GENERALITAT VALENCIANA

IES ABASTOS
Institut d'Educació Secundària i FP Abastos

SuperSmart: Tu supermercado inteligente

Por *Mario Carbonell* y *Jerónimo Sancho*

Curso de Especialización de Inteligencia Artificial y Big Data.
Curso Escolar 2022-2023
28 de mayo de 2023
Tutores: Chelo Richart y Sergio Vivó

Índice

1. Introducción

Bienvenido al proyecto integrador para el curso de especialización en inteligencia artificial y big data desarrollado por Mario Carbonell y Jerónimo Sancho, donde hemos documentado el trabajo e investigaciones realizadas.

Nuestro objetivo era la creación de una pipeline que nos permitiera recabar todos los datos necesarios para analizar los productos de varios supermercados muy conocidos y posteriormente entrenar modelo que permitiese el aprovechamiento de esos datos.

Para permitirnos alcanzar nuestros objetivos hemos desarrollado una arquitectura en cloud orientada al bajo coste y la escalabilidad, ya que nuestros recursos son limitados hemos recurrido a la capa gratuita de muchos servicios de primera línea. Esto ha supuesto que tuviésemos que adaptar muchas de las técnicas para capear las limitaciones de los productos utilizados.

Para la parte de big data hemos utilizado las soluciones gratuitas de MongoDB Atlas, que ofrece almacenamiento y dashboards, colas de mensajes de CloudAMQP, y un servicio de Node-RED para la comunicación con el servicio de scraping, este servicio se ha implementado con Python.

Para la parte de inteligencia artificial hemos realizado un fine-tuning de un modelo Open Source utilizando una técnica que está empezando a ganar importancia en el ecosistema NLP. El objetivo es utilizar los datos recogidos por el scraping diario y con ellos entrenar el modelo una vez por día de manera que los usuarios que accedan a él siempre cuenten con los datos del día anterior.

2. BigData

Para poder obtener y gestionar los datos necesarios para el entrenamiento e inferencia del modelo de IA de este proyecto, ha sido necesario diseñar una arquitectura BigData capaz de obtener datos mediante WebScraping de forma eficiente y fiable. Esos datos se han de almacenar en una base de datos adecuada para la posterior explotación. También se debe tener en cuenta que las webs pueden cambiar de formato y estructura en cualquier momento, por lo tanto se ha dotado a la estructura de mecanismos que detecten esos cambios y avisen al departamento de ingeniería de datos.

La arquitectura también ofrece dashboards para las distintas partes implicadas en todo el flujo de datos, desde la ingesta hasta el análisis.

La arquitectura que se ha implementado para la realización de este proyecto se ha diseñado reduciendo al máximo los costes sin penalizar la flexibilidad o la escalabilidad de la misma. Esta arquitectura es la idónea para proyectos de investigación con poco presupuesto o pequeñas empresas, entre otros supuestos.

2.1. Metodología y legalidad

La metodología de obtención de datos es el Web Scraping, que consiste en la obtención de datos de webs públicas pero de forma automatizada. Esta técnica es la misma que utilizan los buscadores web o los RSS feeds.
A la hora de realizar un scraping a una web hay que tener en cuenta la propiedad de los datos y la legalidad de las peticiones que son necesarias para obtener esos datos.

Teniendo en cuenta nuestro proyecto, los datos que se van a obtener son los relativos a productos publicados en las tiendas online de los supermercados, no se obtienen datos personales de clientes o datos de facturación de las compañías, y además para acceder a esos productos no ha sido necesario realizar ningún login. Por lo tanto no sería una actividad estrictamente ilícita.

2.2. Fuentes de datos

Para cumplir con el objetivo de este proyecto son necesarios todos los datos que identificativos de un producto, como el Nombre o la Descripción, los datos que definen la composición del producto, como los Ingredientes, Valores Nutricionales, información de Alérgenos o métodos de Uso y Conservación. También se han obtenido los distintos tipos de precio de cada producto, la estructura de categorías del supermercado y toda la información que pueda ser usada en un posterior análisis.

El proceso de selección de los orígenes de datos se ha iniciado realizando la búsqueda y explotación de datos de los supermercados Carrefour, Mercadona, Alcampo, Dia, MasYMas, Corte Inglés y Consum. Teniendo en cuenta el tipo, formato y cantidad de los datos obtenidos, se han descartado los supermercados Mercadona y Consum por ser necesarios modelos de

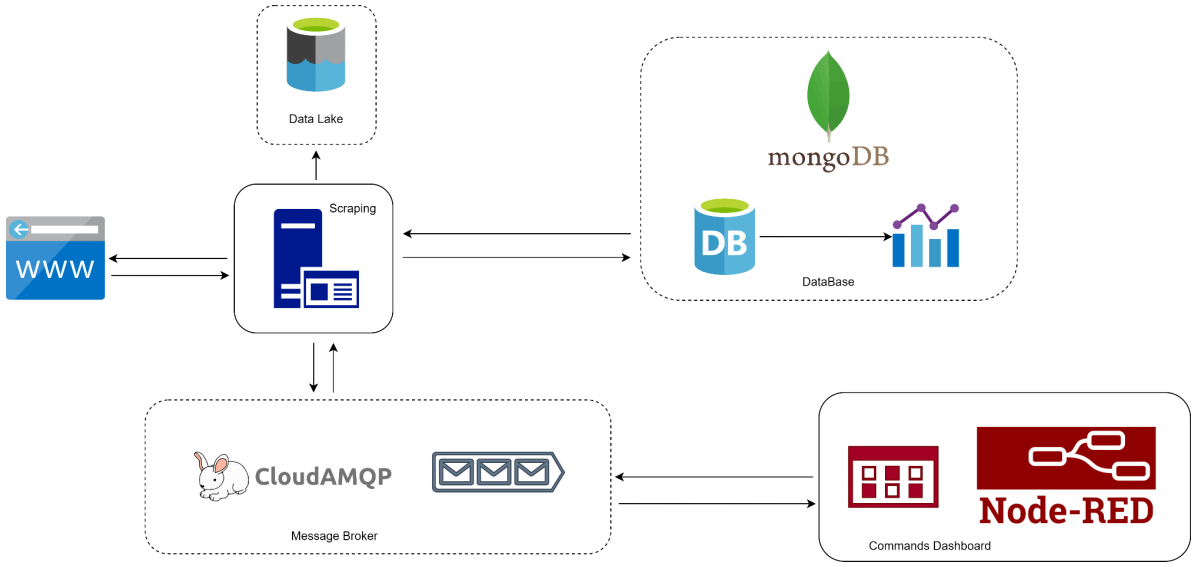
OCR y GPT para la extracción y verificación de los datos, y los supermercados Carrefour y Corte Inglés por la complejidad de la propia extracción ya que ambas webs tienen implementados sistemas anti robots.

De los supermercados restantes, Alcampo, Dia y Mas Y Mas, se han reducido las categorías objetivo para reducir el tiempo necesario para la extracción, y se ha realizado una extracción de cada uno de ellos cada día.

2.3. Arquitectura

Se han utilizado distintos servicios y softwares para llevar a cabo la extracción, transformación y almacenaje de los datos obtenidos.

Cada uno de los servicios o proveedores ha sido elegido, de entre las múltiples opciones que existen en el mercado, en base a las necesidades del proyecto y de los desarrolladores que lo han implantado.



2.4. Scraping

Para realizar la extracción de los datos y el procesamiento se ha implementado un proyecto con el lenguaje de programación Python. El proyecto se ha diseñado siguiendo las directrices de flexibilidad, modularidad e integridad, ya que es posible ejecutar el software en tres distintos modos, se pueden modificar los orígenes de datos sin modificar el core, y es capaz de detectar y gestionar los posibles errores o modificaciones en las webs de los supermercados sin perder los datos almacenados.

El proyecto se compone de dos partes, el dominio o core y la infraestructura. El dominio contiene las clases necesarias para definir el modelo por el que se regirán los datos obtenidos. También contiene el servicio que gestiona las ejecuciones de servicios de scraping. La infraestructura está compuesta por los servicios de scraping, de almacenaje y los servicios que consumen y producen mensajes en los brokers de datos.

