UNIVERSIDAD NACIONAL DE EDUCACIÓN A DISTANCIA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

Trabajo Fin de Máster del
Máster Universitario en Ingeniería y Ciencia de Datos

# ***Large Language Models (LLMs)* para calidad y estandarización de datos**

ALBERT HERRANDO MORAIRA

Dirigido por: RAQUEL MARTINEZ UNANUE

M. LOURDES ARAUJO SERNA

Curso: 2023-2024: 1ª Convocatoria

# Agradecimientos

Quisiera expresar mi agradecimiento a la Universidad Nacional de Educación a Distancia (UNED), especialmente al departamento de Lenguajes y Sistemas Informáticos de la Escuela Técnica Superior de Ingeniería Informática, por darme la oportunidad y los recursos necesarios para realizar esta investigación.

Un agradecimiento especial a mis tutoras, la Dra. Raquel Martinez Unanue y la Dra. M. Lourdes Araujo Serna, que han aceptado mi trabajo y me han guiado en la concepción y el desarrollo del proyecto.

También quiero mencionar el inmenso apoyo de mi familia, que me ha ayudado en los momentos más difíciles. Gracias a mi pareja, Mimi, y a mi hija, Lia, que ha nacido durante la realización de este trabajo. También a mis padres y a mi hermana por su apoyo incondicional.

Finalmente, expresar mi gratitud a mis amigos y compañeros de trabajo, quienes me han animado y dado consejos a lo largo de todo este proyecto.

# Resumen

Actualmente, uno de los principales problemas de las empresas, organizaciones o gobiernos que gestionan datos es la calidad de los mismos. Éstos suelen contener una gran cantidad de errores con multitud de variaciones. En consecuencia, resulta difícil corregirlos para realizar análisis o predicciones. Por otro lado, en los últimos años se han desarrollado grandes modelos del lenguaje con potencial para resolver este tipo de tareas de calidad de los datos. Este proyecto explora la viabilidad de la utilización de grandes modelos del lenguaje para la corrección y la estandarización de datos. Primero, se ha investigado el estado del arte de los modelos y de la calidad de los datos. Seguidamente, se ha diseñado una metodología para realizar experimentos con grandes modelos del lenguaje y tareas de calidad. En particular, se han llevado a cabo tres experimentos. El primero sobre corrección de valores, el segundo sobre estandarización de atributos, y el tercero sobre imputación de valores ausentes. Después de evaluar los experimentos, se ha observado que la aplicación de LLMs en tareas de calidad del dato puede obtener buenos resultados en determinados escenarios muy concretos. Por ejemplo, cuando la cantidad de datos a corregir es pequeña y la tarea es sencilla. Sin embargo, para grandes cantidades de datos o tareas complejas, han surgido dificultades relacionadas con los tiempos de ejecución, los costes económicos y la fiablidad de las respuestas.

**Palabras clave:** LLMs, Calidad del dato, Ingeniería de *prompts*, Corrección y estandarización de datos

# Índice general

# Índice de tablas

# Índice de figuras

# Nomenclatura

ADAM Adaptive Moment Estimation, página 16

AI Artificial Intelligence, página 5

API Application Programming Interface, página 10

ART Automatic Reasoning and Tool Use, página 30

BERT Bidirectional Encoder Representations from Transformers, página 7

CoT Chain-of-Thought, página 26

DAMA Data Management Association, página 32

DL Deep Learning, página 6

FLOPS FLoating point number Operations Per Second, página 16

GNMT Google Neural Machine Translation, página 7

GPT Generative Pre-trained Transformer, página 7

GPU Graphical Processing Unit, página 7

GRU Gated Recurrent Unit, página 6

ICL In-Context Learning, página 25

LLaMA Large Language Model Meta AI, página 10

LLM Large Language Model, página 2

LM Language Modeling, página 5

LSTM Long Short-Term Memory, página 6

MDM Master Data Management, página 37

MoE Mixture of Experts, página 11

NLM Neural Language Models, página 6

NLP Natural Language Processing, página 2

OLTP OnLine Transaction Processing, página 35

PAL Program-Aided Language Model, página 29

PE Positional Encoding, página 15

PII Personal Identifiable Information, página 13

PLM Pre-trained Language Models, página 7

RAG Retrieval-Augmented Generation, página 18

RLHF Reinforcement learning from human feedback, página 9

RNN Recurrent Neural Networks, página 6

SFT Supervised Fine-Tuning, página 17

SGD Stochastic Gradient Descent, página 16

SLM Statistical Language Models, página 6

ToT Tree-of-Thought, página 27

TPU Tensor Processing Unit, página 16

# Capítulo 1

# Introducción

Este primer capítulo introduce el proyecto incluyendo el contexto de la investigación, la motivación que impulsa este trabajo, los objetivos que se quieren lograr, el alcance que se pretende abarcar y la estructura de la memoria que se ha desarrollado.

## 1.1. Contexto

Actualmente, las grandes empresas, organizaciones gubernamentales y administraciones públicas tienen multitud de sistemas de información para dar servicio a sus clientes y a los ciudadanos. Sin embargo, en muchas ocasiones, los datos que se generan y se gestionan carecen de calidad. Existen diferentes razones de la mala calidad del dato, por ejemplo, que los sistemas sean distintos, que haya diferentes modelados de datos o que se realice un mal procesamiento de los mismos.

Para dar determinados servicios o tomar decisiones de negocio objetivas se requiere hacer uso de estos datos. Así pues, al utilizarlos, se encuentra que no están estandarizados y que contienen un gran número de problemas e inconsistencias. Este hecho imposibilita su uso y el cruzamiento entre diferentes sistemas de información para realizar análisis o predicciones.

Con el objetivo de estandarizar la información, las organizaciones definen diferentes tipos de datos. Primero, los datos transaccionales son aquellos que residen en los sistemas de información y suelen tener problemas de calidad. Por lo tanto, es necesario corregirlos siguiendo unos determinados estándares. Por otra parte, se definen los datos maestros y de referencia, que sirven de modelo estándar para garantizar una buena integración entre sistemas.

Así pues, la tarea acaba siendo la corrección y estandarización de datos transaccionales que residen en los sistemas de información, basándose en los datos maestros y de referencia. Sin embargo, es muy difícil encontrar algoritmos generalistas que sean capaces de automatizar la corrección de todos los posibles problemas y alteraciones que sufren los datos. La gran variabilidad de errores de los datos transaccionales provoca que esta tarea sea muy compleja.

Por otro lado, en los últimos años se han desarrollado modelos del lenguaje, llamados *grandes*

*modelos del lenguaje* o *Large Language Models (LLM)*, que permiten entender y generar lenguaje natural de manera general. Estas capacidades presentan el potencial de ser aplicadas en el ámbito de la calidad de los datos, concretamente en la corrección y estandarización de datos transaccionales erróneos o de mala calidad.

En vista de este potencial, este trabajo pretende explorar la viabilidad de aplicar este tipo de algoritmos de redes neuronales profundas para tareas de mejora de la calidad del dato.

## 1.2. Motivación

La motivación de esta investigación radica en la necesidad de mejorar la calidad de los datos de grandes empresas, organizaciones y gobiernos. En particular, se busca incrementar la calidad de los datos transaccionales que residen en sistemas de información. De esta manera, se podrán utilizar para tomar decisiones, dar mejores servicios o realizar análisis más exhaustivos.

Además, se espera que los LLM presenten capacidades lo suficientemente generalistas de procesamiento del lenguaje natural *(Natural Language Processing (NLP))* para hacer frente a la gran variedad de posibles errores que contienen los datos. Por ejemplo, errores de tipografía, de puntuación, de símbolos o de abreviaciones que se generan al momento de la creación de los datos.

En caso de confirmar su viabilidad, se podrían llegar a desarrollar aplicaciones basadas en LLM para corregir y estandarizar datos erróneos de diferentes sistemas.

## 1.3. Objetivos

El objetivo general de este trabajo es comprobar la viabilidad de utilizar LLMs para corregir la mala calidad de los datos transaccionales. La intención es comprobar si los LLMs son suficientemente generalistas e inteligentes para corregir los datos de sistemas de información basándose en datos maestros o de referencia. De esta manera, los datos transaccionales se podrán usar para futuros cruces de información, análisis o implementaciones de modelos de inteligencia artificial.

Para lograrlo, se utilizaran datos maestros y de referencia reales provenientes del portal de datos abiertos de la *Generalitat de Catalunya*. A partir de estos datos, se crearán datos transaccionales ficticios que contendrán una gran cantidad de errores e inconsistencias.

La idea es proporcionar datos de referencia y datos transaccionales a los modelos LLM para verificar si éstos son capaces de corregirlos y estandarizarlos. A su vez, se evaluará la efectividad de las instrucciones que se proporcionan a los LLMs, el tiempo que tardan en ejecutar la tarea y la exactitud con la cual la realizan.

Así pues, para alcanzar el objetivo general se han plantado los siguientes sub-objetivos más concretos:

- Realizar un estudio del estado del arte de los LLMs, de la calidad de los datos y de las potenciales aplicaciones que pueden tener los LLMs para tareas de calidad del dato.
- Diseñar una metodología que permita realizar experimentos de calidad del dato con LLMs, desde la importación de datos hasta la evaluación de los resultados. Además debe ser suficientemente flexible como para utilizar diferentes modelos LLM y compararlos entre ellos.
- Realizar tres experimentos que permitan evaluar la viabilidad del uso de LLMs para:
    1. Corregir valores, en formato texto, de una variable de un conjunto de datos transaccionales, basándose en datos de referencia.
    2. Estandarizar datos transaccionales que no tengan la estructura adecuada, es decir, que no tengan los atributos que marcan los datos de referencia.
    3. Imputar valores ausentes, en formato texto, de datos transaccionales, basándose en datos maestros.
- Analizar los resultados y obtener conclusiones sobre el uso de los LLMs en estas tareas.

De este modo, se conseguirá el objetivo de comprobar si esta tecnología es viable para desarrollar futuras aplicaciones para mejorar la calidad del dato y si es escalable en un entorno real.

## 1.4. Alcance

Teniendo en cuenta el desarrollo aún preliminar de estas tecnologías, esta investigación no pretende desarrollar una aplicación de principio a fin que esté preparada para poner en producción, ni tampoco un producto acabado para lanzar al mercado. Sino que se pretende evaluar el potencial de los modelos y determinar si sería viable una implementación más completa en un futuro.

El alcance de este trabajo, incluye principalmente dos partes: una investigación teórica y otra parte práctica. En la primera, se presentará información sobre el estado del arte actual sobre los modelos, la calidad del dato y algunas investigaciones que han intentado aplicar LLMs para tareas relacionadas. En la segunda parte, se expondrá una metodología para realizar experimentos prácticos que muestren el potencial de los modelos. Además, se realizan tres experimentos con diferentes tareas de calidad del dato para comprobar si los LLMs son capaces de resolverlos. Finalmente, se presentarán los resultados obtenidos junto con un análisis y se expondrán las conclusiones y los potenciales trabajos futuros.

## 1.5. Estructura de la memoria

La estructura de la memoria de este proyecto se organiza en distintos capítulos, cada uno dedicado a exponer las diferentes fases de la investigación. A continuación, se ofrece un resumen de lo que

aborda cada capítulo:

- **Capítulo 1. Introducción:** Presenta una introducción que incluye el contexto de la investigación, la motivación que impulsa este trabajo, los objetivos que se quieren lograr, el alcance que se pretende abarcar y la estructura de la memoria que se ha desarrollado.
- **Capítulo 2. Estado del arte:** Expone el resultado de la investigación sobre el estado actual de los LLMs, la calidad del dato y algunos estudios que los relacionan.
- **Capítulo 3. Metodología:** Muestra el entorno de desarrollo utilizado, junto a las fuentes de datos, librerías, APIs y costes del proyecto.
- **Capítulo 4. Experimentos:** Propone tres experimentos para validar la viabilidad del uso de LLMs en diferentes tareas de calidad del dato.
- **Capítulo 5. Modelos LLM:** Describe los modelos LLM utilizados para realizar el estudio junto a sus características más importantes.
- **Capítulo 6. Evaluación:** Presenta las métricas de evaluación que se han utilizado para evaluar el desempeño de los LLMs en los experimentos planteados.
- **Capítulo 7. Resultados:** Revela los resultados obtenidos de los experimentos. Además, interpreta los resultados obtenidos analizando y comentando las implicaciones prácticas y las limitaciones de la tecnología.
- **Capítulo 8. Conclusiones y trabajos futuros:** Resume los principales hallazgos de esta investigación y ofrece direcciones para futuros trabajos en la misma línea.

