posee **7 billions of parameters** o **7 mil millones de parámetros**

El proceso de entrenamiento llevó 8 horas para cada modelo y debido a los resultados obtenidos, se tuvo que repetir los entrenamientos una segunda vez para probar diferentes configuraciones, así en total el proceso tomó 4 días.

A continuación se expone una comparativa de cada entrenamiento entre los resultados de estos modelos antes y después de ser entrenados con otras versiones de su familia y el modelo GPT3.5.

○ Resultados del primer entrenamiento

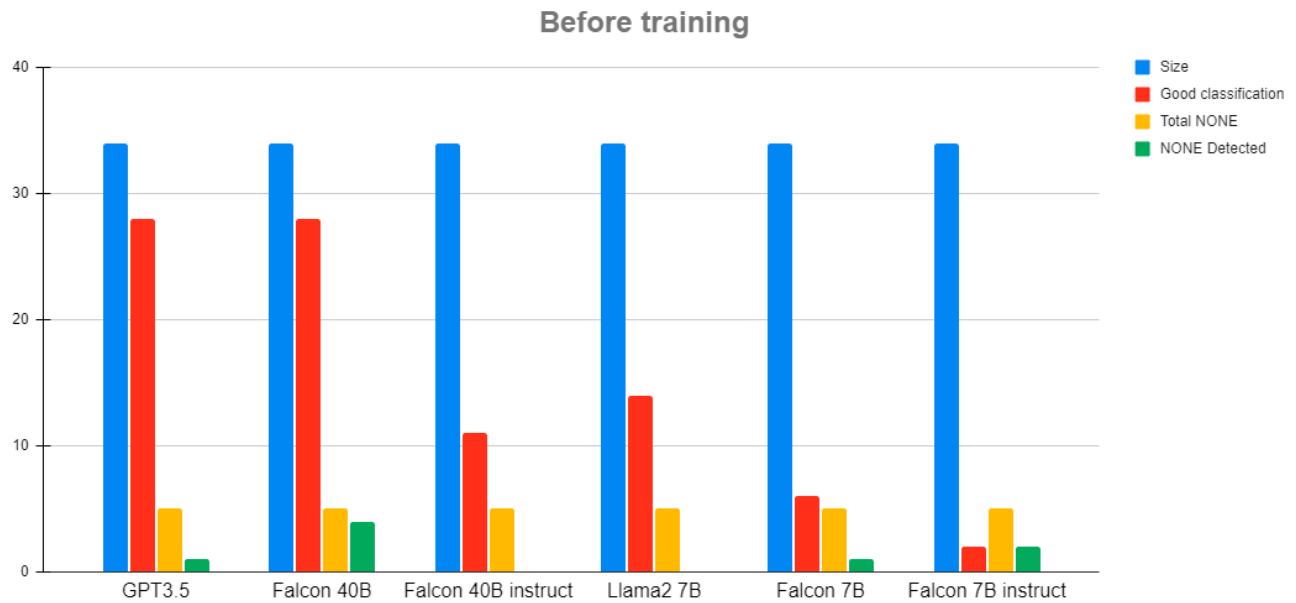


Figura 11: Resultados antes del primer entrenamiento

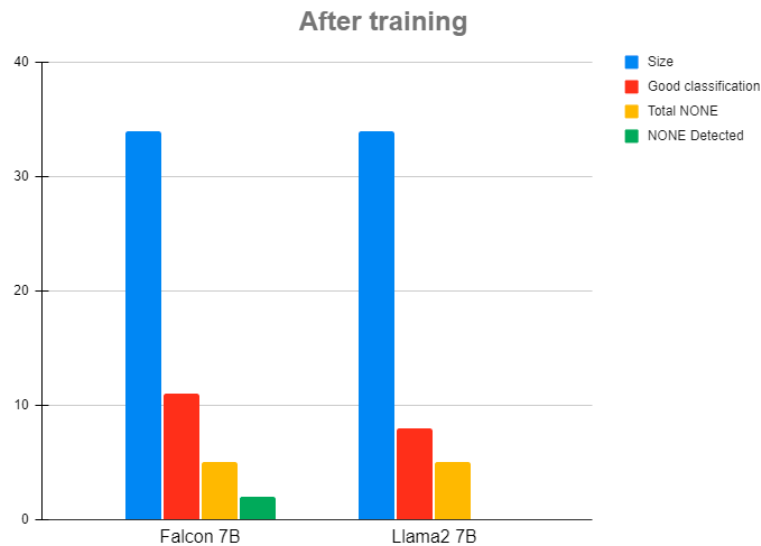


Figura 12: Resultados después del primer entrenamiento

El primer entrenamiento fue de 2 épocas utilizando un conjunto de datos de entrenamiento de 955 conversaciones, evaluado en un conjunto de datos de prueba de 34 conversaciones.