La modularidad del software se ha conseguido desacoplando los servicios de scraping de la infraestructura y las funciones necesarias para la ejecución del scraping, y realizando la carga de los servicios de scraping de forma dinámica. Es decir, para añadir un nuevo origen de datos, se debe diseñar un nuevo servicio de scraping y situarlo en el directorio correcto del proyecto. Este nuevo servicio, es una clase que implementa una interfaz, que es común al resto de scrapings. Esta interfaz obligará a implementar dos métodos, el primero devuelve el código del origen de datos para que los datos obtenidos se puedan almacenar de forma organizada y se pueda mantener la trazabilidad. El segundo método inicia la ejecución del scraping, este recibe por parámetro, la función que se debe ejecutar cuando se ha obtenido un nuevo producto, otro parámetro es la función que se debe ejecutar cuando se ha producido un error de scraping y el tercero es el que se debe ejecutar para buscar el producto en la base de datos. De este modo, sin modificar ningún otro archivo del proyecto, se pueden añadir, modificar o eliminar servicios de scraping.

Dado que el objetivo ha sido implementar una prueba de concepto, se ha limitado el número de categorías obtenidas y se ha diseñado para que solo se obtenga la información que no se ha obtenido en anteriores ejecuciones del scraping.

Mediante parámetros de ejecución en línea de comandos, el proyecto se puede ejecutar en tres distintos modos para que sea flexible y adaptable a cualquier arquitectura. El modo **secuencial** ejecuta todos los servicios de scraping uno detrás de otro, el modo **service**, ejecuta un único servicio de scraping y el modo **stream** inicia el suscriptor que se encarga de gestionar los comandos que se reciben.

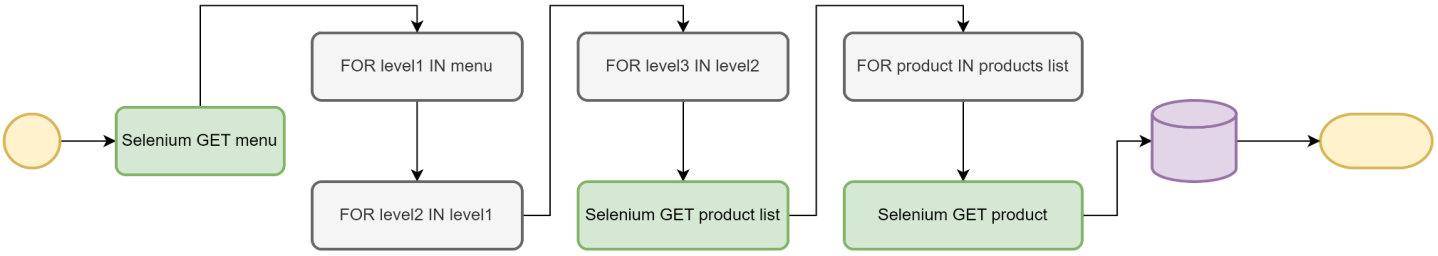
Por último, el proyecto produce cuatro tipos de datos, las **ejecuciones** son registros que se generan cada vez que se lanza un servicio de scraping, contiene la fecha de inicio y fin, y el identificador de la ejecución. Los **productos** son los datos relacionados a cada producto explotado de las webs

de los supermercados. Los **errores de scraping**, son registros que se generan cada vez que se esperaba un elemento HTML en la web y no está, o se ha producido algún tipo de error relacionado con el contenido de cada web. Por último, los **logs**, son las trazas que genera el software, permiten rastrear errores o ejecuciones incorrectas.

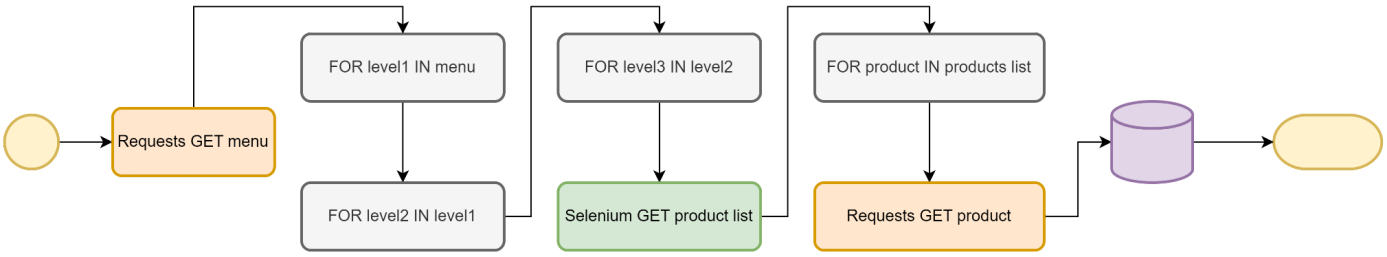
2.4.1. Servicios de scraping

Cada uno de los tres servicios de scraping ha requerido una estrategia totalmente distinta y adaptada a la estructura de la web del supermercado.

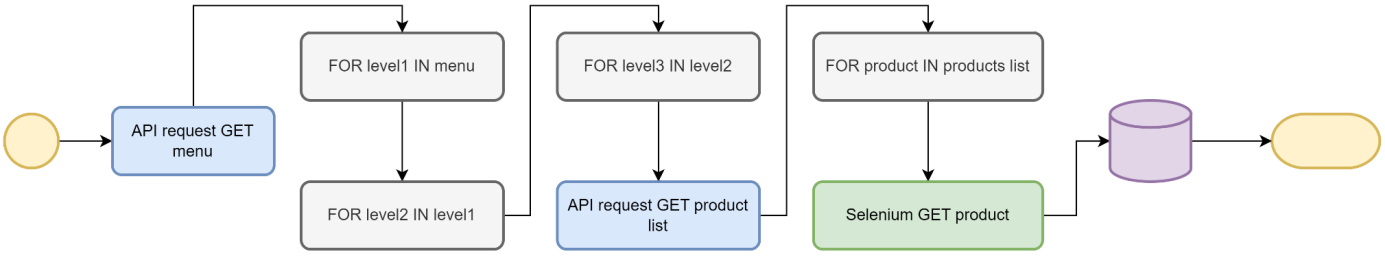
Para la implementación de estas estrategias, se han utilizado distintas herramientas, entre ellas Selenium y BeautifulSoup. Selenium es un software que simula la actividad de una persona real como usuario de un navegador web, para ello es necesario tener instalado un webdriver en el equipo, existen múltiples tipos de WebDriver, pero para este proyecto se ha utilizado uno que evita la detección de los mecanismos anti robot de los orígenes de datos. BeautifulSoup es una librería de Python que permite navegar y obtener datos de código HTML, normalmente se utiliza junto con la librería Requests, que permite realizar peticiones a servidores web.



Para el supermercado Alcampo se ha obtenido la estructura del menú superior utilizando Selenium con un timer para asegurarse de que se ha cargado todo el menú. Posteriormente se ha navegado hasta el nivel más interno del menú y se ha cargado el listado de productos, gestionando también la paginación del listado. Para cada uno de los productos se ha navegado a la web del detalle del producto y se ha obtenido la máxima información posible.



En la implementación del supermercado Dia, se ha utilizado la librería Requests y BeautifulSoup para obtener el menú y se ha utilizado Selenium para la paginación del listado de productos y la consulta del detalle de cada producto.



En el caso del supermercado Mas Y Mas, para la obtención de la estructura del menú se ha realizado una petición a una API y para obtener el listado de productos se ha utilizado la librería BeautifulSoup. Para obtener el detalle de cada producto se ha empleado Selenium.

En los casos que se ha utilizado selenium ha sido necesario también la utilización de timers de espera para simular de una forma más creíble que es una persona la que navega.