# Capítulo 2

# Estado del arte

En este capítulo se presenta el estado del arte de las áreas de interés de esta investigación. La sección 2.1 presenta los grandes modelos del lenguaje, la sección 2.2 introduce la ingeniería de *prompts*, la sección 2.3 describe los temas relacionados con la calidad del dato, y la sección 2.4 profundiza en la aplicación de los grandes modelos del lenguaje en tareas de calidad del dato.

## 2.1. Large language models (LLMs)

El *lenguaje natural* es un sistema de comunicación estructurado propio de una comunidad humana que cuenta generalmente con escritura. Su función es compartir información y expresar pensamientos, emociones y opiniones. Gracias a su complejidad y riqueza, los humanos han sido capaces de desarrollar elevados niveles de inteligencia.

Desde la llegada de la computación y las máquinas, se ha intentado que los ordenadores entiendan nuestro lenguaje para llevar a cabo determinadas tareas. Sin embargo, desarrollar un sistema o algoritmo que comprenda las complejidades del lenguaje natural siempre ha sido un reto durante las últimas décadas.

El *modelado del lenguaje natural (Language Modeling (LM))* es el campo de la lingüística computacional y de la inteligencia artificial *(AI)* que pretende desarrollar modelos que entiendan y generen lenguaje humano. Durante las últimas décadas, ha evolucionado enormemente, desde los modelos del lenguaje estadísticos hasta los actuales grandes modelos neuronales (Zhao et al., 2023).

Los *grandes modelos del lenguaje* o *Large Language Models (LLM)* son algoritmos de redes neuronales artificiales con una arquitectura que les permite modelar el lenguaje de manera general. Típicamente, se entrenan con masivas cantidades de texto y aprenden de los patrones y estructuras del lenguaje (Hadi et al., 2023). Como resultado, tienen la capacidad de comprender, interpretar y generar lenguaje natural. Además se ha demostrado que poseen, hasta cierto punto, habilidades adicionales, llamadas habilidades emergentes, como la capacidad de razonamiento o comprensión del contexto (OpenAI, 2023a).

En esta sección se presenta la historia de los LLM y su situación en la actualidad, incluyendo los datos de entrenamiento utilizados, los tipos de modelos, los entrenamientos, ajustes a tareas posteriores, su puesta en producción e interfaces de usuario, la evaluación y los retos que presentan.

### 2.1.1. Historia

La historia del modelado del lenguaje natural se puede dividir en diferentes etapas según los descubrimientos realizados. A continuación, se explican las fases de acuerdo con Zhao et al. (2023) y Hadi et al. (2023).

Los primeros modelos del lenguaje fueron desarrollados en las décadas de 1950 y 1960. Se basaban en reglas y dependían de pautas de lenguaje y gramática programadas manualmente. Estos primeros modelos tenían muy pocas capacidades y les costaba interpretar la complejidad y variabilidad del lenguaje natural.

En los 1980s y los 1990s, se empezaron a desarrollar modelos estadísticos del lenguaje *(Statistical Language Models (SLM))*, que usaban métodos probabilísticos para estimar la probabilidad de una secuencia de palabras dado un determinado contexto. Gracias a los nuevos métodos de computación, pudieron procesar mayor cantidad de texto y obtener mejores resultados que los anteriores modelos basados en reglas. Por otro lado, la rama del procesamiento del lenguaje natural *(Natural Language Processing (NLP))* fue ganando interés científico. Sin embargo, todavía tenían muchas limitaciones, por ejemplo la baja dimensionalidad que podían procesar. Además, carecían de capacidades para entender el contexto y la semántica de los textos.

Durante la década de los 2010s se produjo un descubrimiento importante, el desarrollo de modelos del lenguaje neuronales *(Neural Language Models (NLM))* que usaban aprendizaje profundo *(Deep Learning (DL))* para modelar patrones y estructuras de grandes cantidades de texto. Se basaban en caracterizar la probabilidad de secuencias de palabras a través de redes neuronales. Así, pudieron generar textos más naturales que sus modelos antecesores. Los modelos más destacados fueron las redes neuronales recurrentes *(Recurrent Neural Networks (RNN))*, las cuales tenían una estructura de bucle que permitía la persistencia de información secuencial. Sin embargo, presentaban un gran problema, el desvanecimiento del gradiente, lo que les hacía olvidar la información inicial en secuencias largas. Debido a este inconveniente, se desarrollaron las memorias de corto y largo plazo *(Long Short-Term Memory (LSTM))*, que pretendían tener mayor control sobre el flujo de información a través de puertas de entrada, olvido y salida de información y celdas de estado. Además, se desarrollaron las unidades recurrentes de puertas *(Gated Recurrent Unit (GRU))*, las cuales eran una simplificación de las LSTM para hacerlas más eficientes en cuanto a computación.

Seguidamente, en 2013, se presentó y se utilizó el concepto de las representaciones distribucionales como *word2vec*, que proponía el aprendizaje distribuido de representaciones de palabras utilizando redes neuronales superficiales. Este paso demostró una mejora en la efectividad de tareas de NLP.

En 2014, Bahdanau y Cho (2014) publicaron los llamados mecanismos de atención. Un concepto

revolucionario, que dejaría de tratar los textos como elementos secuenciales y daría a cada palabra una relación con todas los demás. Mediante redes neuronales, a cada palabra se le asignaba un vector llave (que definía la palabra) y un vector consulta (que presentaba las relaciones que buscaba esa palabra). Seguidamente, se calculaba el vector de atención para cada palabra, que era el producto escalar entre su vector consulta y todos los otros vectores llave del resto de palabras. Repitiendo este proceso para cada palabra se obtenía la matriz de atención, que representaba en qué se focalizaba el modelo cuando recibía una determinada palabra como entrada. Seguidamente se utilizaban los vectores de atención para realizar una suma ponderada de los vectores valores de las palabras. El resultado fue un mecanismo que permitía relacionar las palabras sin importar lo alejadas que estén en el texto y que incorporaba el contexto de las frases en los modelos. Esto supuso la solución a uno de los principales problemas de las RNN, LSTM y GRU, la falta de memoria. En los próximos años, se trabajó en incorporar mecanismos de atención a RNN, LSTM y GRU obteniendo así mejores resultados en textos largos.

Otro paso importante, fue en 2015, cuando Google presentó un modelo del lenguaje neuronal de traducción llamado *Google Neural Machine Translation (GNMT)* . La novedad fue que se entrenó a gran escala con grandes cantidades de texto bilingüe y fue capaz de alcanzar el rendimiento de los traductores convencionales de la época.

En 2017, sucedió el hito de la publicación del modelo *"Transformer"*, en el famoso artículo *Attention is all you need*, de Vaswani et al. (2017). Se presentó una arquitectura que revolucionaría los modelos del lenguaje neuronales y se demostró que solamente mediante mecanismos de atención, se podrían superar los modelos anteriores. Particularmente, se codificaba la posición de cada palabra en el texto mediante un vector cuyas componentes eran ondas sinusoidales de diferentes frecuencias. Además, se codificaba todo el texto a la vez, lo que hizo a esta arquitectura altamente paralelizable y permitió el entrenamiento masivo mediante GPUs.

Un año más tarde, en 2018, se presentaron los modelos pre-entrenados basados en *transformers (Pre-trained Language Models (PLM))*, una revolución que sentaría las bases de los LLMs. Se propusieron modelos que capturaban representaciones de palabras teniendo en cuenta su contexto. Además, al utilizar la arquitectura de los *transformers*, los modelos pudieron ser paralelizables y escalar con el poder de computación de las GPUs.

También, en junio de 2018, OpenAI lanzó GPT-1 *(Generative Pre-trained Transformer (GPT))*, un modelo basado en *transformers* con 117 millones de parámetros. GPT-1 marcó un avance significativo ya que pudo resolver diferentes tareas de NLP y generar texto basado en contexto. La metodología de entrenamiento de GPT-1 asentó unos principios que después seguirían modelos posteriores. Se entrenó en un proceso con dos fases, la primera con un proceso de pre-entrenamiento no supervisado y la segunda con un ajuste fino supervisado.

En 2019, se publicó BERT *(Bidirectional Encoder Representations from Transformers)*, un modelo con 330 millones de parámetros que, a diferencia de los modelos anteriores que analizaban el texto de manera unidireccional, era capaz de analizar el contexto de las palabras en ambas direcciones

simultáneamente. Esto le hizo comprender mejor el significado del texto. Además, asentó las bases del entrenamiento actual, con una fase de pre-entrenamiento y otra de ajuste fino. A partir de este paradigma, en febrero de 2019 también surgieron mejoras en la arquitectura, como con GPT-2 de OpenAI, un modelo de 1.500 millones de parámetros. Llegados a este punto, se investigó el efecto que tenía el escalado tanto de los tamaños de los modelos como de los conjuntos de datos, pensando que aumentando su tamaño se incrementarían las capacidades de los modelos del lenguaje.

En junio de 2020, OpenAI publicó GPT-3, su tercera versión de modelo GPT con 175.000 millones de parámetros. Este tamaño y escalas masivas le permitieron realizar tareas de NLP con una sofisticación sin precedentes. Además, GPT-3 desarrolló la capacidad del aprendizaje en contexto (*in-context learning*), es decir, aprender de los ejemplos provistos en los textos de entrada y adaptarse a ellos aunque no estuvieran en el conjunto de datos de entrenamiento. También demostró habilidades de generalización a partir del conocimiento preexistente. Sin embargo, también presentaba limitaciones, como la imprecisión o las alucinaciones, es decir, dar información errónea en la generación de texto.

A partir de este momento, la década de los 2020s esta siendo la época dorada de los LLMs. En sólo tres años, desde 2020 a 2023 se han publicado multitud de modelos, tanto en la industria, como en la investigación. La Figura 2.1 muestra una línea temporal de los lanzamientos de grandes modelos del lenguaje.

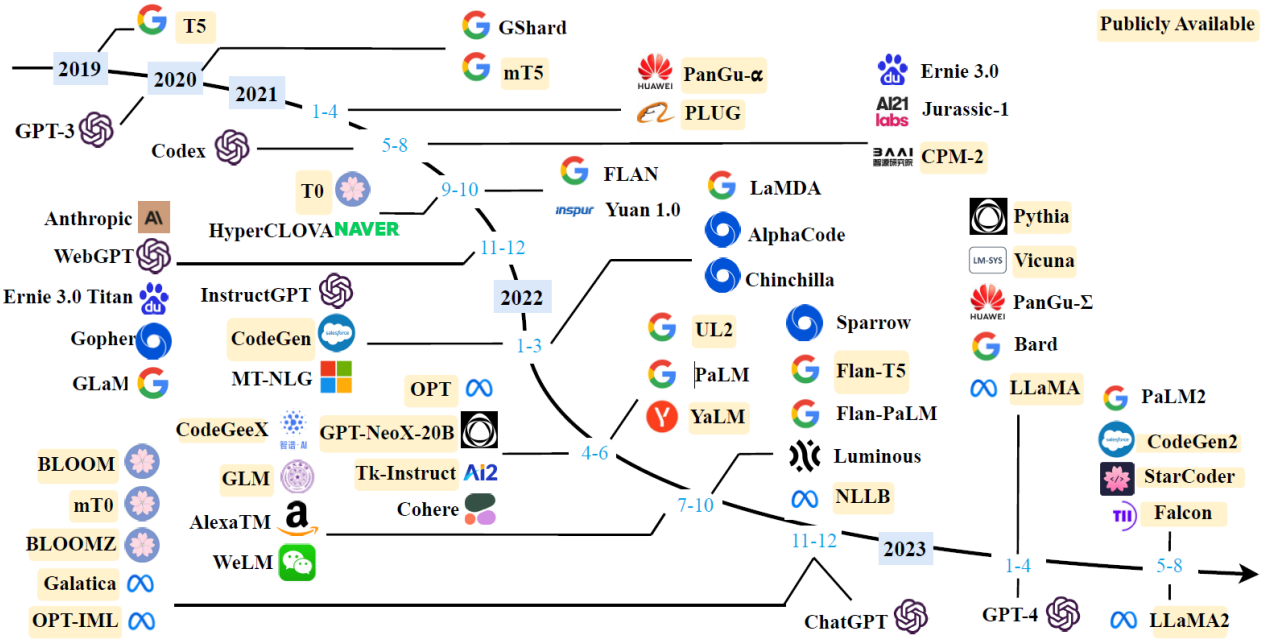


Figura 2.1: Línea temporal de lanzamientos de LLMs desde 2020 hasta 2023. Tomado de Zhao et al. (2023).