Como se mencionó, se ajustó la versión 7B (con 7 mil millones de parámetros) de los modelos. Sin embargo, también hay versiones de 40B y versiones **instruct** de dichos modelos. Los resultados mostrados comparan estas otras versiones de modelos de la familia Falcon con GPT3.5. Como era de esperar, los modelos con un mayor número de parámetros funcionan mejor sin ningún ajuste.

Además, Falcon 40B presentó un mejor rendimiento en comparación con Falcon 40B Instruct, a pesar de tener tamaños idénticos. Esto se debe a que la palabra **instruct** significa que este modelo ha sido entrenado para seguir instrucciones en lugar de simplemente generar texto. Esta diferencia existe debido al prompt. Un prompt diseñado para el modelo base no funcionará tan bien en la versión **instruct** del mismo modelo.

Del primer entrenamiento no se obtuvieron buenos resultados. Después del mismo, se apreció una mejora en los resultados de Falcon 7B pero un declive en el rendimiento para Llama 2 7B.

Otro aspecto que no mostró mejoras fue la detección de casos **None**. Hasta ahora, se ha explicado que el sistema debía para enrutar la tarea adecuada basada en la conversación del usuario. Sin embargo, durante una interacción real, es posible que el usuario quiera hacer algo que no esté considerado en la lista de tareas (tarea None). Aquellos casos que fueron etiquetados en el análisis como “None Detected”, son los casos de “tarea None” que el sistema identificó correctamente.

Esto no se logró para casi ningún modelo, incluido GPT3.5, y después del entrenamiento tampoco mejoró. Siguiendo estos resultados, se cambiaron algunos parámetros y se lanzó el segundo entrenamiento.