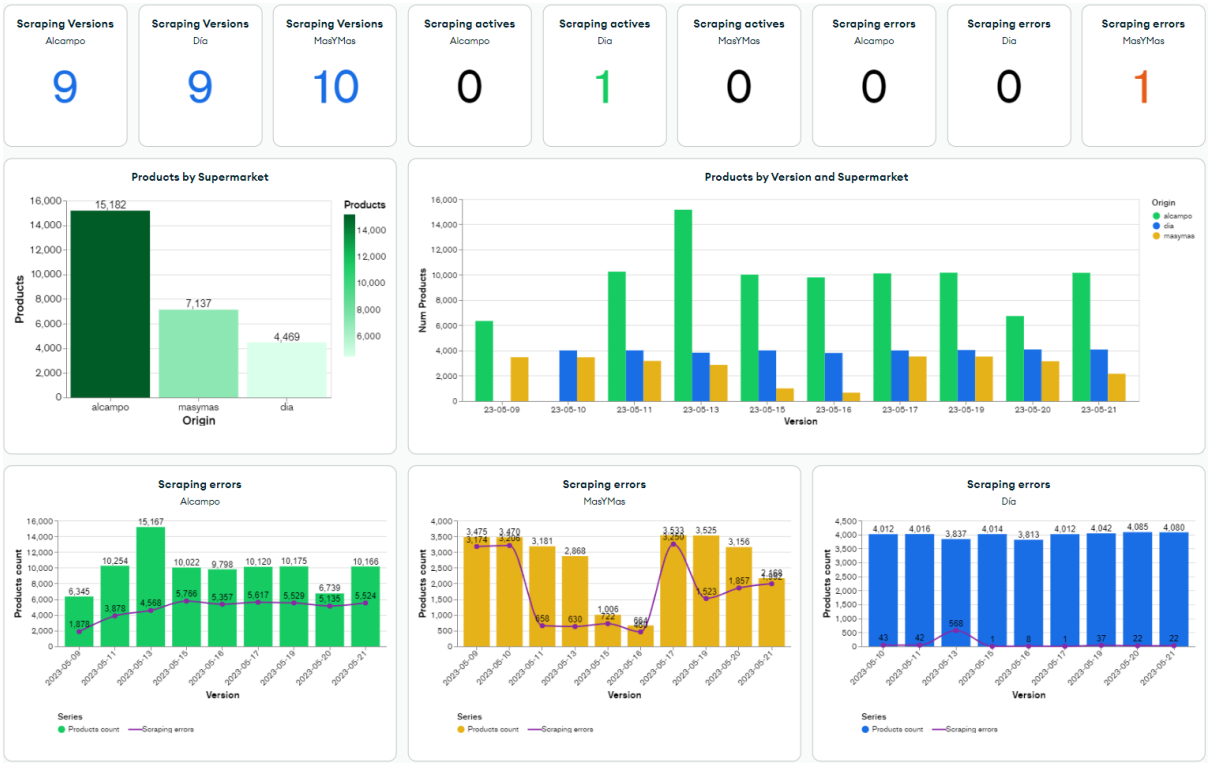
Todo el código de este proyecto python se puede encontrar en el repositorio de GitHub [super_smart]

2.4.2. MongoDB Atlas

MongoDB es una base de datos NoSQL orientada a documentos y MongoDB Atlas es el servicio cloud. Este servicio cloud ofrece diversas funcionalidades en su versión gratuita y sin límite de tiempo, entre ellas el almacenamiento y los dashboards, estos últimos ofrecen la posibilidad de analizar los datos en tiempo real.

En este proyecto se ha creado una base de datos para las ejecuciones con dos colecciones, una con un registro para cada ejecución y otra colección con los errores de scraping que se han producido. En la otra base de datos se han almacenado otras dos colecciones, una con todos los productos de todos los servicios de scraping y otra con la estructura de categorías de los supermercados. Cada uno de los documentos de la colección de productos está diseñado para almacenar cada una de las versiones o explotaciones de ese producto, facilitando el acceso para posteriores consultas.

2.4.2.1. Dashboard de Ingeniería de Datos



Este dashboard ha sido diseñado pensando en el ingeniero de datos, que se encarga de analizar la explotación e ingesta de los datos.

Para este proyecto, se ha configurado el dashboard para poder ver los servicios de scraping que se han realizado hasta ahora, los servicios que se encuentran activos en este momento y los servicios de scraping que han reportado algún error fatal que ha detenido la ejecución del scraping.
Por otro lado se muestra el número de productos distintos que se han obtenido hasta el momento de cada uno de los orígenes de datos y en el siguiente gráfico, los productos que se han obtenido por cada supermercado en cada versión.

También se pueden analizar los errores de scraping que se han generado, comparándolos con el número de productos que se han obtenido, y todo esto, viendo la progresión en el tiempo. Con estos gráficos se pueden detectar cambios en las webs de los supermercados.

2.4.2.2. Dashboard de Análisis de Datos

Este es un ejemplo de dashboard diseñado para realizar el análisis de todos los datos obtenidos de cada producto, para obtener nuevas oportunidades de negocio.

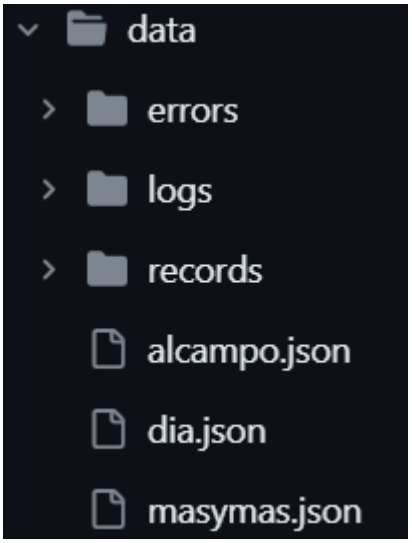
En este ejemplo se ha preparado el dashboard para que sea dinámico, por lo que es necesario aplicar un filtro de origen de datos al dashboard completo, de este modo todos los gráficos mostrarán valores de un mismo origen de datos.

En la sección superior, se muestran los gráficos de categorías, donde aparece el número de productos por cada una de ellas y en cada uno de los niveles.

En la sección inferior se muestra un ejemplo de análisis de precios, donde se muestra el porcentaje de aumento del precio de los productos respecto a la primera explotación de datos.

2.4.3. Data Lake

Todos los datos que ha generado el proyecto de scraping se han almacenado en MongoDB pero cabe la posibilidad de que se produzca algún error y que no se inserten los datos. Por ello se ha configurado un data lake para que los datos siempre estén almacenados en varios sitios. Este data lake está formado por directorios y ficheros JSON.



Dentro del directorio raíz data, se encuentran los siguientes subdirectorios:

- **errors**: contiene un fichero JSON por cada origen de datos y versión de explotación. Cada documento contiene un array JSON con los errores de scraping que se han detectado para un origen de datos y una versión concreta.
- **logs**: contiene un fichero por cada uno de los orígenes de datos y versión de explotación. Los documentos son ficheros de texto y cada una de las líneas del documento contiene la fecha y hora de generación de esa traza, el tipo de log, DEBUG, INFO o ERROR, y el mensaje de la traza.
- **records**: contiene un documento JSON por cada uno de los orígenes de datos y versión de explotación de datos. Y cada documento contiene un array JSON con todos los productos obtenidos en esa versión y orígen de datos en cuestión.

En el directorio raíz también se encuentra un fichero JSON por cada uno de los orígenes de datos. Cada uno de estos ficheros contiene un array JSON, y cada elemento de este array contiene el identificador y la fecha de inicio y fin de la ejecución.

Todos los ficheros del DataLake se pueden encontrar en el repositorio de GitHub [super_smart]

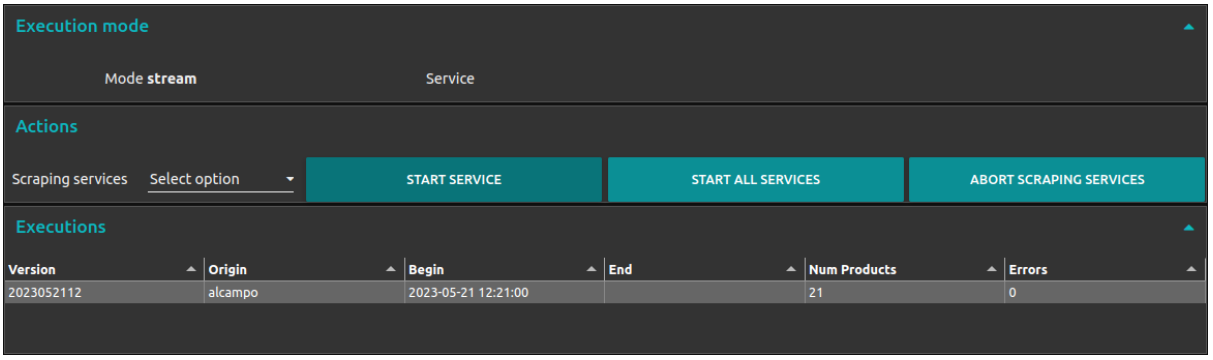
2.4.4. CloudAMQP

Esta plataforma ofrece un servicio RabitMQ cloud hasta 10 colas y 10.000 mensajes simultáneos por cola, de forma gratuita. También ofrece servicios más potentes pero de pago.