Entre 2020 y 2021, con el éxito de GPT-3, no sólo el mundo de la investigación científica se centro en los LLMs, sino que empresas privadas de la industria entraron en el desarrollo de modelos. Por ejemplo, Google presentó modelos como FLAN, GLaM y LaMDA, y surgieron pequeñas

empresas especializadas en la creación de LLMs, por ejemplo Antrhopic. Además, el desarrollo se internacionalizó, llegando a China con empresas como Huawei, Alibaba o Baidu y a Corea del Sur con Naver.

Respecto a OpenAI, en 2021 lanzó el modelo Codex, una evolución de GPT-3 diseñado y entrenado específicamente para entender y generar código en varios lenguajes de programación. Por primera vez, un modelo del lenguaje podía programar a partir de descripciones en lenguaje natural y realizar auto-completados inteligentes de código.

Por otro lado, la comunidad de código abierto (*open-source*) no se quedó de brazos cruzados. En 2021, la empresa Hugging Face, Inc. recaudó 40 millones de dólares en una ronda de Serie B para trabajar en una plataforma de compartición de modelos y conjuntos de datos, además de trabajar en el desarrollo de LLMs de código abierto con diferentes investigadores.

El año 2022 fue revolucionario para el sector de los LLMs, llevándolos de ser un tema de investigación de nicho a demostrar sus capacidades al gran público en aplicaciones reales. A principios de año, en marzo, OpenAI publicó una mejora de su modelo, llamada GPT-3.5, con una arquitectura mejorada y optimizada (Ray, 2023). Por otro lado, hasta ese momento, los LLMs eran capaces de completar texto dado un texto de entrada. Sin embargo, tenían limitaciones para seguir instrucciones y responder a solicitudes de usuarios humanos de manera efectiva y alineada con sus expectativas. Para remediar esto, OpenAI lanzó el modelo InstructGPT, una variación del modelo GPT-3 que introdujo el uso de la técnica de aprendizaje reforzado a partir de retroalimentación humana *(Reinforcement learning from human feedback (RLHF))* (Ray, 2023). En la subsección 2.1.2.3 se explica con más detalle esta técnica. El resultado de InstructGPT fue una mejora en la ejecución de tareas, reducción de respuestas no deseadas y personalización de conversaciones con humanos.

Por otro lado, Google, viendo como los LLMs de OpenAI podían ponerle en riesgo su modelo de negocio de búsquedas en internet, lanzó PaLM, un modelo de 540.000 millones de parámetros. Éste fue el modelo más masivo del momento y se exploraron las capacidades de escalado y rendimiento.

Durante los siguientes meses, también Meta (antigua Facebook) empezó a experimentar y desarrollar LLMs, empezaron con Galactica, OPT-IML y NLLB, pero fueron investigaciones internas que no liberaron al gran público.

El 30 de noviembre de 2022, OpenAI lanzó ChatGPT, un modelo que interaccionaba con el usuario en forma de conversación. Se basó en la arquitectura de GPT-3.5 y fue entrenado con un gran corpus de datos de texto y ajustado para la tarea específica de generar respuestas similares a las humanas. ChatGPT fue una revolución para el público general, ya que se lanzó junto a una interfaz de usuario muy amigable que permitía a la gente interactuar con el modelo y ver su gran potencial. Además, junto a este lanzamiento, se presentó GPT-3.5 Turbo, una versión más eficiente, rápida y económica.

El siguiente gran año fue 2023, en el cual se presentaron novedades de LLMs casi cada mes. A continuación se muestran los hitos más importantes del año:

- Febrero 2023: Meta lanza LLaMA *(Large Language Model Meta AI (LLaMA))* un modelo de 65.000 millones de parámetros entrenado en un corpus de 20 idiomas y lo pone a disposición de la comunidad científica. Los investigadores realizaron ajustes finos del modelo creando variaciones como el llamado Alpaca de la Universidad de Stanford. Además, al cabo de una semana de su publicación se acabó filtrando el modelo a través de internet y se hizo viral.

- Marzo 2023: Google presenta Bard, un modelo conversacional equivalente a ChatGPT pero basado en su modelo LaMDA y lo abre de forma limitada. Por otro lado, OpenAI, lanza GPT-4, un modelo cuya arquitectura mantienen como secreto de industria, publicando simplemente un reporte técnico (OpenAI, 2023a). GPT-4 demuestra mejoras en la mayoría de pruebas de evaluación, mayor conocimiento, adaptabilidad y la capacidad de entender el contexto de textos más largos.

- Mayo 2023: Google lanza el modelo PaLM 2, una versión optimizada con 340.000 millones de parámetros y a su vez actualiza Bard para que use PaLM 2 y lo pone a disposición del público general. OpenAI también da un paso más, lanzando ChatGPT en una aplicación móvil, extendiendo así su accesibilidad y mejorando la experiencia de usuario. Por otro lado, la empresa Stability AI presenta el modelo StableVicuna, el primer modelo conversacional de código abierto que utilizó la técnica de RLHF.

- Julio 2023: Meta lanza LLaMA 2, un conjunto de tres modelos de 7.000, 13.000 y 70.000 millones de parámetros. Esta publicación se hace bastante importante ya que Meta introduce una licencia de uso comercial que permite el acceso abierto a los modelos con ciertas cláusulas de responsabilidad y protección. El hecho que se publicara en este formato alertó a otras empresas de código privado como OpenAI o Google.

- Septiembre 2023: La empresa francesa Mistral AI publica su modelo Mistral 7B, con 7.300 millones de parámetros bajo una licencia Apache 2.0 y lo publica en Hugging Face, que, en el momento, es la plataforma de referencia para LLMs de código abierto.

- Noviembre 2023: La *start-up* de Elon Musk, xAI, publica un modelo conversacional llamado "Grok-1". Sin embargo, no llega a estar al nivel de los mejores modelos como GPT-4. Por otro lado, OpenAI realiza un evento donde presenta GPT-4 Turbo, un modelo con 128.000 tokens de contexto, mejoras de rendimiento y precios menores al usuario. Además, presenta una API *(Application Programming Interface (API))* para el desarrollo de asistentes con capacidad de recuperación de la información y acciones para interactuar con APIs externas. No solo esto, sino que también lanzaron "GPTs", una evolución de la plataforma de ChatGPT que permitía la creación de GPTs personalizados, combinando instrucciones, conocimiento y acciones mediante conexiones a otras APIs.

- Diciembre 2023: Google, sin querer quedarse atrás, lanza Gemini, una colección de modelos (Gemini Nano, Gemini Pro y Gemini Ultra) y pone a disposición Gemini Pro. Según el anuncio, Gemini Ultra era capaz de superar a GPT-4 en la mayoría de pruebas de evaluación, sin embargo, no fue abierto al público en ese momento. Además, Bard se actualizó para utilizar los nuevos modelos Gemini. Finalmente, Mistral AI publica el modelo de código abierto Mixtral 8x7B, que revoluciona el sector al ser un modelo con arquitectura MoE *(Mixture of Experts (MoE))*.

### 2.1.2. Actualidad

Los últimos años de desarrollo en el campo de los LLMs han sido frenéticos, hasta llegar a la actualidad. Como se ha visto en la sección 2.1.1, OpenAI ha sido la compañía que ha marcado el ritmo de innovación y el estado del arte de los LLMs.

A principios del año 2024, momento en el que se escribe esta investigación, el estado del arte de los LLMs está marcado por modelos como GPT-4, GPT-4 Turbo, Gemini Pro o Mixtral 8x7B. Respecto a sus arquitecturas, OpenAI y Google no las han publicado, pero se sospecha que son MoE. Para más detalle, consultar la subsección 2.1.2.2.

También se incluyen los modelos conversacionales como ChatGPT (basado en GPT-4) o Bard (basado en Gemini Pro), que presentan interfaces gráficas sencillas y una barrera de entrada muy pequeña para que el gran público pueda usarlos.

Asimismo, no sólo son importantes los modelos, sino el ecosistema con el cual se ponen a disposición. Para los desarrolladores, se utilizan APIs de inferencia y de creación de asistentes con sistemas de recuperación de la información, conocimiento personalizado y acciones mediante APIs externas.

El 10 de enero de 2024, OpenAI publicó "GPT Store", una plataforma digital de desarrollo sin código para crear, compartir y monetizar LLMs basados en ChatGPT. Este hito puede ser una revolución para la creación y el uso de LLMs por parte del público general.

Respecto a esta sección, a continuación se presentan las técnicas que se utilizan a día de hoy para desarrollar los LLMs más avanzados. Se describe cuál es el proceso de preprocesado de datos, los diferentes modelos y arquitecturas, cómo se hace el entrenamiento, los ajustes para tareas posteriores, la puesta en producción, interfaces de usuario, evaluaciones, diferentes retos y el tema de la seguridad e IA responsable.

#### 2.1.2.1. Preprocesado de conjuntos de datos

El primer paso para desarrollar un LLM es preparar el conjunto de datos con el cual se entrenará al modelo.

Los datos se suelen clasificar en corpus según el tipo de texto y su fuente. Por ejemplo, existen conjuntos de datos basados en libros, páginas web, código de programación, artículos científicos,

entre otros. En la práctica, para obtener un buen rendimiento e información relevante, se suelen necesitar diversos de estos corpus para pre-entrenar un LLM.

Como referencia, según Zhao et al. (2023), algunos de los conjuntos de datos utilizados por LLMs y publicados han sido:

- GPT-3 (175.000 millones de parámetros) fue entrenado con un conjunto de datos mixto de 300.000 millones de tokens, incluyendo corpus como CommonCrawl (páginas web), WebText2 (hilos de Reddit), Books1, Books2 (libros) y Wikipedia.
- PaLM (540.000 millones de parámetros) utilizó un conjunto de datos de 780.000 millones de tokens, que provienen de conversaciones en las redes sociales, páginas web filtradas, libros, Github, Wikipedia multilingüe, y noticias.
- LLaMA se entrenó con datos de varias fuentes, incluyendo CommonCrawl (páginas web), C4 (páginas web), Github, Wikipedia, libros, ArXiv y StackExchange. El tamaño de los datos de entrenamiento para LLaMA 6B y LLaMA 13B es de 1,0T de tokens, mientras que LLaMA 32B y LLaMA 65B utilizaron 1,4T de tokens.

La Figura 2.2 presenta algunos ejemplos del contenido de los conjuntos de datos que se utilizaron para entrenar LLMs.