- Resultados del segundo entrenamiento

Before training

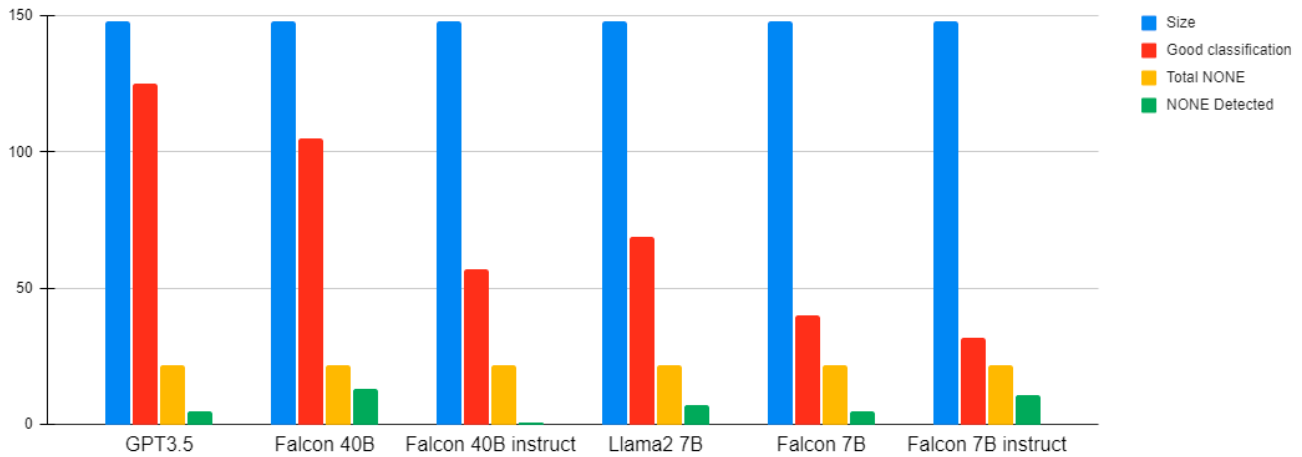


Figura 13: Resultados antes del segundo entrenamiento

After training

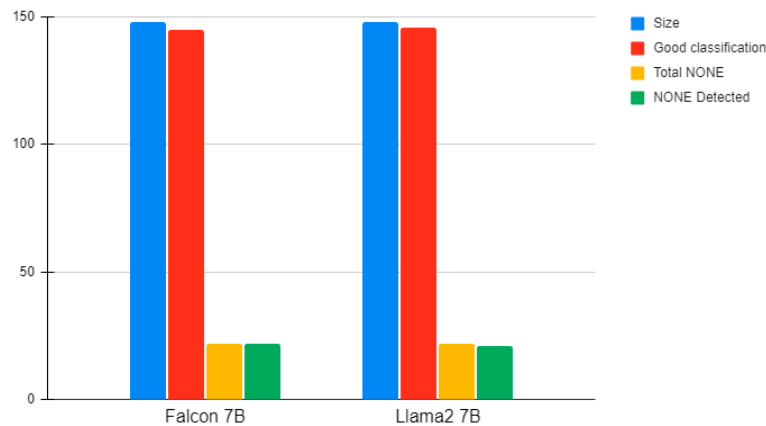


Figura 14: Resultados después del segundo entrenamiento

Aquí, se duplicaron el número de épocas y se aumentó el tamaño del conjunto de datos de entrenamiento a 1670 conversaciones, lo que resultó en resultados significativamente mejorados. Esto incluyó un rendimiento mejorado en las conversaciones bien enrutadas y una mejor detección de tareas **None**.

Desafortunadamente, debido a los altos porcentajes de acierto, hubo un riesgo considerable de que hubiese ocurrido el los que se conoce como **overfitting**, pero no en el sentido convencional.

Durante las pruebas en vivo usando Mojodex con el enrutador colocado, se observó una disminución significativa en el rendimiento. Si bien, a menudo realizaba selecciones correctas, mostraba una tendencia a no poder cambiar la elección a otra tarea o dejar de detectarla cuando la tarea había finalizado. Este comportamiento no era deseado para una aplicación destinada a conversaciones uno a uno.

- **¿Cuál fue el problema aquí?**

Contrariamente a las expectativas, el problema no fue el **overfitting** en el sentido convencional. Normalmente, ocurre overfitting cuando un modelo no aprende realmente a realizar la tarea prevista; en cambio, memoriza la respuesta correcta para cada entrada. Esto conduce a una alta precisión durante el entrenamiento pero a una precisión significativamente menor cuando se analizan nuevos datos. Este no fue el caso, porque los conjuntos de datos durante el entrenamiento y durante las pruebas comprendían conversaciones completamente diferentes. Por lo tanto, se puede afirmar que el modelo aprendió con éxito a generalizar su tarea a ejemplos que no había mirado previamente.

Lo más probable es que el problema radique en la calidad de estas conversaciones sintéticas. A pesar de pasar por un proceso de validación manual, estas conversaciones fueron creadas artificialmente. En consecuencia, es posible que el modelo se haya adaptado a este estilo de conversación estructurada, que no representa fidedignamente una conversación real con un usuario humano real.

Por lo tanto, después de estas conclusiones, se decidió evaluar el segundo enfoque que se introdujo al principio del documento.

4. Enfoque de Prompt Engineering

En este segundo enfoque se trató de aprovechar la flexibilidad y conocimiento general que ofrecen los LLM pre-entrenados sin realizar ningún fine-tuning o entrenamiento adicional.

Los LLMs han mostrado poder responder a diversas situaciones para las cuales nunca fueron entrenados. Esto se debe a la enorme cantidad de información con la cual se los entrena. Por lo que, realizando un buen trabajo de **prompt engineering** y usando las técnicas expuestas anteriormente es posible lograr que el modelo se comporte como el enrutador que buscamos sin necesidad de ajustar los pesos.

Primero se realizan las siguientes definiciones para presentar las métricas utilizadas para analizar los resultados de este enfoque.

4.1. Puntos de medición

En primer lugar, se introduce lo que se identificó como puntos clave en una conversación típica con el asistente de IA:

- **Punto de inicio**

Es la entrada del usuario durante el chat, que refleja la intención del usuario de iniciar una tarea específica. Por ejemplo, en el contexto de Mojodex, esto podría ser:

User: Please help me out to summarize my meeting

Esta entrada provocaría que Mojodex iniciara la tarea de “Resumen de reunión”.

- **Punto final**

Es la entrada del usuario durante el chat, que indica que la tarea en curso ha sido completada. Por ejemplo, podría ser:

User: That's fine , thanks for the help

- **Punto de cambio:**

Dependiendo de las solicitudes del usuario para completar una tarea, puede haber casos en los que Mojodex necesite cambiar a otra tarea en medio de la conversación. Estas situaciones sirven como **puntos de cambio**, ya que Mojodex cambia de tarea durante la ejecución de otra tarea. Por ejemplo:

User: I need to prepare an email for the participant of yesterday's meeting

Mojodex: Of course , I can help you with that. To prepare the email for the participant of yesterday's meeting , to start I need the following information: Who was the participant you had the meeting with yesterday?

User: Before giving you that information I would like to create the recap of the meeting to include it on the email

En este ejemplo, la última entrada del usuario sirvió como punto de cambio porque Mojodex debería cambiar de “**Preparar un correo electrónico de seguimiento**” a “**Resumen de reunión**”.