En este proyecto se utilizan las colas de mensajes para transmitir los comandos desde el cuadro de comandos hasta el proyecto de scraping y desde el proyecto de scraping se envían mensajes al cuadro de comandos.

2.4.5. Node-RED

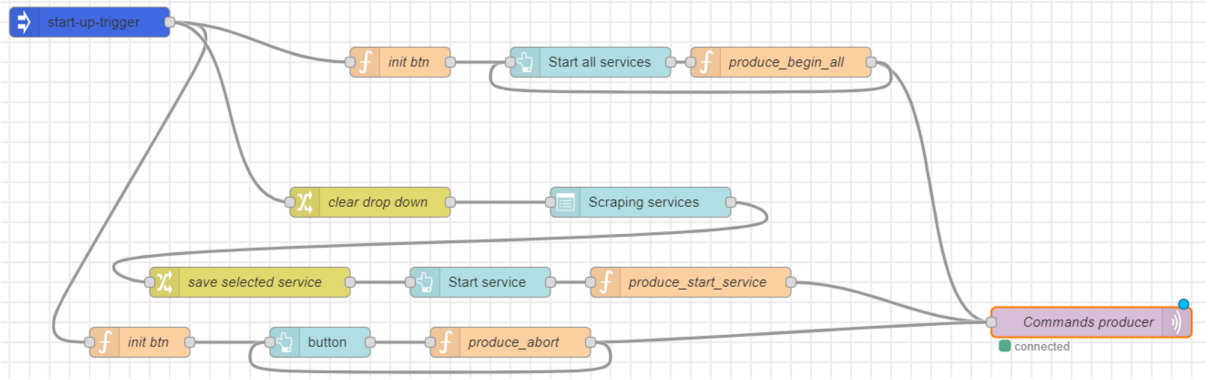
Node-Red es una herramienta de programación de flujos en entorno visual que permite, entre otras cosas, conectar dispositivos IoT, hacer peticiones a una API, consumir y enviar mensajes a un broker RabitMQ o publicar dashboards.



En este proyecto se ha utilizado Node-RED para crear un panel de comandos, desde donde se puede gestionar los servicios de scraping en tiempo real. Esto es posible debido a que el proyecto se puede ejecutar en modo stream, el cual, configura el proyecto para que envíe mensajes mediante un topic de RabitMQ al cuadro de comandos y ejecute los comandos que recibe a través de otro topic.

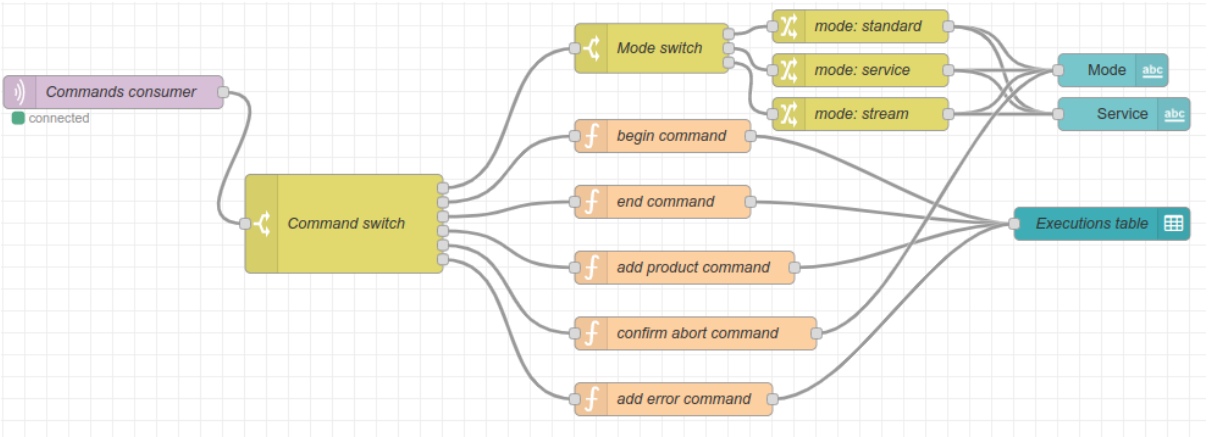
El cuadro de comandos que se ha diseñado para este proyecto es un ejemplo de todas las posibilidades que ofrece esta arquitectura. En la parte superior se muestra el modo en el que se está ejecutando el proyecto de scraping en tiempo real, ya que aunque el proyecto de scraping no se ejecute en modo stream, también se envían los mensajes al broker de datos.
En el apartado de acciones, se ofrece la posibilidad de detener los scrapings activos, ejecutar todos los scrapings en modo secuencial o ejecutar un servicio concreto.
En el apartado inferior, se muestra un grid de datos con las ejecuciones activas, donde se puede ver la versión de la ejecución, el origen de datos, fecha de inicio y fecha de fin de la ejecución. En la siguiente columna del grid se muestra el número de productos que se han obtenido, este dato se actualiza en tiempo real. Y en la última columna el número de errores de ejecución que han ocurrido.

Cuando se envía el comando de abortar, el servicio de ejecuciones se encarga de esperar a que se obtenga el siguiente producto para que no se pierda información, y posteriormente detiene la ejecución, teniendo en cuenta siempre la integridad de los datos. Dado que la detención de todos los procesos puede tardar un tiempo, cuando se han detenido, el proyecto de scraping envía un mensaje al cuadro de comandos indicando que se han detenido todos los servicios.
El flujo de ejecución que envía datos al proyecto de scraping es el siguiente:



Se puede observar que el flujo de ejecución se inicia cuando se despliega el dashboard de comandos, y que cada nodo de tipo componente del dashboard desemboca en la producción de un mensaje a un topic.

El flujo de ejecución que recibe los comandos del proyecto de scraping es el siguiente:



Cuando se recibe un mensaje se identifica el tipo de comando que es y se modifica el elemento del dashboard correspondiente.

El fichero fuente del flujo anterior se puede encontrar en el repositorio GitHub [super_smart]

2.5. Flujo de ejecución

Teniendo en cuenta la arquitectura implementada, la ejecución completa consiste en ejecutar el proceso de scraping en modo stream, de modo que permanezca a la escucha de comandos enviados desde el cuadro de Node-RED. A la hora que se considere oportuno, se debe enviar el comando de iniciar todos los servicios de modo secuencial.
En cada uno de los servicios de scraping se generarán los diversos datos indicados en el apartado anterior. Al iniciarse y acabarse cada uno de los servicios de scraping se generará un registro de ejecución, al obtenerse un producto se generará el respectivo registro y en caso de que se produzca un error de scraping, también se generará el registro correspondiente. Todos estos datos se almacenarán en el DataLake, donde siempre se podrán consultar en caso de que el procesamiento de datos haya ido mal. Del mismo modo, todas las trazas de log, con fecha y hora se almacenarán también en el DataLake.

Del mismo modo, cuando se inicie y se finalice cada uno de los servicios de scraping y se obtenga un producto, se generará un comando en la cola de mensajes. Además, en el caso de que se produzca un error, se enviaría una alerta de error en forma de comando al cuadro de comandos Node-RED. Este aspecto es importante ya que en un entorno de producción la pérdida de datos es inaceptable ya que puede provocar una pérdida de precisión en la inferencia de modelos IA y un aviso a tiempo permite la gestión del error minimizando la pérdida de datos.

Cuando se obtiene un producto se envía ese registro a la base de datos MongoDB alojada en el cloud Atlas, esto permite que desde el dashboard del servicio MongoDB se pueda ver el progreso de cada servicio de scraping. Y poder reaccionar a tiempo en caso de que algo vaya mal.

En el caso de que algún servicio de scraping haya ido mal, se podrá ejecutar un servicio de scraping de forma individual para subsanar la falta de datos.