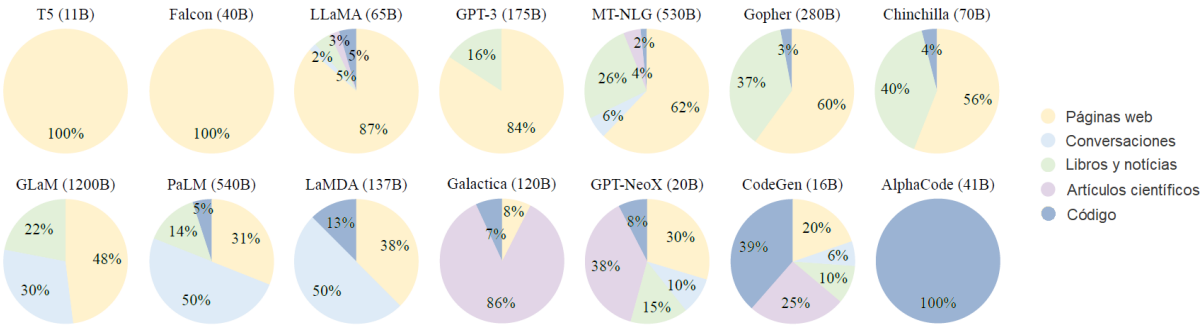


Figura 2.2: Composición de conjuntos de datos de algunos LLMs. Adaptado de Zhao et al. (2023).

Una de las mayores limitaciones de GPT-3, que se entrenó solamente con texto plano, fue la falta de habilidades de razonamiento, creación de código y resolución de problemas matemáticos. Esto fue debido a la falta de un conjunto de datos que tuviera este tipo de ejemplos. Posteriormente, con GPT-4 se ha mejorado en este aspecto. Por otro lado, para los modelos tipo ChatGPT y Bard, se requieren colecciones de datos de conversaciones generadas por humanos.

Respecto al tamaño del conjunto de datos, se ha demostrado que el efecto de escalarlo mejora las capacidades hasta cierto punto en arquitecturas basadas en *transformers*. GPT-3 y PaLM exploraron los límites del tamaño utilizando conjuntos de 300.000 y 780.000 millones de tokens respectivamente.

Sin embargo, los recursos computacionales necesarios para entrenar y utilizar estos modelos tan masivos pueden ser excesivos (Zhao et al., 2023).

Una vez se seleccionan los diferentes corpus, se procede al preprocesado de los datos. La Figura 2.3 muestra el proceso general y a continuación se detalla cada paso (Hadi et al., 2023).

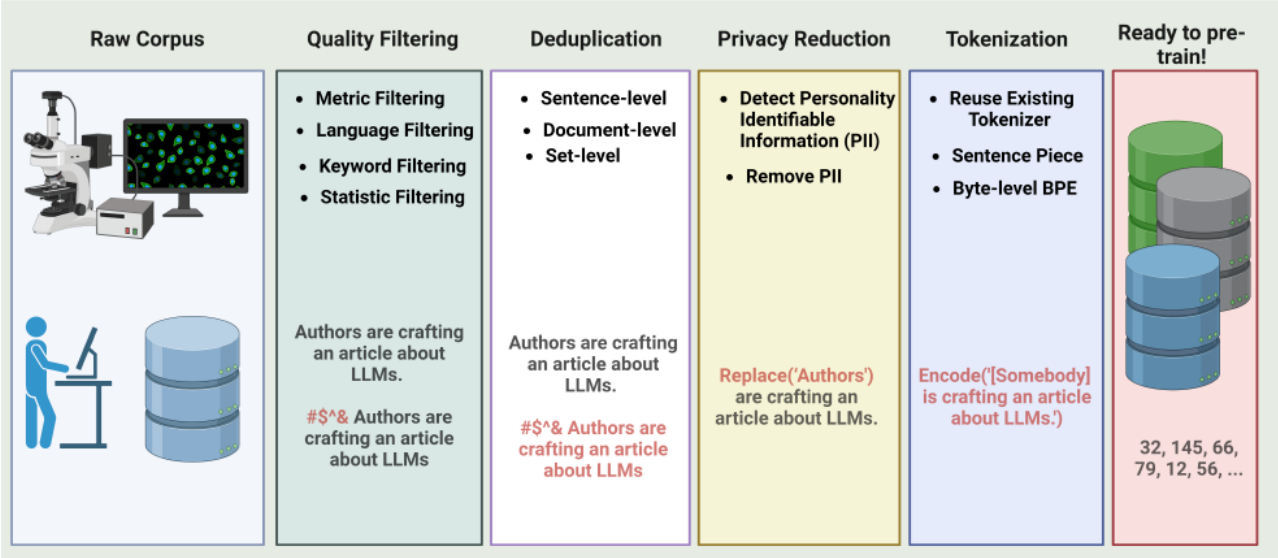


Figura 2.3: Preprocesado del conjunto de datos para el pre-entrenamiento de LLMs. Tomado de Hadi et al. (2023).

El primer paso consiste en realizar un filtrado de calidad, donde se eliminan los datos de bajo valor. Principalmente, existen dos técnicas, una basada en clasificadores y otra basada en heurística. En la primera, se entrena un clasificador que es capaz de distinguir datos de calidad. Sin embargo, se ha demostrado que estos clasificadores pueden llegar a eliminar datos de valor como conversaciones informales. Por otro lado, la técnica heurística se basa en el uso de determinados filtros, como de lenguaje, de métricas (perplejidad), estadísticas (ratio símbolos-textos, puntuaciones, etc.) o filtrado a base de palabras clave.

El segundo paso es la de-duplicación de información, que se puede realizar a nivel de oración, de documento o de conjunto de datos. Se deben eliminar aquellos textos que contengan palabras y frases repetidas. Además, se estudia el solapamiento entre documentos, comparando sus contenidos y eliminándolos si fuera necesario.

A continuación se realiza la reducción de privacidad, donde se elimina la información personal identificable *(Personal Identifiable Information (PII))*. Para ello se utilizan modelos o técnicas basadas en reglas, como la detección de palabras clave, para identificar y eliminar PII, incluyendo nombres, direcciones, números de teléfono, entre otros.

El último paso antes de tener el conjunto de datos preparado para su uso es la *tokenización*. La *tokenización* consiste en segmentar el texto en tokens individuales que luego se usan como entradas en los LLMs. Existen tres principales métodos de *tokenización*:

- Tokenización de Codificación de Pares de Bytes (BPE): Originalmente era un algoritmo de compresión de datos de 1994, adaptado para NLP. Comienza con símbolos básicos y combina pares frecuentes de tokens consecutivos. Se utiliza en modelos como GPT-3, BART y LLaMA.
- Tokenización WordPiece: Desarrollado por Google, es similar al BPE, pero utiliza un modelo de lenguaje para puntuar y seleccionar pares para fusionar. Este método se empleó en BERT.
- Tokenización Unigram: Empieza con un conjunto grande de subcadenas posibles y elimina tokens iterativamente. Se basa en un modelo de lenguaje unigram y se utiliza en modelos como T5 y mBART.

Normalmente, se utilizan *tokenizadores* ya existentes, sin embargo, si la tarea o los datos son muy especializados, podría llegar a ser beneficioso diseñar una *tokenización* personalizada.

#### 2.1.2.2. Arquitecturas de los modelos

En esta sección se presentan las arquitecturas más comunes que se utilizan para desarrollar LLMs. Nótese que las arquitecturas más recientes, como el caso de GPT-4, no han sido publicadas y podrían contener cambios o evoluciones significativas.

Los LLMs se basan en los mecanismos de atención presentados en 2014 por Bahdanau y Cho (2014) y la arquitectura del *transformer* publicada en 2017 por Vaswani et al. (2017). Gracias a su capacidad de paralelización han permitido desarrollar modelos de cientos de miles de millones de parámetros.

Según Zhao et al. (2023), en general, las arquitecturas más comunes se pueden clasificar en tres tipos: *encoder-decoder*, *causal decoder* y *prefix decoder*.

- *Encoder-decoder* es la arquitectura por defecto del *transformer*. Contiene dos partes principales: el codificador y el decodificador. El codificador utiliza capas de auto-atención para procesar la secuencia de entrada y crea los vectores de la misma. Por otro lado, el decodificador las utiliza para generar la secuencia objetivo de manera auto-regresiva, generando cada parte de la secuencia una tras otra. Actualmente, muy pocos LLMs utilizan esta arquitectura ya que ha sido superada por otras.
- *Causal decoder* utiliza atención unidireccional para asegurar que cada *token* de entrada sólo pueda relacionarse con *tokens* anteriores y consigo mismo, evitando así el acceso a información futura. Esto es crucial para la generación de lenguaje coherente y secuencial. Por otro lado, los tokens de entrada y salida se manejan de la misma manera, lo que contribuye a que el proceso de la información sea uniforme. Esta arquitectura es ampliamente utilizada en LLMs como GPT-3, LLaMA 2, PaLM y BLOOM, entre otros.

- *Prefix decoder* permite la atención bidireccional sobre los *tokens* de prefijo y unidireccional sobre los generados. Funciona de forma similar a la arquitectura de codificador-decodificador, codificando bidireccionalmente la secuencia de prefijo y prediciendo autoregresivamente los tokens de salida, uno a uno. Comparte parámetros durante la codificación y decodificación, lo que optimiza el proceso. Esta arquitectura sugiere evolucionar desde decodificadores causales para una convergencia más rápida. Ejemplos incluyen U-PaLM, derivado de PaLM, y otros modelos de lenguaje grandes como GLM-130B.

A continuación, la Tabla 2.1 muestra diferentes LLMs con su categoría de arquitectura y otros parámetros como su tamaño, el tipo de normalización, el tipo de codificación posicional *(Positional Encoding (PE))*, $\#L$ el número de capas, $\#H$ el número de cabezas de atención, $d_{\text{model}}$ el tamaño de los estados ocultos, y MCL la longitud máxima de contexto durante el entrenamiento.

| Model | Category | Size | Normalization | PE | Activation | Bias | #L | #H | $d_{model}$ | MCL |
|---|---|---|---|---|---|---|---|---|---|---|
| GPT3 | Causal decoder | 175B | Pre LayerNorm | Learned | GeLU | ✓ | 96 | 96 | 12288 | 2048 |
| PanGU-$\alpha$ | Causal decoder | 207B | Pre LayerNorm | Learned | GeLU | ✓ | 64 | 128 | 16384 | 1024 |
| OPT | Causal decoder | 175B | Pre LayerNorm | Learned | ReLU | ✓ | 96 | 96 | 12288 | 2048 |
| PaLM | Causal decoder | 540B | Pre LayerNorm | RoPE | SwiGLU | × | 118 | 48 | 18432 | 2048 |
| BLOOM | Causal decoder | 176B | Pre LayerNorm | ALiBi | GeLU | ✓ | 70 | 112 | 14336 | 2048 |
| MT-NLG | Causal decoder | 530B | - | - | - | - | 105 | 128 | 20480 | 2048 |
| Gopher | Causal decoder | 280B | Pre RMSNorm | Relative | - | - | 80 | 128 | 16384 | 2048 |
| Chinchilla | Causal decoder | 70B | Pre RMSNorm | Relative | - | - | 80 | 64 | 8192 | - |
| Galactica | Causal decoder | 120B | Pre LayerNorm | Learned | GeLU | × | 96 | 80 | 10240 | 2048 |
| LaMDA | Causal decoder | 137B | - | Relative | GeGLU | - | 64 | 96 | 8192 | - |
| Jurassic-1 | Causal decoder | 178B | Pre LayerNorm | Learned | GeLU | ✓ | 76 | 96 | 13824 | 2048 |
| LLaMA | Causal decoder | 65B | Pre RMSNorm | RoPE | SwiGLU | × | 80 | 64 | 8192 | 2048 |
| LLaMA 2 | Causal decoder | 70B | Pre RMSNorm | RoPE | SwiGLU | × | 80 | 64 | 8192 | 4096 |
| Falcon | Causal decoder | 40B | Pre LayerNorm | RoPE | GeLU | × | 60 | 96 | 8192 | 2048 |
| GLM-130B | Prefix decoder | 130B | Post DeepNorm | RoPE | GeGLU | ✓ | 70 | 96 | 12288 | 2048 |
| T5 | Encoder-decoder | 11B | Pre RMSNorm | Relative | ReLU | × | 24 | 128 | 1024 | 512 |

Tabla 2.1: Arquitecturas de varios LLMs que han sido publicados. Se muestra la categoría de arquitectura, su tamaño, el tipo de normalización, el tipo de codificación posicional *(Positional Encoding (PE))*, $\#L$ denota el número de capas, $\#H$ denota el número de cabezas de atención, $d_{\text{model}}$ denota el tamaño de los estados ocultos, y MCL denota la longitud máxima de contexto durante el entrenamiento. Tomado de Zhao et al. (2023).