- **Selección de tarea:**

Mojodex debe distinguir cuando cambiar de una tarea a otra, pero también seleccionar la tarea correcta, ya que, la elección de la tarea incorrecta resultaría en proporcionar una respuesta no deseada al usuario.

- **Tiempo de inferencia:**

Por último, el **tiempo de inferencia** es un factor crucial. Incluso si el modelo tiene un rendimiento de precisión excepcional, si tarda una eternidad (más de 0.3 segundos), en seleccionar una tarea básica, no proporcionará una buena experiencia para el usuario final.

4.2. Evaluación

Para este segundo enfoque se procedió a recrear nuevas conversaciones por 2 motivos. Primero para evitar volver a obtener malos resultados posible poca representatividad de la realidad que tenían las conversaciones generadas sintéticamente y segundo para tener conversaciones que permitiesen medir estos nuevos **puntos** presentados previamente.

Esta vez, en lugar de generar conversaciones completamente sintéticas, se instruyó a GPT4 para que respondiera y actuase como Mojodex. Luego se comenzó a hablarle creando situaciones que ejemplificaran los puntos de medición explicados anteriormente. Como resultado, se obtuvieron conversaciones más realistas.

El objetivo en este enfoque fue evaluar qué tan bien funcionaban las versiones **instruct** de estos modelos en comparación con GPT3.5 usando las mismas entradas (conversaciones).

Una vez creadas las nuevas conversaciones que se utilizaron para evaluar los modelos. Se generó los **prompts** que permitirán hacer actuar al LLM como enrutador.

Los mismos se diseñaron haciendo uso de las siguientes técnicas de prompt engineering:

- ◇ Cero-Shot
- ◇ Few-Shots con Chain of Thoughts
- ◇ ReAct

Los modelos evaluados fueron los mismos de antes, con una adición de ultimo momento, el modelo Mistral 7B, modelo que fue lanzado en el mismo momento en el que se realizaba esta evaluación. Se eligió la versión **instruct** de todos estos modelos.

El proceso de evaluación consistió en pasar cada conversación a través de cada modelo con cada prompt generado. Posteriormente, se calculó una **matriz de confusión** para cada punto de medición con cada combinación de modelo y prompt (ver Figura 15), lo que permitió obtener información significativa de los resultados.