3. Entrenamiento de un LoRA para la adaptación de un LLM a nuestras necesidades

Tras automatizar, formatear y insertar los datos recabados mediante scrapping en nuestra base de datos MongDB, tenemos a nuestra disposición un dataset con información útil que plantea un reto muy interesante: ¿Cómo podemos acercar estos datos a los usuarios? y en especial ¿Cómo los acercaremos a los usuarios sin conocimientos técnicos?

Con la emergente proliferación de los modelos de lenguaje natural conversacionales y su creciente popularidad, como por ejemplo ChatGpt, Google Bard, BingChat por mencionar algunos, se nos abre la posibilidad de utilizar herramientas similares como vehículo para llegar a los usuarios.

Mientras que OpenAi, creadores de ChatGpt, cada vez se vuelve menos “open” y sus modelos son cada vez menos accesibles para el público común, Meta, anteriormente conocida como Facebook, ha desarrollado y liberado para uso académico su modelo Llama[paper], que ha sido entrenado exclusivamente con datasets públicos y atestiguan que la versión de 13 billones de parámetros supera en rendimiento a GPT-3 que cuenta con 175 billones.

A raíz del lanzamiento de Llama otros proyectos Open Source han visto la luz, a nosotros nos interesan particularmente dos, Stanford Alpaca[paper], modelo desarrollado específicamente para seguir instrucciones por la Universidad de Stanford a partir Llama y Vicuna-13B[paper], también desarrollado a partir de Llama pero esta vez orientado como chatbot. Ambos modelos han utilizado técnicas de fine-tuning para extender las capacidades originales del modelo Llama.

Vamos viendo indicios de cuáles son las técnicas que actualmente utiliza la industria para desarrollar soluciones, a groso modo parten desde un modelo base y aplican alguna clase de adaptación con un nuevo dataset que especializa el comportamiento de dicho modelo. Nosotros seguiremos una aproximación similar, partiendo del modelo Vicuna-13B utilizaremos la técnica LoRA (Low-Rank Adaptation of Large Language Models)[paper] para “enseñar” la información recogida en nuestra base de datos a nuestro modelo.

Dados los recursos a los que tenemos acceso vamos a utilizar una versión del modelo Vicuna adaptado para funcionar en máquinas con pocos recursos, en nuestro caso un notebook en Colab.

Dada la intrínseca dificultad de evaluar el rendimiento de un modelo de lenguaje natural y las limitaciones de tiempo de este proyecto, vamos a centrarnos en la metodología y elección de las técnicas y herramientas utilizadas más que en los propios resultados de nuestro modelo.

3.1. Qué son los LLM (Large Language Model)

Antes de entrar en materia tenemos que tener claro que son los LLM. Las siglas hacen referencia a Large Language Model, que como su propio nombre indica son modelos desarrollados para entender y generar texto similar al humano. Estos modelos están entrenados en cantidades ingentes de textos, de ahí que lleve el sufijo “Large”. Su proceso de entrenamiento implica aprender patrones dentro de los datos, como gramática, hechos sobre el mundo e incluso capacidad de razonamiento.

Uno de los más famosos es GPT-3, desarrollado por OpenAi. GPT-3 tiene 175 billones de parámetros, lo que es varias veces superior al tamaño de previos modelos, y es que hasta ahora había una relación directa entre el número de parámetros que tenía el modelo y sus capacidades. Poco a poco han ido surgiendo modelos que optimizan el número de parámetros, empezando por Chinchilla[paper] y más recientemente LLaMA, que con un número de parámetros tremendamente inferior, aproxima sus capacidades a las expuesta por los modelos de OpenAI GPT-3.5 y GPT-4.

Pero ¿por qué el número de parámetros es relevante si ya hemos visto que no es una medida adecuada para contabilizar sus capacidades? El número de parámetros que tiene un modelo va a tener un impacto directo en la potencia computacional necesaria para su entrenamiento y su inferencia, esto quiere decir que para dos modelos entrenados sobre el mismo hardware aquel que tenga más parámetros necesitará más tiempo para entrenarse.

El entrenamiento de modelos requiere de hardware potente y costos con un gran consumo eléctrico, para hacer una ide de cifras, la versión del modelo LLaMA de 7 billones de parametros necesito de 82.000 horas de gpu para ser entrenado y su versión de 13 billones de parametros necesito 135.000 horas de gpu. Es decir, un coste económico enorme y un auténtico riesgo para el medio ambiente, ya que como hemos comentado antes, las gpus tiene un alto

consumo eléctrico y no siempre puede garantizarse que la energía utilizada provenga de fuentes renovables.

Entender este impacto es esencial para tomar decisiones responsables en relación a la contaminación y los costes, buscando una forma en la que limitar la huella de carbono. Es interesante incentivar un cambio de rumbo hacia formas más eficientes de investigar en machine learning, como por ejemplo, utilizar técnicas de fine-tuning para hacer evolucionar los modelos ya existentes, en lugar de invertir en volver a entrenar nuevos modelos cada vez más grandes[We're getting a better idea of AI's true carbon footprint], es por esto que ya se empieza a hablar de modelos fundacionales, es decir modelos que servirán para construir soluciones incrementales a partir de ellos.

Model Name	**LLaMA**	**Alpaca**	**Vicuna**
Dataset	Publicly available datasets (1T token)	Self-instruct from davinci-003 API (52K samples)	User-shared conversations (70K samples)
Training code	N/A	Available	Available
Evaluation metrics	Academic benchmark	Author evaluation	GPT-4 assessment
Training cost (7B)	82K GPU-hours	$500 (data) + $100 (training)	$140 (training)
Training cost (13B)	135K GPU-hours	N/A	$300 (training)

3.2. Que es un modelo LoRA

Ahora que ya somos conscientes del impacto que tiene el entrenamiento de modelos de gran tamaño y de la importancia de buscar soluciones más sostenibles.

El proceso de fine-tuning es una etapa crucial en el despliegue de LLMs. Mientras que los modelos poseen sólidas bases de comprensión lingüística, suelen requerir de cierta adaptación para que su rendimiento sea adecuado en tareas específicas.

Fine-tuning es el proceso de ajustar los pesos de un modelo pre-entrenado continuando su entrenamiento con dataset menor pero más específico. Normalmente el fine-tuning implica actualizar los parámetros de todo el modelo, lo que puede llegar a ser muy costoso, en computación y tiempo, en especial si el modelo tiene varios billones de parámetros.

Es en este punto en el que la técnica LoRA[codigo] ofrece una alternativa mucho más eficiente que el fine-tuning tradicional.

El proceso LoRA cuenta de 4 partes:

- Descomposición del modelo pre-entrenado
- Low-Rank Adaptation
- Reconstrucción del modelo
- Fine-tuning

3.2.1. **Low-Rank Approximation**:

La aproximación de bajo rango es una técnica que simplifica matrices complejas, reduciendo su tamaño sin pérdida significativa de información. Es útil en aprendizaje automático para comprimir grandes modelos, preservando su capacidad predictiva. LoRA emplea esta técnica para adaptar eficientemente modelos de lenguaje grandes, trabajando en una representación reducida y de bajo rango del modelo que consume menos recursos y tiempo para su adaptación. Tras ajustar el modelo de bajo rango, se reconstruye en el modelo completo, logrando mantener el rendimiento original al tiempo que se reducen los costos de adaptación.

3.2.2. **Descomposición del modelo pre-entrenado**

El proceso LoRA comienza con la descomposición del modelo de lenguaje grande ya entrenado. Para esto, se utilizan técnicas de factorización de matriz de rango bajo, como la Descomposición en Valores Singulares (SVD) o SVD truncada, que se aplican a las matrices de peso del modelo. La descomposición resulta en un conjunto de matrices más pequeñas que

forman una aproximación de rango bajo del modelo original. El propósito de este proceso es retener la información más relevante del modelo completo, pero reduciendo su tamaño y complejidad de manera significativa.

3.2.3. **Low-Rank Adaptation**

Después de descomponer el modelo previamente entrenado, se adapta su representación de rango bajo a la tarea o dominio objetivo, lo cual se logra ajustando las matrices más pequeñas obtenidas en el proceso de descomposición usando un conjunto de datos específico para la tarea. Dado que la representación de rango bajo es mucho menor que el modelo original, este proceso de adaptación es más rápido y consume menos recursos computacionales en comparación con los métodos de ajuste fino tradicionales.