Un inconveniente de las arquitecturas anteriores es que cada vez que el modelo genera un texto de salida, éste debe de usar todos sus parámetros. Por lo que si se utilizan modelos masivos, el poder de computación necesario y su precio pueden acabar siendo un problema. Aquí es donde la arquitectura de Mezcla de Expertos *(Mixture of Experts (MoE))* presenta una solución. En este caso, solamente un subconjunto de pesos de la red neuronal se activan para dar el resultado, necesitando menos computación y aumentando la eficiencia y eficacia del modelo. Además, se ha demostrado que se puede observar una mejora sustancial en el rendimiento aumentando el número de expertos (Zhao

et al., 2023). Un ejemplo de esta arquitectura lo muestra el modelo Mixtral 8x7B, y potencialmente GPT-4 y GPT-4 Turbo.

#### 2.1.2.3. Pre-entrenamiento

El entrenamiento de LLMs generalmente consiste en dos fases: un pre-entrenamiento y un ajuste a tareas concretas. En esta sección se explica el pre-entrenamiento de los modelos y en la sección 2.1.2.4 se presenta el ajuste a tareas posteriores.

El pre-entrenamiento es el proceso inicial donde el modelo aprende patrones del lenguaje y estructuras contextuales al realizar un entrenamiento con grandes cantidades de texto. El objetivo es optimizar los parámetros del modelo para maximizar la verosimilitud (*likelihood*) al generar la siguiente palabra dado un texto con un determinado contexto. Esta optimización normalmente se realiza con el gradiente del descenso estocástico *(Stochastic Gradient Descent (SGD))* o alguna de sus variantes como Adam *(Adaptive Moment Estimation (ADAM))*. Además se combina con la técnica de *backpropagation*, que calcula los gradientes para actualizar los parámetros del modelo en cada iteración de entrenamiento (Hadi et al., 2023).

La mayoría de LLMs se pre-entrenan con un proceso llamado pre-entrenamiento generativo autosupervisado. Dado un conjunto de datos de texto, el modelo predice los *tokens* que deberían generarse y lo compara con el texto original. De esta manera no se necesita etiquetado de datos y el entrenamiento puede escalar masivamente.

Por otro lado, existen dos estilos de pre-entrenamiento generativo:

- Autorregresivo: El objetivo es predecir la siguiente palabra dado un texto de entrada. Esta técnica da lugar a los famosos *Generative Pre-trained Transformers (GPT)* y han sido la base del entrenamiento de los modelos de OpenAI (GPT-1, GPT-2, GPT-3).
- Enmascarado: El objetivo es predecir una determinada palabra enmascarada en mitad de una frase. Esta técnica fue la que se utilizó para modelos como BERT de Google.

| Tipo de pre-entrenamiento | Entrada | Predicción |
|---|---|---|
| Autorregresivo | Iré a ver una película _____ | Iré a ver una película al cine. |
| Enmascarado | Iré a ver una _____ al cine. | Iré a ver una película al cine. |

Tabla 2.2: Ejemplos de pre-entrenamiento autorregresivo y enmascarado.

El pre-entrenamiento generativo de LLMs suele requerir cientos o incluso miles de GPUs o TPUs *(Tensor Processing Unit (TPU))*. Por ejemplo, el modelo LLaMA original reportó haber utilizado 2048 GPUs del tipo NVIDIA A100-80G. Normalmente, se utilizan los FLOPS *(FLoating point number Operations Per Second (FLOPS))* para estimar los recursos computacionales necesarios(Zhao et al., 2023). Por otro lado, C. Li (2020) estimó que el coste de entrenamiento de GPT-3 podría ser de

unos 4,6 millones de dólares, utilizando una infraestructura de GPUs Tesla V100 paralelizadas con un gran ancho de banda proporcionadas por Microsoft.

#### 2.1.2.4. Ajuste a tareas posteriores

En esta sección se presentan las técnicas de ajuste para realizar tareas posteriores más concretas. Estos ajustes son adaptaciones de LLMs pre-entrenados para tareas específicas que no estaban directamente previstas durante el pre-entrenamiento inicial. A continuación se explican el ajuste fino (*Fine-tuning*), el aprendizaje reforzado a partir de retroalimentación humana (*Reinforcement learning from human feedback (RLHF)*), el ajuste de instrucciones (*Instruction tuning*) y la generación aumentada por recuperación (*Retrieval-Augmented Generation (RAG)*).

**Ajuste fino (*Fine-tuning*)**

El ajuste fino o *fine-tuning* es el proceso de adaptación de un LLM pre-entrenado a una nueva tarea o un nuevo conjunto de datos, realizando un entrenamiento posterior. El ajuste fino implica el cambio de ciertos parámetros del modelo pre-entrenado para optimizarlo a la nueva tarea o a la inclusión de nueva información. A diferencia del pre-entrenamiento, que usa una gran cantidad de datos y tarda mucho tiempo, el *fine-tuning* suele ser más rápido y utiliza menos datos (Yenduri et al., 2023). La técnica que se utiliza es la de ajuste fino supervisado *(Supervised Fine-Tuning (SFT))*, que hace uso de un conjunto de datos etiquetados para incluir información de la nueva tarea (Wu et al., 2023).

**Aprendizaje reforzado a partir de retroalimentación humana (*Reinforcement learning from human feedback (RLHF)*)**

El algoritmo de aprendizaje reforzado a partir de retroalimentación humana (RLHF) fue introducido en enero de 2022 con el lanzamiento del modelo InstructGPT de OpenAI (Lowe y Leike, 2022). Anteriormente ya existían algoritmos de aprendizaje reforzado, pero lo que hizo OpenAI fue adaptarlo al ajuste de un LLM, en este caso GPT-3, para preparar al modelo para realizar la tarea de tener conversaciones naturales.

El RLHF consiste en utilizar las preferencias humanas como una señal de recompensa para ajustar al modelo. Primero, se recopila un conjunto de datos de demostraciones escritas por humanos y se realiza un aprendizaje supervisado de base. Luego, se recopila un conjunto de datos que contiene, para cada consulta al modelo, dos respuestas y, además, la etiqueta de cuál prefiere el humano. Seguidamente, se entrena un modelo de recompensa para predecir qué salida preferirían los etiquetadores humanos. Finalmente, se utiliza este modelo como función de recompensa y se ajusta el modelo para maximizarla mediante el algoritmo PPO (Lowe y Leike, 2022).

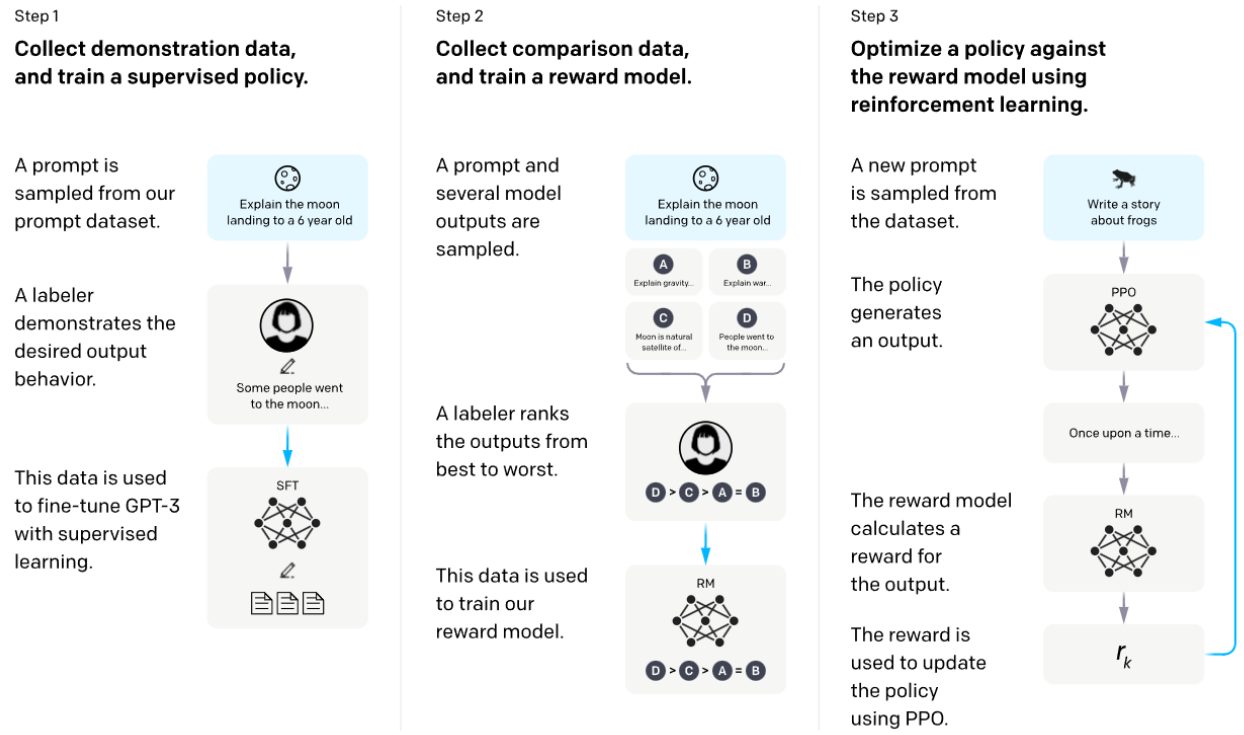


Figura 2.4: Proceso de aprendizaje reforzado a partir de retroalimentación humana (RLHF). Tomado de Lowe y Leike (2022).

### Ajuste de instrucciones (*Instruction tuning*)

El ajuste de instrucciones (*Instruction tuning*) se basa en optimizar y ajustar los LLMs no sólo con datos, sino con instrucciones y directrices específicas. Así pues, el modelo es entrenado adicionalmente con un conjunto de instrucciones. Estas instrucciones son formulaciones claras y directas de lo que se espera que el modelo haga en diversas situaciones. Normalmente, las instrucciones son pares de preguntas y respuestas o comandos y acciones esperadas. Después, éstas se utilizan para afinar el modelo de manera supervisada (Zhao et al., 2023).

### Generación aumentada por recuperación (*Retrieval-Augmented Generation (RAG)*)

La generación aumentada por recuperación (*Retrieval-Augmented Generation (RAG)*) combina la recuperación de la información con la generación del lenguaje mediante LLMs. Con esta técnica, se busca información relevante para la consulta del usuario en una base de datos, que puede ser una base de datos vectorial con los *embeddings* de textos externos. La idea es identificar segmentos que sean potencialmente útiles para responder a la pregunta o realizar la tarea pedida. Una vez recuperada la información relevante, el sistema utiliza un LLM para generar una respuesta cohesionada y con el contexto correcto. El LLM tomará la información recuperada como entrada, y generará una

respuesta natural (Gao et al., 2024).