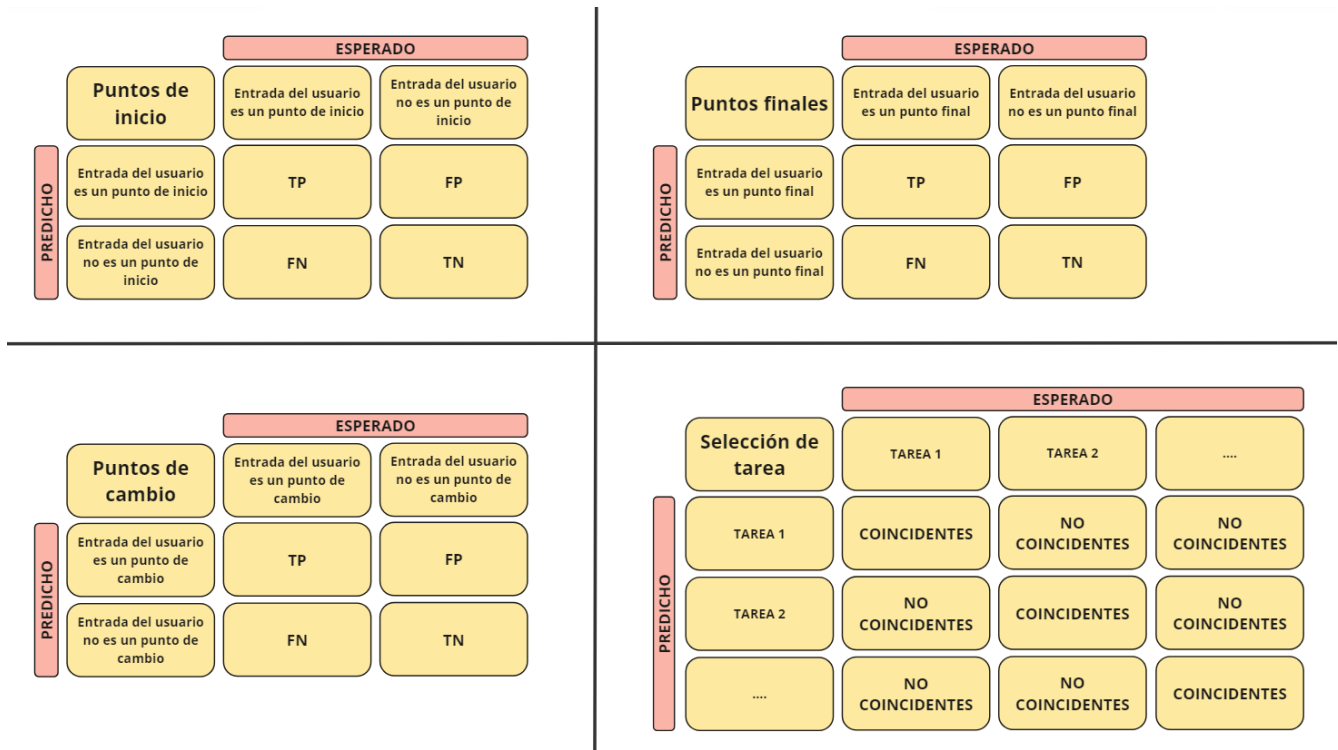


Figura 15: Matrices de confusión calculadas

Luego, utilizando estos resultados, se calcularon las siguientes métricas para cada una de las matrices de confusión obtenidas. Esto, con el fin de comparar tanto el rendimiento del modelo como el potencial de los métodos de prompt engineering utilizados para detectar cada punto de medición.

- **Accuracy:** permite analizar qué tan bien el modelo y el prompt logran hacer una detección correcta.

$$\frac{TP + TN}{TP + TN + FP + FN} \quad (1)$$

- **Precision:** permite analizar cuántos de los casos correctamente predichos resultaron ser verdaderos.

$$\frac{TP}{TP + FP} \quad (2)$$

- **Recall:** permite analizar cuántos de los casos positivos reales fueron predichos correctamente.

$$\frac{TP}{TP + FN} \quad (3)$$

5. Resultados

A continuación se exponen los resultados obtenidos en el segundo enfoque (Sección: 4 - Enfoque de Prompt Engineering), los cuales, mostraron ser mucho mejores que los que se obtuvieron en el primer enfoque (Sección: 3 - Enfoque de Fine-tuning).

Los modelos de código abierto hicieron un trabajo realmente bueno al elegir la tarea correcta y detectar los puntos de inicio, finalización y cambio.

El modelo que obtuvo mejores resultados fue Llama 2 chat 70B. En las Figuras 16, 17, 18 y 19 se exponen las matrices de confusión obtenidas y métricas calculadas para dicho modelo utilizando el prompt generado por la técnica ReAct y haciendo una comparativa con el modelo GPT3.5:

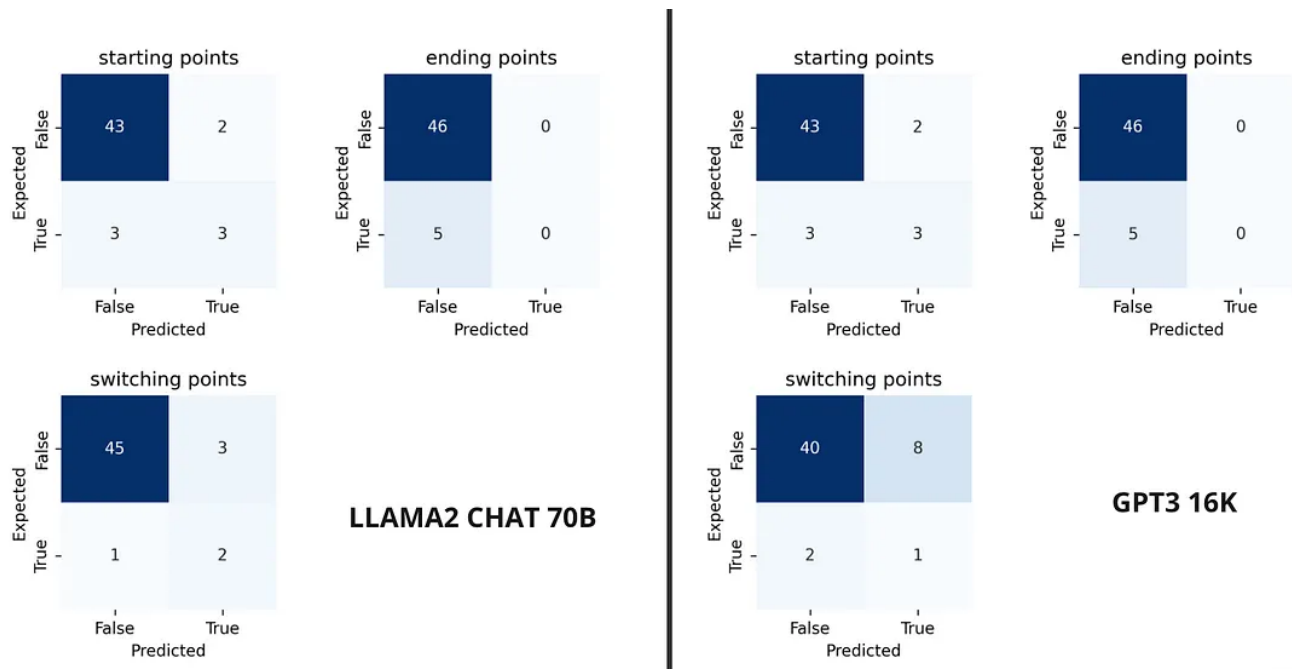


Figura 16: Matrices de confusión: Llama2 vs GPT3

LLAMA 2 CHAT 70B				GPT3 16K			
/	Starting Points	Ending Points	Switching Points	/	Starting Points	Ending Points	Switching Points
Accuracy	90.20%	90.20%	92.16%	Accuracy	90.20%	90.20%	80.39%
Precision	60%	0%	40%	Precision	60%	0%	11.11%
Recall	50%	0%	66.67%	Recall	50%	0%	33.37%

Figura 17: Métricas resultantes: Llama2 vs GPT3

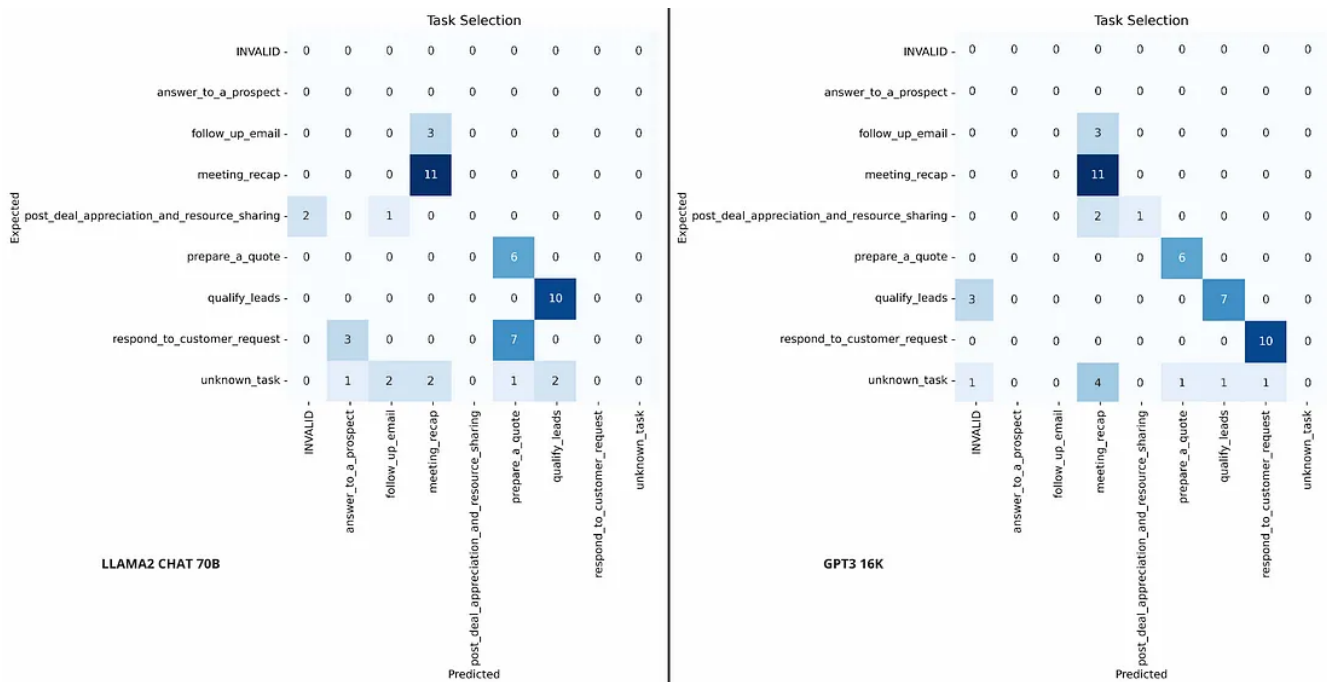


Figura 18: Matriz de confusión de “routing” de tareas: Llama2 vs GPT3

Task well predicted		
/	LLAMA 2 CHAT 70B	GPT3 16K
Accuracy	52.94%	68.63%
Precision	36.21%	62.81%
Recall	52.94%	68.63%

Figura 19: Task routing results: Llama2 vs GPT3

Estos resultados muestran, que se pueden usar modelos de código abierto y aún así lograr un nivel aceptable de rendimiento en las selecciones.