3.2.4. **Reconstrucción del modelo**

Una vez finalizada la adaptación de rango bajo, se reconstruye el modelo completo al combinar las matrices de rango bajo adaptadas. Para ello, se invierte el proceso de descomposición, "reensamblando" básicamente las matrices de peso del modelo a partir de los componentes de rango bajo adaptados. El resultado es un modelo de lenguaje de tamaño completo que se ha adaptado eficientemente a la tarea objetivo, manteniendo el rendimiento del modelo original que fue previamente entrenado.

3.2.5. **Fine-tuning**

El paso final en el proceso LoRA implica una fase opcional de fine-tuning. Si bien se espera que el modelo reconstruido funcione bien en la tarea objetivo, se puede aplicar fine-tuning adicional para mejorar aún más su rendimiento. Esto implica actualizar los parámetros del modelo reconstruido en el conjunto de datos específico de la tarea, similar a los métodos tradicionales de fine-tuning. Sin embargo, como el modelo ya ha pasado por la adaptación de rango bajo, este último paso suele ser más rápido y eficiente, lo que lleva a un mejor rendimiento con costos computacionales reducidos.

3.3. Lora vs fine-tunig

Ahora exploramos rápidamente las ventajas e inconvenientes de la técnica LoRA frente al fine-tuning tradicional.

Ventajas	Desventajas
• Consumo de recursos reducido • Adaptación mas rapida • Consumo energético menor • Más accesible	• Potencial pérdida de información durante el proceso de aproximación que podría afectar el rendimiento del modelo adaptado. • La elección de las técnicas de descomposición y la selección de rango pueden influir en la efectividad de LoRA, requiriendo un ajuste y experimentación cuidadosos.

3.4. La Importancia del Open Source en los modelos LLM

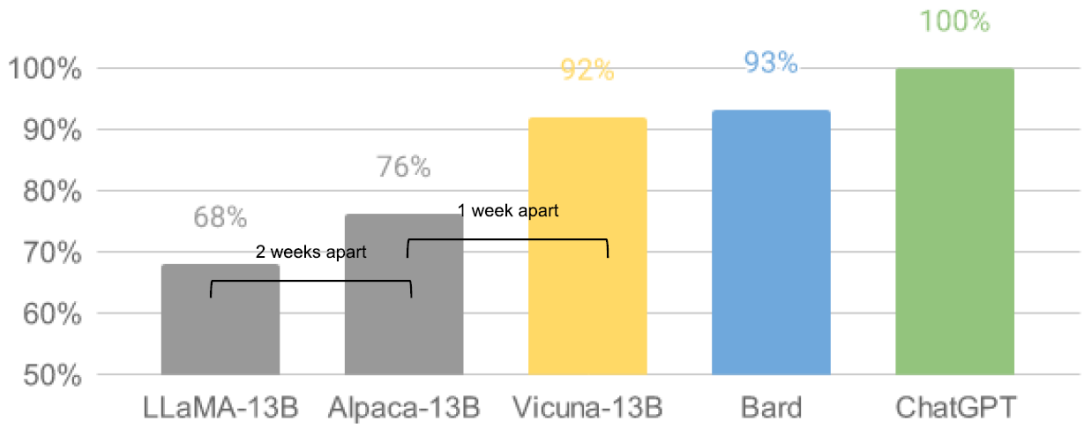
Ahora que hemos visto la importancia de seguir investigando de forma responsable y algunas de las técnicas más interesantes para poder hacer aportes incrementales sobre un modelo de lenguaje natural tiene más sentido que nunca recalcar la importancia del desarrollo Open Source.

En un reciente informe interno de google filtrado a la prensa [articulo], se discute que los grandes líderes del mercado actual pueden estar perdiendo su ventaja competitiva en el mercado de los grandes modelos de lenguaje natural.

> *"Aunque nuestros modelos siguen teniendo una ligera ventaja en términos de calidad, la brecha se está cerrando con asombrosa rapidez. Los modelos de código abierto son más rápidos, más personalizables, más privados y, libra por*

libra, más capaces. Hacen cosas con 100 dólares y 13.000 millones de parámetros que a nosotros nos cuestan 10 millones y 540.000 millones de parámetros. Y lo hacen en semanas, no en meses".

La siguiente gráfica demuestra el poco tiempo que hizo falta para lanzar los modelos alpaca y vicuna tras el lanzamiento de LLama, y aun es mas llamativo lo mucho que se acerca su rendimiento al de bard y ChatGpt por una fracción del coste y horas invertidas.



*GPT-4 grades LLM outputs. Source: https://vicuna.lmsys.org/

"Y lo que es más importante, han resuelto el problema del escalado hasta el punto de que cualquiera puede retocarlo. Muchas de las nuevas ideas proceden de gente corriente. La barrera de entrada para la formación y la experimentación se ha reducido de la producción total de una gran organización de investigación a una persona, una tarde y un portátil robusto".

En este artículo también se habla de cómo LoRA es clave para que los modelos Open Source sigan creciendo.

"Parte de lo que hace que LoRA sea tan eficaz es que, al igual que otras formas de ajuste, es apilable. Mejoras como el ajuste de las instrucciones pueden aplicarse y luego

aprovecharse a medida que otros colaboradores añaden diálogo, razonamiento o uso de herramientas. Aunque los ajustes finos individuales son de bajo rango, su suma no tiene por qué serlo, lo que permite que las actualizaciones de rango completo del modelo se acumulen con el tiempo.Esto significa que, a medida que se disponga de nuevos y mejores conjuntos de datos y tareas, el modelo podrá actualizarse a bajo coste, sin tener que pagar nunca el coste de una ejecución completa".

3.5. Modelos cuantificados a 4-bits

Dado el limitado hardware que tenemos a nuestra disposición ha sido necesario buscar las alternativas más eficientes. Ya hemos visto la técnica LoRA que nos abre un montón de posibilidades pero también podemos recurrir a versiones de los modelos que queremos utilizar cuantificadas a 4-bits.

En modo 4-bits, los modelos que son cargados solo necesitan un 25% de memoria VRAM. Por lo que modelos como LLaMA-7B podría cargarse en una GPU de 6gb, y LLaMA-30B podría cargarse en una GPU con 24gb de VRAM. La mayoría de las gráficas de consumo actuales están en esta horquilla de capacidad. Por ejemplo una AMD Radeon 5700 tiene a su disposición 8gb de ram, suficiente para cargar cómodamente LLaMA-7B, y es una gráfica que salió al mercado hace 4 años. Y aún más llamativo es que las gráficas disponibles en el tier gratuito de Google Colab son tesla T4 que cuentan con 15gb de memoria VRAM, suficiente para cargar el modelo LLaMA-13B.

Esta optimización es posible gracias a que se adapta el modelo para que represente números utilizando 4-bits en lugar de los tradicionales 32 o 64 bits. Este proceso tiene un precio, al reducir la cantidad de bits para representar números también reducimos la precisión con la que podemos representarlos. En muchos casos no es necesaria tanta precisión pero es cierto que puede conllevar una pérdida en la calidad del modelo.

En nuestro caso hemos utilizado un modelo cuantificado con GPTQ [paper] a 4-bits.

3.6. Modelo de IA base Alpaca 4 bits/ Vicuna 4 bits

Ahora tan solo nos queda elegir el modelo más adecuado para nuestra tarea. De los modelos Open Source que hemos visto, los dos que tienen el mejor rendimiento son Vicuna y Stanford Alpaca. Ambos tienen características muy interesantes, pero para tomar una decisión adecuada tenemos que fijarnos en el tipo de dataset con el que fueron entrenados.

- Stanford Alpaca se entrenó utilizando un conjunto de datos de 52 mil ejemplos que siguen instrucciones. Este conjunto de datos se generó utilizando las técnicas descritas en el artículo Self-Instruct, con algunas modificaciones. Cada ejemplo en el conjunto de datos consta de una instrucción, un contexto de entrada opcional y una salida generada por el modelo 'text-davinci-003' de OpenAi.
- Para entrenar a Vicuna, el equipo de investigación recopiló alrededor de 70,000 conversaciones de ShareGPT.com y mejoró los scripts de entrenamiento proporcionados por Alpaca para manejar mejor las conversaciones de múltiples rondas y las secuencias largas.