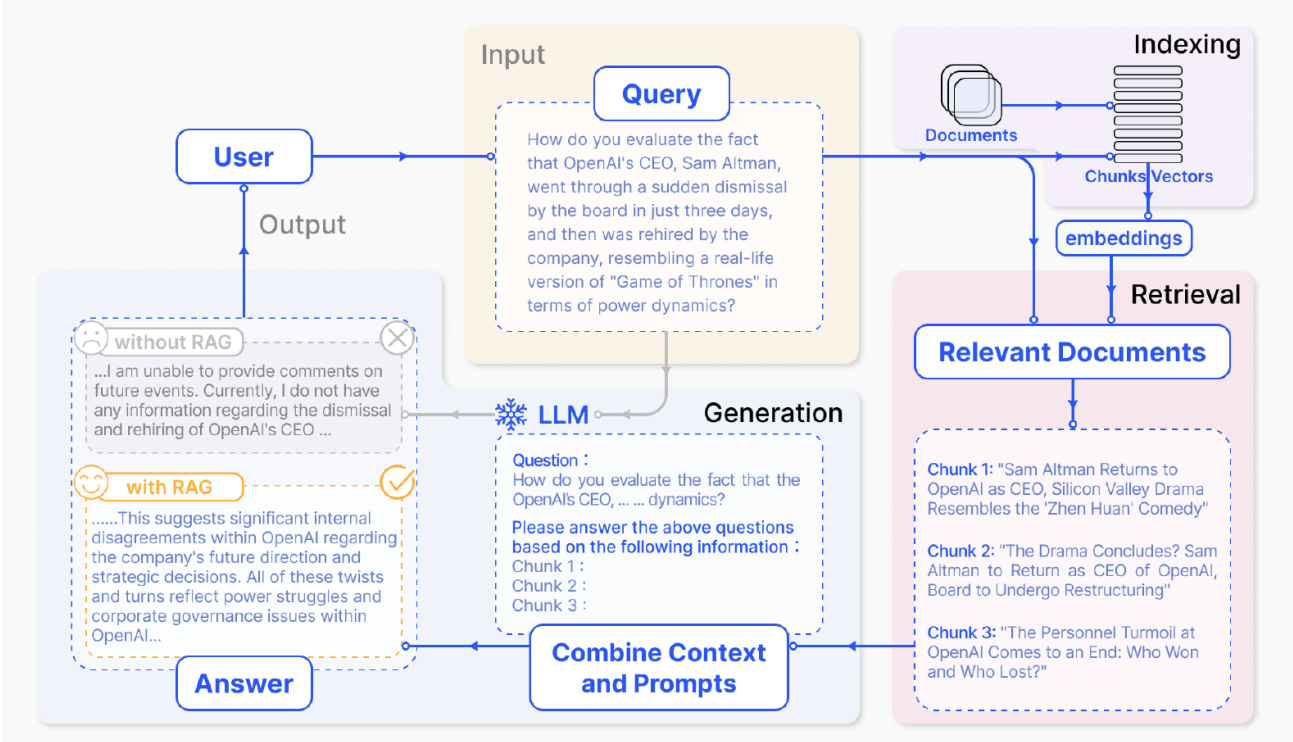


Figura 2.5: Proceso de generación aumentada por recuperación (RAG). Tomado de Gao et al. (2024).

#### 2.1.2.5. Interfaces de usuario

Para interactuar con un LLM como usuario, existen diferentes interfaces de usuario. Entre las más destacadas, se encuentra la interfaz de *prompts* de texto, que permite a los usuarios introducir comandos o preguntas en formato textual. Esta interfaz es esencial para facilitar un diálogo, donde la interacción se desarrolla a través de una secuencia de mensajes escritos.

Por otro lado, las interfaces de audio están ganando popularidad, especialmente en aplicaciones donde la interacción por voz es más natural. Estas interfaces suelen incluir un sistema de transcripción audio-texto, que convierte el habla del usuario en texto, permitiendo al LLM procesar la solicitud y responder. Esta respuesta puede ser devuelta en formato de texto o, mediante el uso de tecnología de síntesis de voz, convertida nuevamente en audio.

Respecto al desarrollo de aplicaciones, se utilizan las APIs para conectar los LLMs y las aplicaciones finales. La aplicación envía datos (como texto o comandos de voz transcritos) a un servidor que aloja el LLM. Este servidor procesa la entrada, ejecuta el modelo para generar una respuesta y luego la envía de vuelta. Este proceso se realiza generalmente en tiempo real, permitiendo interacciones fluidas y dinámicas. Sin embargo, tiene un coste asociado que el desarrollador paga al que aloja el LLM y proporciona la API.

#### 2.1.2.6. Evaluación de LLMs y habilidades emergentes

Una vez que disponemos de LLMs entrenados y ajustados a tareas posteriores, se realizan diferentes tipos de pruebas de evaluación para conocer las capacidades de los mismos y poder compararlos con otros LLMs. Además, con estas evaluaciones también se puede medir las habilidades emergentes de los modelos.

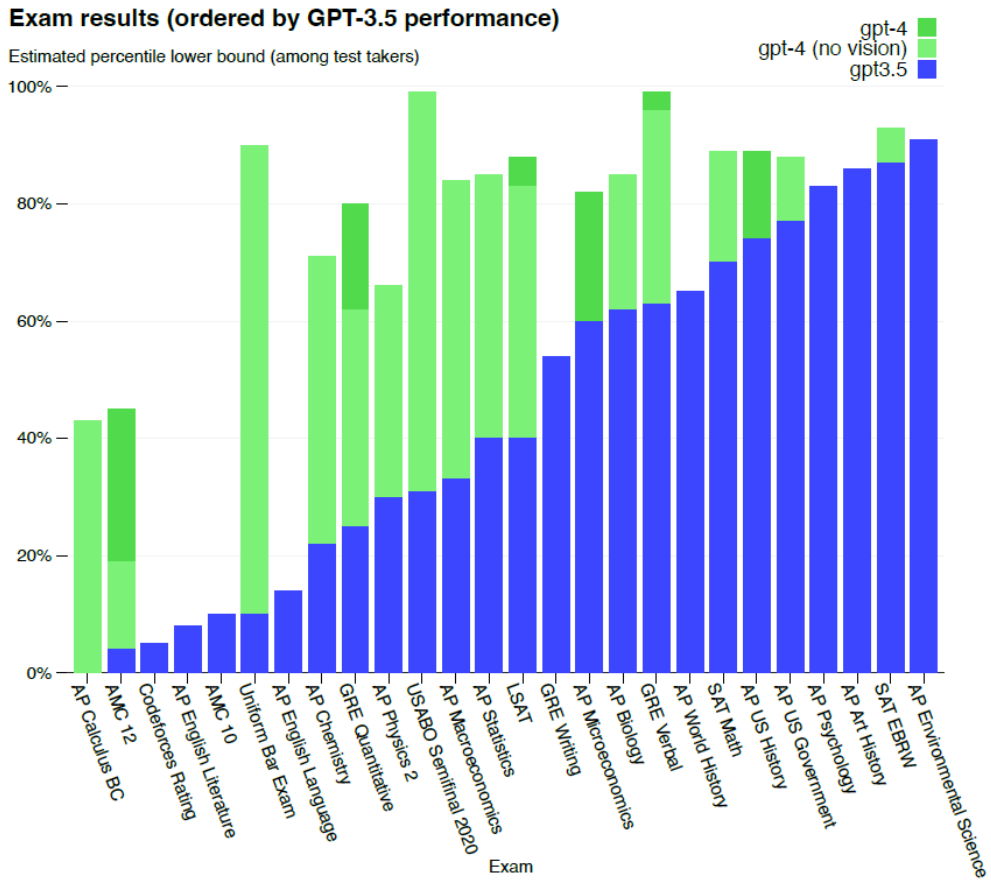


Figura 2.6: Exámenes humanos utilizados para evaluar LLMs. Tomado de OpenAI (2023a).

La evaluación de LLMs no es una tarea sencilla como podría ser la de un modelo de aprendizaje automático estándar. En este caso se debe valorar capacidades como la compresión, la generación de texto coherente, el conocimiento, la capacidad de adaptación, los sesgos, la capacidad multilingüe o la capacidad de extrapolación de tareas. Para ello, actualmente se utilizan dos tipos de pruebas: (1) Exámenes que realizan humanos (por ejemplo, el GRE) y (2) Comparaciones (*benchmarks*) de diferentes capacidades (generales, de razonamiento, matemáticas o de código). La Figura 2.6 muestra los exámenes humanos más habituales y los resultados de los LLMs de OpenAI (OpenAI, 2023a).

Por otro lado, las Figuras 2.7 y 2.8 presentan pruebas especialmente diseñadas para la evaluación de LLMs. En la primera, se muestra la tabla de comparación de GPT-4 con los modelos que había disponibles en el momento de su lanzamiento. En la segunda, se muestra una comparación de

diciembre de 2023, momento en el que se lanzan los modelos Gemini de Google. Ésta muestra que Gemini Ultra supera a GPT-4 en la mayoría de pruebas, sin embargo, Google todavía no ha abierto al público este modelo.

| | GPT-4<br>Evaluated few-shot | GPT-3.5<br>Evaluated few-shot | LM SOTA<br>Best external LM evaluated few-shot | SOTA<br>Best external model (incl. benchmark-specific tuning) |
|---|---|---|---|---|
| MMLU [49]<br>Multiple-choice questions in 57 subjects (professional & academic) | **86.4%**<br>5-shot | 70.0%<br>5-shot | 70.7%<br>5-shot U-PaLM [50] | 75.2%<br>5-shot Flan-PaLM [51] |
| HellaSwag [52]<br>Commonsense reasoning around everyday events | **95.3%**<br>10-shot | 85.5%<br>10-shot | 84.2%<br>LLaMA (validation set) [28] | 85.6<br>ALUM [53] |
| AI2 Reasoning Challenge (ARC) [54]<br>Grade-school multiple choice science questions. Challenge-set. | **96.3%**<br>25-shot | 85.2%<br>25-shot | 85.2%<br>8-shot PaLM [55] | 86.5%<br>ST-MOE [18] |
| WinoGrande [56]<br>Commonsense reasoning around pronoun resolution | **87.5%**<br>5-shot | 81.6%<br>5-shot | 85.1%<br>5-shot PaLM [3] | 85.1%<br>5-shot PaLM [3] |
| HumanEval [43]<br>Python coding tasks | **67.0%**<br>0-shot | 48.1%<br>0-shot | 26.2%<br>0-shot PaLM [3] | 65.8%<br>CodeT + GPT-3.5 [57] |
| DROP [58] (F1 score)<br>Reading comprehension & arithmetic. | 80.9<br>3-shot | 64.1<br>3-shot | 70.8<br>1-shot PaLM [3] | **88.4**<br>QDGAT [59] |
| GSM-8K [60]<br>Grade-school mathematics questions | **92.0%***<br>5-shot chain-of-thought | 57.1%<br>5-shot | 58.8%<br>8-shot Minerva [61] | 87.3%<br>Chinchilla + SFT+ORM-RL, ORM reranking [62] |

Figura 2.7: Principales pruebas (*benchmarks*) para evaluar LLMs. Tomado de OpenAI (2023a).

Según Pichai y Hassabis (2023) y OpenAI (2023a), encontramos evaluaciones de conocimiento general como MMLU, que contiene preguntas de múltiple respuesta sobre 57 temas diferentes, incluyendo ciencias, humanidades y otros. También tenemos pruebas de razonamiento (como HellaSwag, Big-Bench Hard o DROP), pruebas de matemáticas (como GSM8K o MATH) y evaluaciones de código (como HymanEval o Natural2Code).