Sin embargo, hay otro aspecto a considerar. Aunque los modelos de código abierto se desempeñaron razonablemente bien en términos de predicciones, sus tiempos de inferencia varían significativamente en comparación con GPT3.5, como se expone en la Figura 20.

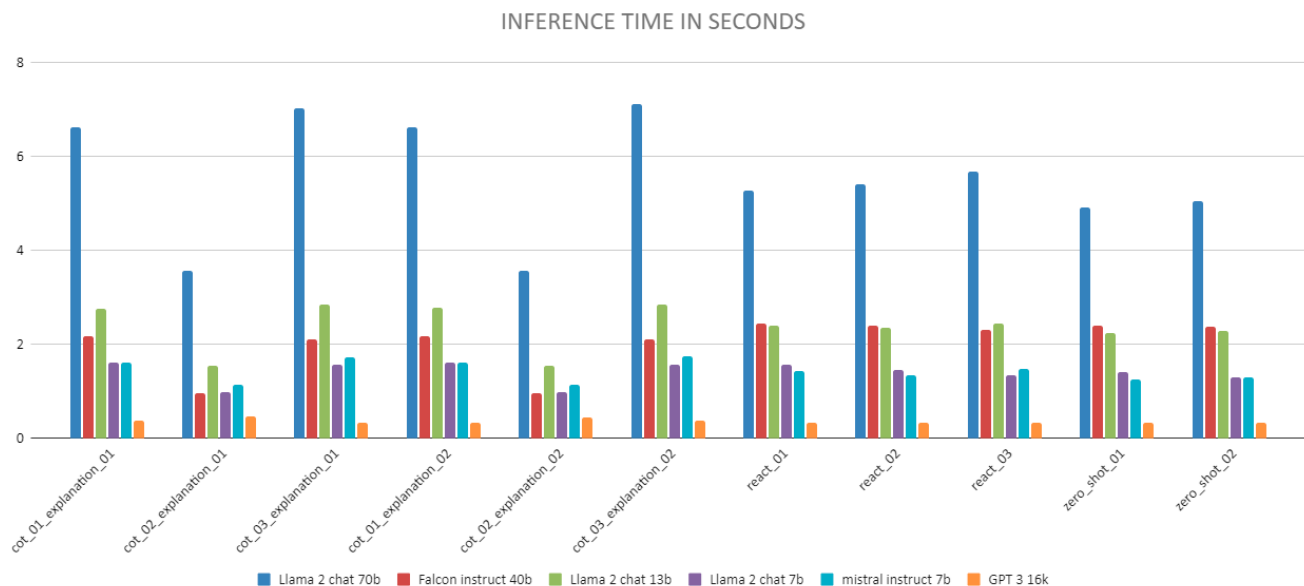


Figura 20: Tiempo de inferencia en segundos

En cuanto a los resultados de predicción, está claro que Llama 2 chat 70B ofreció los mejores

resultados entre todos los demás modelos de código abierto. Sin embargo, su tiempo de inferencia es seis veces más lento que el de GPT3.5 y mucho más de lo que se solicita para el caso de uso propuesto.

Un punto importante a tener en cuenta es que el modelo Mistral Instruct 7B produjo resultados que estaban a la par con los de Llama 2 chat 70B pero con tiempos de inferencia significativamente más rápidos (menos de 1 segundo). Sin embargo, no fue considerado como el modelo con mejores resultados por una razón clave que no se había mencionado hasta ahora: **el formato de salida**. Uno de los grandes desafíos que se tiene con los modelos de código abierto es asegurar el formato de salida correcto. En el caso de uso de este proyecto, se esperaba que el modelo devolviera solo el nombre de la tarea seleccionada, pero en muchos casos, no lo hizo, agregando información irrelevante que producían resultados en el sistema.

En la Figura 21, se expone que en casi todos los casos, Llama 2 chat 70B superó a Mistral Instruct 7B en este aspecto. Es una suposición razonable pensar que esta discrepancia se debe a la gran diferencia en el tamaño del modelo, siendo Mistral 7B un modelo de 7 mil millones de parámetros y Llama 2 chat 70B uno de 70 mil millones de parámetros.