Ambos modelos fueron entrenados a partir del modelo LLaMA de Meta, en nuestro caso tenemos intención de crear un chatbot, Vicuna ha sido especialmente diseñado para esta tarea por lo que será el más adecuado. Si lo que hubiésemos querido fuese un resultado más basado en instrucciones, Alpaca habría sido la elección más acertada.

3.7. Creando nuestro propio LoRa

Ahora que ya tenemos casi todas las piezas de nuestro puzle estamos en situación de poder entrenar nuestro modelo. Para hacer el desarrollo, pruebas y la iteración más amena hemos recurrido a una interfaz visual Open Source [text-generation-webui]. Esta interfaz visual está pensada para lanzarse en ordenadores de escritorio, por lo que para poder utilizarla en Colab ha habido que hacerle algunas adaptaciones. También fue necesario recurrir a algunos workaround, el proyecto aún está en etapas muy tempranas y no es todo lo estable que nos gustaría.

También fue necesario añadir dependencias adicionales y algunas adaptaciones para permitirle trabajar con modelos en modo de 4-bits. Dado que todos los scripts y dependencias son muy nuevas aún no están

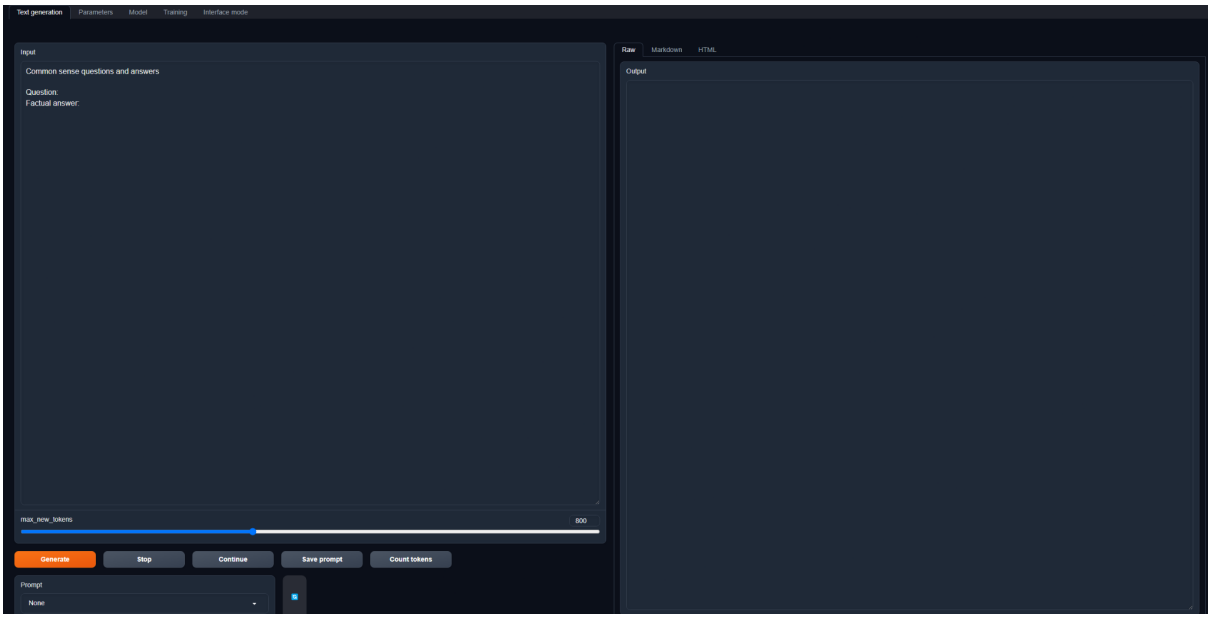
disponibles a través de pip, por lo que ha sido necesario instalarlas con ayuda de git.

!pip install git+https://github.com/sterlind/GPTQ-for-LLaMa.git@lora_4bit

Esta sintaxis permite instalar un repositorio como dependencia y cargar todos sus scripts.

Utilizamos una versión cuantificada a 4-bits de Vicuna-13b disponible en HugginFace.

Text generation webui hace uso de gradio para crear un túnel para la máquina en la que se ejecuta. Funciona de forma similar a una VPN, esto permite que sirva un link a un servidor http alojado en la máquina que ejecuta el colab, este servicio http expone la interfaz que hemos utilizado para entrenar nuestro modelo.



Tenemos 4 pestaña:

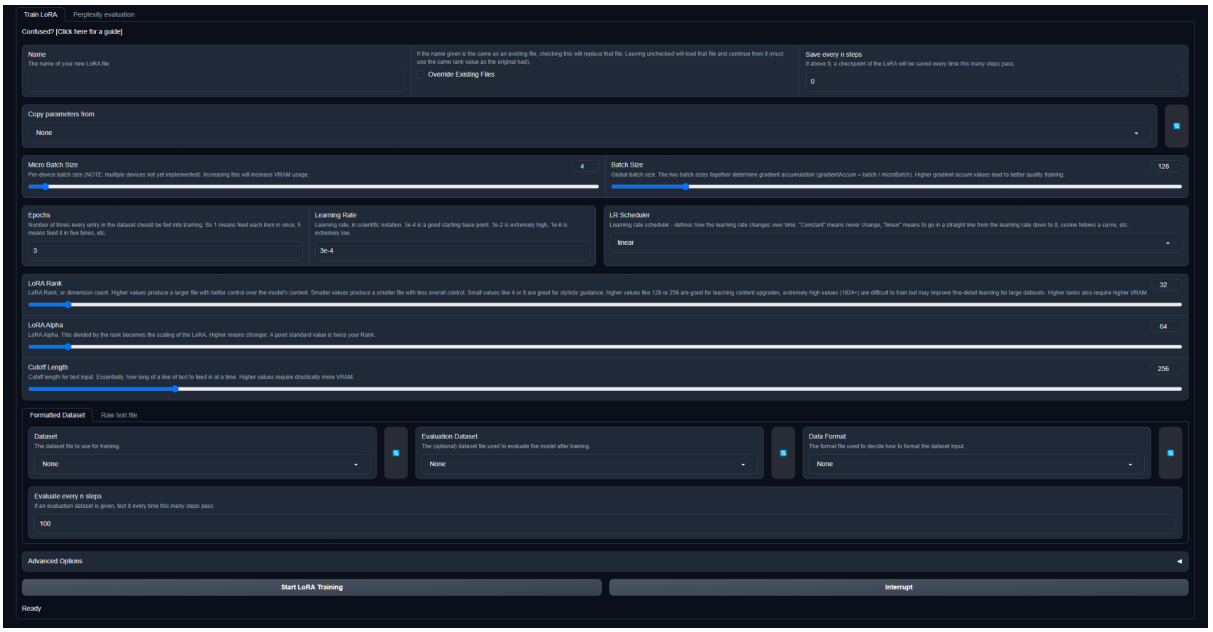
- **Generation:** Esta pestaña está dedicada a la inferencia. En el panel de la izquierda podremos escribir nuestro promt, el panel de la derecha mostrará el resultado de la generación, también hay un selector que nos permitirá ver algunos promts “precocinados”.
- **Parameters:** En esta pestaña podemos ajustar los parámetros de generación, tales como la temperatura, el top p, el top k. Son parámetros muy similares a los disponibles en los modelos de OpenAi,

lo que nos lleva a pensar que son estándares. Probablemente los dos parámetros que más nos interesen serán la temperatura, algo así como lo imaginativo o creativo que queremos que sea el resultado de la generación, y el repetition_penalty, este valor penaliza las repeticiones de tokens previamente generados.