**TEXT**

| Capability | Benchmark<br>Higher is better | Description | Gemini Ultra | GPT-4<br>API numbers calculated where reported numbers were missing |
|---|---|---|---|---|
| General | MMLU | Representation of questions in 57 subjects (incl. STEM, humanities, and others) | **90.0%**<br>CoT@32* | 86.4%<br>5-shot**<br>(reported) |
| Reasoning | Big-Bench Hard | Diverse set of challenging tasks requiring multi-step reasoning | **83.6%**<br>3-shot | 83.1%<br>3-shot<br>(API) |
| | DROP | Reading comprehension (F1 Score) | **82.4**<br>Variable shots | 80.9<br>3-shot<br>(reported) |
| | HellaSwag | Commonsense reasoning for everyday tasks | 87.8%<br>10-shot* | **95.3%**<br>10-shot*<br>(reported) |
| Math | GSM8K | Basic arithmetic manipulations (incl. Grade School math problems) | **94.4%**<br>maj1@32 | 92.0%<br>5-shot CoT<br>(reported) |
| | MATH | Challenging math problems (incl. algebra, geometry, pre-calculus, and others) | **53.2%**<br>4-shot | 52.9%<br>4-shot<br>(API) |
| Code | HumanEval | Python code generation | **74.4%**<br>0-shot (IT)* | 67.0%<br>0-shot*<br>(reported) |
| | Natural2Code | Python code generation. New held out dataset HumanEval-like, not leaked on the web | **74.9%**<br>0-shot | 73.9%<br>0-shot<br>(API) |

* See the technical report for details on performance with other methodologies
** GPT-4 scores 87.29% with CoT@32 - see the technical report for full comparison

Figura 2.8: Comparación de Gemini Ultra y GPT-4 en las principales pruebas (*benchmarks*) de evaluación de LLMs. Tomado de Pichai y Hassabis (2023).

#### 2.1.2.7. Retos y consideraciones

El desarrollo de LLMs ha sido controvertido por diferentes aspectos, incluyendo la privacidad, las alucinaciones, los sesgos, su escasa explicabilidad, sus implicaciones sociales y éticas y su seguridad. En esta sección se explican cada uno de estos elementos.

**Privacidad**

Como se ha visto en la sección 2.1.2.1, los LLMs requieren de masivas cantidades de datos textuales para ser entrenados. Estos conjuntos de datos, en muchos casos se extraen de internet y pueden incluir información personal o sensible. Por esta razón, algunos LLMs pueden plantear preocupaciones sobre la privacidad. Los modelos pueden aprender datos personales y posteriormente dar detalles privados. Entre las soluciones utilizadas para este problema, se incluyen la anonimización de datos durante el pre-procesado de conjuntos de entrenamiento. Por otro lado, también se han utilizado materiales, por ejemplo libros, publicaciones o notícias, con derechos de autor sin remunerar a los autores originales. Este aspecto también ha creado polémica, con varias demandas a compañías que desarrollan LLMs. A día de hoy, todavía no existe ninguna regulación al respecto, pero probablemente en los próximos años se crearán.

**Alucinaciones**

La naturaleza de las arquitecturas de los LLMs y el uso de *transformers* hace que los LLMs a veces generen información que puede parecer correcta pero realmente no lo es. A estos fenómenos se los denomina alucinaciones. Las alucinaciones pueden ser problemáticas si los requisitos de veracidad y precisión son elevados, como en el sector médico. Se está trabajando para mitigar este aspecto, por ejemplo, anotando las fuentes de información de donde proviene la información.

**Sesgos**

El entrenamiento de los LLMs y el conocimiento que éstos acaban teniendo se basa en los conjuntos de datos de entrenamiento. Normalmente, estos datos pueden incluir sesgos y prejuicios que reflejan la misma sociedad. Podemos encontrar sesgos de género, de raza, de nacionalidad, de edad, entre otros. Los LLMs entrenados en estos datos pueden perpetuar estos sesgo incluso amplificarlos en sus resultados. Por lo tanto, es crucial que se aborden estos prejuicios con estrategias de mitigación (Liu et al., 2024).

### Explicabilidad

Según Liu et al. (2024), entender cómo un LLM llega a un determinado resultado es un desafío complejo, haciendo difícil evaluar su fiabilidad y la manera en que realiza sus razonamientos. La falta de transparencia puede ser un factor crítico en determinados contextos donde se necesita saber la razón por la cual el LLM ha dado una determinada respuesta. Por ejemplo, en los sectores de salud, justicia o políticas públicas es necesaria una justificación debido a aspectos legales. Actualmente se esta trabajando en mejorar la explicabilidad de los LLMs para garantizar la confianza y la transparencia de los mismos.

### Implicaciones sociales y éticas

Por otro lado, los LLMs pueden tener profundas implicaciones sociales y éticas. Pueden llegar a influir en la opinión de las personas o bien generar información falsa o dañina. Además, determinados estereotipos pueden ser reforzados, aumentando los sesgos de la propia población. Por este motivo es esencial un desarrollo de la seguridad y de la IA responsable (Liu et al., 2024).

Según Ray (2023), la técnica para hacer los LLMs más seguros y responsables se denomina *alineamiento*. A día de hoy, los modelos más utilizados por el público tienen una capa de seguridad que se aplica después del entrenamiento y del ajuste posterior. De esta manera se protege al usuario de usos nocivos, contenidos inapropiados, aspectos de seguridad y se garantiza el cumplimento de normativas y leyes. La Figura 2.9 muestra las capas de entrenamiento, RLHF y seguridad en las que se basa ChatGPT de OpenAI.

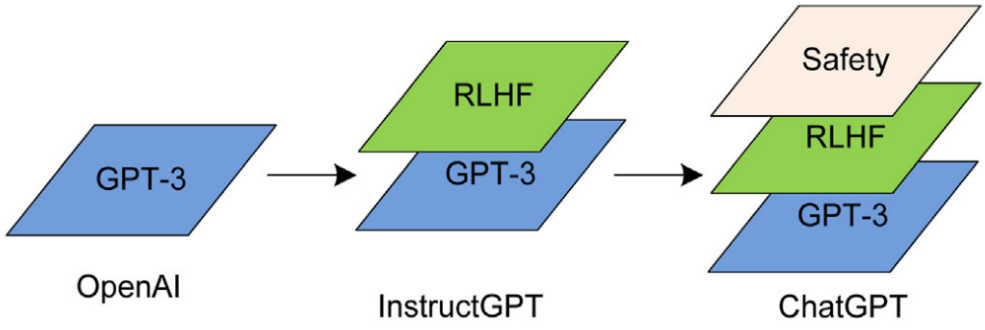


Figura 2.9: Capas del entrenamiento de modelos tipo ChatGPT. Se incluye una fase de preentrenamiento, una fase de RLHF y una última capa de seguridad. Tomado de Ray (2023).

## 2.2. Ingeniería de *prompts*

En esta sección se trata la ingeniería de *promtps*, una rama de la IA generativa que ha surgido como consecuencia de la interacción de los humanos con los LLMs, que tiene el objetivo de optimizar y mejorar los resultados que proporcionan los modelos del lenguaje.

Se define como *prompt* a una instrucción en formato texto o una serie de palabras que se le proporcionan a un modelo del lenguaje como entrada para solicitar una respuesta o para que realice una acción. Los *prompts* son la entrada de los LLMs y pasan por diferentes fases para que el modelo los procese. Primero, se realiza la *tokenización* del texto, donde se divide en unidades más pequeñas, los *tokens*, que pueden ser sílabas, letras o caracteres. Después, los *tokens* se convierten en vectores *embedding*, asignando un vector numérico con unas determinadas dimensiones a cada *token*. Los vectores *embedding* contendrán el significado de las palabras y determinaran si son más similares o no, en un espacio multidimensional. El siguiente paso es el procesamiento de la secuencia mediante la arquitectura del *transformer*, que aplica los mecanismos de auto-atención para comprender el texto y las relaciones entre palabras. Al final, el LLM genera una secuencia de *embeddings* como respuesta, que se transforman a tokens para acabar formando las palabras de la salida.

Así pues, la ingeniería de *prompts (prompt engineering)* es el proceso de estructuración del texto para que sea interpretado y entendido por un LLM con el objetivo de dar una determinada respuesta o realizar una acción (Mayo, 2024).

Según Wei et al. (2022), se ha demostrado que los LLMs tienen múltiples habilidades emergentes y una de ellas es el aprendizaje en contexto *(In-Context Learning (ICL))*. Éste consiste en que los modelos pueden aprender temporalmente de los propios *prompts*. De esta manera, si se definen *prompts* que enseñen al modelo o ejemplifiquen tareas, los LLMs pueden llegar a ser herramientas muy versátiles y adaptables a una gran cantidad de tareas de procesamiento del lenguaje natural. En este sentido, dependiendo del número de ejemplos que se introduzcan en un *prompt*, se pueden clasificar en diferentes tipos. Si el *prompt* no contiene ningún ejemplo, tenemos lo que se denomina *zero-shot learning*, si tiene un ejemplo *one-shot learning*, y si consta de dos o más ejemplos, se clasifica como *few-shot learning*.

Otro concepto importante es el de *ventana de contexto*, que se refiere a la cantidad de *tokens* que un LLM es capaz de aceptar como entrada. Este número dependerá de la arquitectura del modelo y es diferente para cada uno de ellos. Por lo tanto, la *ventana de contexto* representa el límite máximo de cada consulta que le podemos presentar a un LLM. Esta limitación puede afectar al procesamiento de textos largos y el modelo puede perder información relevante de los mismos.

Por otra parte, según Bsharat et al. (2023), para formular buenos *prompts* o instrucciones y obtener respuestas de alta calidad de los LLMs, se deben seguir los siguientes principios de diseño:

- **Concisión y claridad:** Los *prompts* deben ser concisos, evitando información innecesaria que no contribuya a la tarea, siendo al mismo tiempo específico para guiar al modelo.

- **Relevancia contextual:** Los *prompts* deben proporcionar un contexto relevante que ayude al modelo a entender el trasfondo y el dominio de la tarea. Se deben incluir palabras de la terminología del dominio o descripciones concretas de la situación.
- **Alineación con la tarea:** Los *prompts* deben estar alineados con la tarea en cuestión, utilizando un lenguaje y estructura adecuados a la consulta. Por ejemplo, formulando el enunciado con preguntas, con comandos o una afirmación a completar con espacios en blanco.
- **Demostraciones con ejemplos:** Se pueden incluir ejemplos dentro del *prompt* para demostrar el formato deseado de la respuesta. De este modo, el modelo realizará aprendizaje en contexto (ICL) y realizará la tarea con más calidad.
- **Evitar sesgos:** Los *promtps* se deben diseñar evitando los sesgos de los LLMs, utilizando un lenguaje neutro y siendo consciente de las posibles implicaciones éticas y sociales.
- ***Prompts* incrementales:** Para tareas complejas o con diferentes fases, los *prompts* pueden estructurarse para seguir una secuencia de pasos. De esta manera, se ayuda al LLM a descomponer la tarea en enunciados más simples, que realizados uno detrás del otro, favorecen la correcta ejecución de la tarea.

Junto a estos principios, existen determinadas técnicas que pueden mejorar el rendimiento y los resultados de las tareas de los LLMs. En la sección 2.2.1 se explican estas técnicas, y en la sección 2.2.2 se explican otras buenas prácticas adicionales.