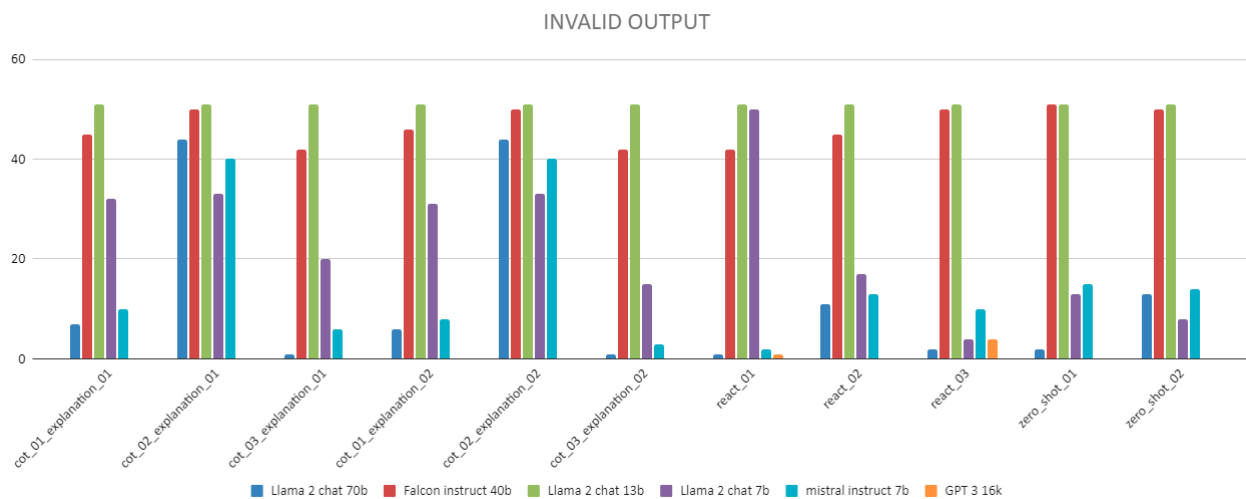


Figura 21: Número de respuestas con formatos inválidos producidas por cada modelo - prompt

6. Conclusiones

Como conclusión de ambos análisis, y de la evaluación de viabilidad se determina que la mejor solución para el enrutamiento es utilizar modelos privados como GPT3.5 o GPT4, utilizando un enfoque de prompt engineering. Cambiar a una alternativa de código abierto no era una opción viable en el momento en el que se realizó esta evaluación, principalmente debido a la diferencia significativa en el tiempo de inferencia. Sin embargo, esta última evaluación también subraya que los LLMs de código abierto podrían ser una buena alternativa, acercándose al rendimiento de las soluciones privadas.

Por otra parte, la principal desventaja del enfoque de Fine tuning radica en la ausencia de datos de alta calidad adecuados para el caso de uso específico presentado. Adquirir tales datos mejoraría significativamente la viabilidad de volver a intentar este enfoque en un modelo.

A modo de actualización de este proyecto, a principios del corriente año, 2024, la empresa Mistral ha conseguido lanzar al mercado nuevos modelos que pueden lograr superar no solo en tiempo sino también en resultados en pruebas genéricas al modelo GPT3.5 sin lograr superar aun al modelo GPT4. Esto sostiene lo dicho anteriormente que eventualmente los modelos de código abierto superarán a los modelos privados.

Bibliografía

- [1] *Sage Maker Documentation* <https://huggingface.co/docs/transformers/index>
- [2] *Transformer Library Documentation* <https://huggingface.co/docs/transformers/index>
- [3] *QLoRA: Efficient Finetuning of Quantized LLMs* <https://arxiv.org/abs/2305.14314>
- [4] *Prompt Engineering Techniques* <https://www.promptingguide.ai/>
- [5] *Chain-of-Thought Prompting Elicits Reasoning in Large Language Models* <https://arxiv.org/abs/2201.11903>
- [6] *Tree of Thoughts: Deliberate Problem Solving with Large Language Models* <https://arxiv.org/abs/2305.10601>
- [7] *Llama 2: Open Foundation and Fine-Tuned Chat Models* https://scontent-cdg4-1.xx.fbcdn.net/v/t39.2365-6/10000000_662098952474184_2584067087619170692_n.pdf?_nc_cat=105&ccb=1-7&_nc_sid=3c67a6&_nc_ohc=4mv90LTCsgEAX8YoQCt&_nc_ht=scontent-cdg4-1.xx&oh=00_AfCcuIaQnepnTevmka0gDyEtZjJtYQCcQ00UQCJXXVSZUA&oe=657C3EFF
- [8] *LLMs for Your Use Case: A Simple Evaluation Approach* <https://blog.hoomano.com/llms-for-your-use-case-a-simple-evaluation-approach-dbae6cb51a0d>
- [9] *Attention is All You Need.* <https://arxiv.org/abs/1706.03762>