- **Model:** En esta pestaña podemos elegir el modelo que queremos usar y aplicarle algún LoRA, tendremos que tener los modelos en local, por lo que también permite descargar nuevos modelos de hugginface. Antes de empezar el entrenamiento deberemos asegurarnos de que no hay ningún LoRA aplicado, porque influirá en los resultados obtenidos.
- **Training:** Aquí es donde vamos a dedicar más tiempo, en esta pestaña podemos ajustar los parámetros de entrenamiento. Los más interesantes van a ser:
    - MicroBatch, cuanto mas alto, mas rapido ira el entrenamiento pero más VRAM consumirá. Nos interesa dejarlo a 1, a nuestro entorno no le sobra la memoria.
    - Batch Size, este valor junto con el MicroBatch impactará tanto en la velocidad como en la calidad del entrenamiento. Cuanto más alto esté, más rápido y mejor aprenderá el modelo, pero consumirá más VRAM. En las pruebas que hemos hecho hemos conseguido lanzar el entrenamiento con un valor de 128, lo que nos ofrece un buen punto de partida en relación a calidad y velocidad.
    - Epochs, esto funciona como con cualquier otro modelo. Más epochs tiende a tener mejores resultados pero lleva más tiempo, hay que tener cuidado con el overfitting. Este valor no lo hemos subido mucho ya que este es uno de los valores que más impactan en los tiempos de entrenamiento, con el tiempo limitado que hemos tenido para realizar el proyecto era más interesante experimentar con otros parámetros y dejar este para el final.
    - Learning rate: Este valor funciona igual que en los modelos tradicionales. Hemos utilizado un learning rate de $3e^{-4}$
    - LoRA Rank: Este es uno de los parámetros más importantes, esto afectó al punto en el que empieza a actuar LoRA. Un valor más pequeño resultará en un fichero más pequeño pero también en una adaptación más ligera, valores bajos serán útiles para simplemente darle un poco de estilo al modelo. Valores más altos incurrirá en un fichero mayor y una adaptación más profunda, esto nos será especialmente útil para hacer que el modelo “aprenda” el contenido proporcionado. Este valor también afecta

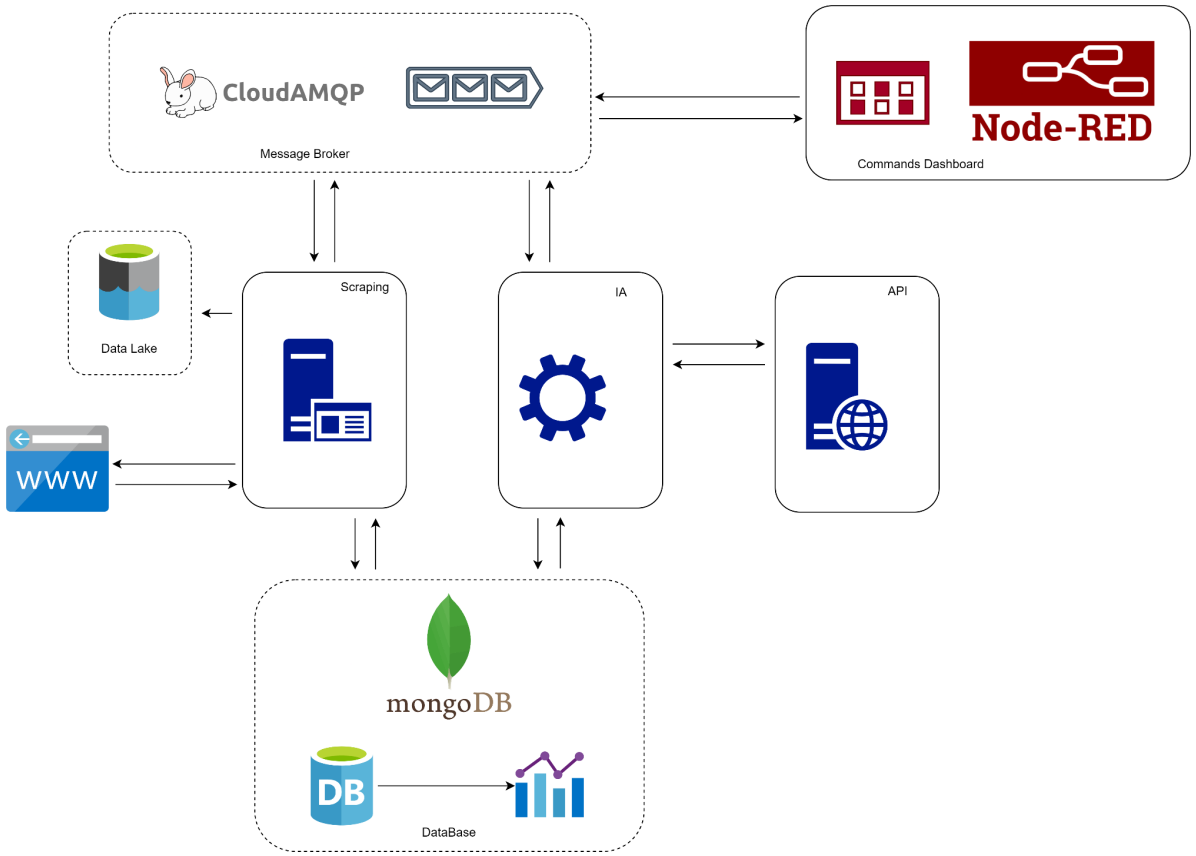
al consumo de VRAM. En nuestro caso hemos utilizado un valor de 256, con más VRAM podríamos subir más este valor para mejorar la precisión de nuestro modelo.

- Dataset: En nuestro caso vamos a usar un dataset que será un json servido en un fichero txt, el modelo es capaz de interpretar json y extraer información de él. Otra opción hubiese sido proveer un fichero json con una estructura determinada, parecida al que se usó para entrenar a Alpaca, esto sería un dataset más parecido a los que se suelen utilizar en otro tipo de modelos, si hubiésemos utilizado Alpaca como modelo base y hubiésemos querido un resultado más instruccional este formato hubiese sido el más adecuado.



Para nuestro modelo hemos utilizado un dataset con 15 000 productos de alcampo. Con los parámetros que hemos estado utilizando, el tiempo de entrenamiento rondaba las 5 horas. En cuanto a los resultados, es difícil valorarlos, aunque es cierto que empezábamos a ver cierta consistencia en las generaciones. Seguimos teniendo muchos valores que no se ceñían a nuestros datos de entrenamiento, pero es suficiente para validar esta línea de investigación. Con más tiempo y un hardware un poco más potente podríamos haber conseguido muy buenos resultados.

3.8. Cómo servir el modelo al usuario



Ahora que ya hemos visto cómo entrenar nuestro modelo ya solo nos queda ponerlo a disposición de nuestros usuarios. Como este tema quedaba fuera del alcance de nuestro proyecto haremos una propuesta a nivel meramente teórico.

Podemos ofrecer acceso a nuestro modelo mediante una API hecho con Flask o con FastApi. Entrenaríamos nuestro modelo una vez al día con los datos recuperados del scraping de ese mismo día y serviríamos el modelo actualizado a través de un endpoint para la inferencia.

Una vez disponible la inferencia otras integraciones serían posibles e interesantes de explorar, como por ejemplo, integración con Whatsapp o con Telegram.

4. Conclusión

Y con esto concluimos nuestro proyecto integrador para el curso de especialización en inteligencia artificial y big data. Ha sido una oportunidad

maravillosa para aunar y desarrollar todos los conocimientos y técnicas adquiridas a lo largo de este año a la par que también nos ha servido de excusa para investigar por nuestra cuenta el estado actual de la industria y enfrentarnos a la clase de desafíos que nos encontraremos en el mercado laboral.

5. Bibliografía

https://github.com/mariocarbonell/super_smart.git

https://www.semianalysis.com/p/google-we-have-no-moat-and-neither

https://www.xataka.com/robotica-e-ia/open-source-nos-puede-eclipsar-google-cree-que-ella-openai-estan-perdiendo-carrera-ia

https://lmsys.org/blog/2023-03-30-vicuna/

https://devinschumacher.hashnode.dev/lora-low-rank-adaptation-of-large-language-models

https://lightning.ai/pages/community/tutorial/lora-llm/

https://research.facebook.com/file/1574548786327032/LLaMA--Open-and-Efficient-Foundation-Language-Models.pdf

https://crfm.stanford.edu/2023/03/13/alpaca.html

https://www.deepmind.com/publications/an-empirical-analysis-of-compute-optimal-large-language-model-training

https://www.technologyreview.com/2022/11/14/1063192/were-getting-a-better-idea-of-ais-true-carbon-footprint/