### 2.2.1. Técnicas

En este apartado se presentan diferentes técnicas de ingeniería de *prompts* que han demostrado un incremento en la eficiencia de resolución de tareas por parte de LLMs. Según Greyling (2023) y Mayo (2024) las principales técnicas son:

- **Cadena de pensamiento *(Chain-of-Thought (CoT))*:** Se divide un problema complejo en subtareas más pequeñas y sencillas. El modelo utilizará internamente las salidas de las subtareas para resolver la siguiente subtarea hasta llegar al resultado deseado. Ejemplo:

  *"Si actualmente tengo 28 años, ¿cuántos años tendré en 2040?*
  *Para calcular la edad de una persona en el año 2040 si actualmente tiene 28 años, primero determinamos el año actual. Luego, calculamos cuántos años pasarán hasta 2040. Finalmente, sumamos esos años a la edad actual de la persona."*

- **Conocimiento generado *(Generated knowledge)*:** Se pide al modelo que genere información o conocimientos relevantes y que luego los utilice para responder a la pregunta principal. Ejemplo:

*"Primero, describe las últimas tendencias en energías renovables hasta 2024. Luego, con base en esa información, proporciona una propuesta detallada sobre cómo un pequeño país podría implementar estas tendencias para mejorar su autosuficiencia energética."*

- **De menos a más *(Least-to-Most)*:** Se descompone un problema complejo en una serie de subproblemas más simples. Seguidamente se soluciona cada uno de ellos, utilizando las respuestas de los subproblemas anteriormente resueltos. Como ejemplo, si se quiere saber cómo contribuyen las células en un tejido orgánico, el *prompt* sería:

  *"¿Cuáles son los elementos básicos de una célula?*
  *Basado en la respuesta anterior, ¿cómo funcionan estos elementos juntos dentro de una célula?*
  *Considerando el funcionamiento de la célula, ¿cómo contribuye esto al funcionamiento de un tejido orgánico?"*

- **Auto-consistencia *(Self-consistency)*:** Se generan múltiples cadenas de pensamiento (CoT) para una misma pregunta y luego se elige la conclusión más común entre todas. Ejemplo:

  *"Para resolver el problema de matemáticas: 'Si tengo 5 manzanas y regalo 2, ¿cuántas manzanas me quedan?', describe al menos tres formas diferentes de pensar o razonar para llegar a la respuesta correcta, y luego indica cuál sería esa respuesta basándote en el razonamiento más común entre las tres explicaciones."*

- ***Prompting* basado en la complejidad *(Complexity-based prompting)*:** Se generan varias cadenas de pensamiento (CoT) para una misma pregunta y se seleccionan aquellas más largas. Seguidamente, se elige la conclusión más común entre las cadenas complejas. Ejemplo:

  *"Tengo un examen la semana que viene y quiero diseñar un plan de estudio. ¿Cuántos días debería estudiar si el examen requiere unas 30 horas de estudio y, mientras estudio, quiero tener descansos de 10 minutos cada hora? Describe varias formas de diseñar el plan y indica la respuesta basándote en los razonamientos más largos y completos."*

- **Auto-refinamiento *(Self-Refine)*:** Se pide al LLM que resuelva un problema, luego se le pide que critique su solución y finalmente se le indica que resuelva el problema teniendo en cuenta lo anterior. Ejemplo:

  *"Paso 1: ¿Cómo podrías resolver el problema de la contaminación en una gran ciudad?*
  *Paso 2: Evalúa críticamente la solución que acabas de proporcionar.*
  *Paso 3: Vuelve a considerar el problema de la contaminación urbana y tu crítica a la solución, ¿existe una manera de mejorar esa solución?"*

- **Árbol de pensamiento *(Tree-of-Thought (ToT)*:** Se inspira en cómo la mente humana hace tareas complejas a través de ensayo y error. Se combinan varios módulos para guiar al

LLM en una conversación multi-ronda, donde se registra el progreso y se permite la exploración estratégica. Ha demostrado ser útil en tareas como resolución de Sudokus o problemas matemáticos. Ejemplo:

*"Para resolver el siguiente problema de matemáticas, primero identificarás los datos clave y posibles fórmulas a utilizar. Luego, explorarás diferentes métodos de solución, evaluando cada paso para asegurarte de que estás en el camino correcto. Si un método no funciona, retrocede y prueba con una alternativa.*
*Problema: Si un tren sale de Madrid a las 10:00 a.m. a una velocidad de 80 km/h y otro tren sale de Barcelona hacia Madrid a las 11:00 a.m. a una velocidad de 100 km/h, ¿a qué hora y a qué distancia de Madrid se encontrarán?"*

- ***Prompting* de estímulo direccional *(Directional-stimulus prompting)*:** Se incluye una pista o indicación en el *prompt* para guiar al LLM hacia la respuesta deseada. Ejemplo:

  "*Escribe un breve párrafo sobre la importancia de la conservación del medio ambiente, asegurándote de incluir las palabras: sostenibilidad, ecosistema y futuro"*

- **Auto-preguntarse *(Self-Ask)*:** Es una evolución del CoT donde el razonamiento del LLM se muestra explícitamente y también se descomponen las consultas en pequeñas preguntas seguidas. Ejemplo:

  *"Pregunta: Quien vivió mas tiempo Theodor Haecker o Harry Vaughan Watkins?*
  *Se necesitan preguntas de seguimiento: Sí. Sigue el siguiente esquema:*
  *Pregunta de seguimiento:*
  *Respuesta intermedia:*
  *Pregunta de seguimiento:*
  *Respuesta intermedia:*
  *...*
  *Pregunta de seguimiento*
  *Respuesta final: "*

- **Auto-CoT:** Evolución del CoT donde se siguen y se actualizan planes de acción, al mismo tiempo que se recopila información adicional. Ejemplo:

  "*Imagina que eres un organizador de eventos y necesitas planear una conferencia sobre tecnología sostenible. Primero, enumera los pasos lógicos para organizar la conferencia, incluyendo la selección de ponentes, la búsqueda de patrocinadores y la promoción del evento. Luego, genera un plan de acción detallado para cada paso, considerando posibles desafíos y cómo superarlos.*"

- ***Meta-prompting*:** Se pide no sólo la respuesta a una pregunta, sino también que el modelo reflexione sobre su actuación y si debe ajustar instrucciones en base a la reflexión. Ejemplo:

*"Piensa en cómo has respondido preguntas similares en el pasado y considera qué podrías mejorar en esta ocasión. Dada tu comprensión actual y tus capacidades, ¿cuál es la explicación más clara y concisa del Teorema de Pitágoras?"*

- **ReAct, Razonamiento y Acción *(Reasoning and Acting)*:** Se combina la idea del razonamiento y la toma de acciones. De esta manera, el razonamiento permite crear y actualizar planes de acción, mientras que las acciones permiten la recolección de información adicional. Ejemplo:

  "*Estoy planeando un viaje a Japón y necesito ayuda para organizarlo. Primero, razona sobre los pasos necesarios para planificar el viaje, incluyendo la obtención de un visado, la reserva de vuelos y alojamiento, y la preparación de un itinerario. Luego, actúa generando una lista detallada de acciones que debo realizar para cada paso, incluyendo recursos útiles y consejos específicos para un viajero que visita Japón por primera vez.*"

- **Razonamiento simbólico y PAL *(Program-Aided Language Model (PAL))*:** Se pide al modelo que haga un razonamiento simbólico, como distinguir entre diferentes tipos de objetos y colores. Para ello, se le proporciona una representación estructurada (como un diccionario) para organizarse y filtrar la información. Ejemplo:

  *"Tengo tres manzanas, dos perros, cinco naranjas, un gato, cuatro plátanos y un coche. ¿Cuántas frutas tengo? Utiliza una representación estructurada de la información, como un diccionario, para calcular el resultado."*

- ***Prompting* iterativo *(Iterative prompting)*:** Se van diseñando diferentes *prompts* de forma iterativa donde los siguientes utilizan la información de los anteriores. Es el equivalente a tener una conversación y que el modelo vaya asimilando y usando la información de sus respuestas. Ejemplo:

  *"Prompt inicial: "¿Puedes explicarme qué es la fotosíntesis?*
  *Respuesta del modelo: [Explicación de la fotosíntesis]*
  *Prompt iterativo: Ahora, teniendo en cuenta la [Explicación de la fotosíntesis], ¿podrías decirme cómo este proceso afecta al ciclo del carbono en la naturaleza?"*

- **Skeleton-of-Thought:** Se incluye un esqueleto de la respuesta y después se puede llegar a abordar cada punto en paralelo, acelerando el proceso de generación de texto. Ejemplo:

  *"Describe la Segunda Guerra Mundial.*
  *Esqueleto: La Segunda Guerra Mundial fue un conflicto militar que tuvo lugar entre 1939 y 1945 e involucró a muchas de las naciones del mundo.*
  *Expansión de puntos: Desarrolla el inicio, los principales eventos, los líderes y las consecuencias de la Segunda Guerra Mundial."*

- **Razonamiento automático y uso de herramientas *(Automatic Reasoning and Tool Use (ART))*:** Se descompone un problema complejo en pasos más sencillos y se pide al modelo que utilice herramientas para resolver los pasos intermedios. Ejemplo:

  "*Calcular la distancia en kilómetros entre París y Berlín. Primero busca las coordenadas geográficas de ambas ciudades. Luego, utiliza una herramienta de cálculo de distancia para determinar la distancia exacta basándose en estas coordenadas.*"

### 2.2.2. Estrategias y buenas prácticas

A parte de las técnicas de ingeniería de *promts* presentadas en la sección anterior, existen estrategias y buenas prácticas que se utilizan para obtener mejores resultados. OpenAI (2023c) presenta los siguientes consejos:

- Escribir instrucciones claras:
  - Incluir cuantos más detalles mejor.
  - Pedir al modelo que adopte una personalidad (por ejemplo, ser experto en un tema).
  - Usar delimitadores para estructurar las distintas partes del *prompt*.
  - Especificar los pasos necesarios para realizar la tarea.
  - Proporcionar ejemplos.
  - Especificar el formato y longitud de la salida
- Proporcionar información de referencia:
  - Incluir textos de referencia en el *prompt*.
  - Hacer que el modelo responda citando textos de referencias externas.
- Dividir las tareas complejas en subtareas más sencillas.
- Ayudar al modelo a pensar utilizando técnicas:
  - Especificar al modelo que elabore diferentes soluciones antes de apresurarse a dar una respuesta final.
  - Utilizar un diálogo interno o secuencia de consultas para que el modelo razone.
  - Preguntar al modelo si se le pasó por alto algo en intentos anteriores

Por otro lado, según Bsharat et al. (2023), existen otros trucos que pueden ser de utilidad en ciertos casos, por ejemplo, algunos de ellos pueden ser:

- Usar la frase "Piensa paso a paso".

- Utilizar ordenes afirmativas como "Haz...", evitando el lenguaje negativo como "No hagas...".
- Incentivar al modelo a esforzarse, por ejemplo: "Te daré una propina de X € por una solución mejor!".
- Alertar al modelo que puede ser penalizado, por ejemplo: "Serás penalizado si...".
- Especificar la tarea con frases como "Tu tarea es...", o "Debes...".

Finalmente, remarcar que la ingeniería de *prompts* es una rama bastante experimental y no una ciencia exacta, por lo que a menudo requiere de una mezcla de creatividad, intuición y prueba y error para obtener los mejores resultados.

## 2.3. Calidad del dato

En esta sección se presenta el estado del arte sobre la calidad del dato. Primero, se describen diferentes conceptos generales, seguidos de los principios y dimensiones de la calidad del dato. Luego, se explica qué se entiende por mala calidad del dato y las causas de la misma. Después se comenta el impacto que tiene la calidad del dato en las organizaciones, empresas y gobiernos, junto a diferentes soluciones y correcciones que se pueden aplicar.

De acuerdo con McGregor (2021), los datos están por todos lados. Se generan automáticamente en miles de dispositivos y continuamente en el tiempo. Sin embargo, la famosa revolución del dato tiene un problema principal: la calidad del dato.

La calidad del dato es una medida de cómo los datos son aptos y están disponibles en el momento adecuado, para los usuarios apropiados, con el fin de tomar decisiones correctas, realizar operaciones, planificaciones o predicciones y alcanzar un resultado deseado. Esta definición se puede ampliar considerando que los datos de buena calidad son aquellos que sean seguros, legales y procesados de manera justa, correcta y segura (King y Schwarzenbach, 2020).

Actualmente, los datos se consideran un activo de valor que puede traer beneficios a las organizaciones, empresas y gobiernos. Por lo tanto, es fundamental garantizar una alta calidad de los mismos siguiendo los principios y herramientas tecnológicas disponibles.

### 2.3.1. Dimensiones y atributos

Las dimensiones de la calidad del dato son aquellas características con las cuales medimos la calidad del dato. Éstas nos ayudan a evaluar si los datos son suficientemente buenos para ser utilizados o si requieren de correcciones. Según UK Government Data Quality Hub (2021), las dimensiones son:

- **Precisión:** Los datos representan la realidad con exactitud. Por ejemplo datos como nombres o direcciones que existen en la realidad.

- **Completitud:** Los datos para un uso específico están presentes y disponibles.
- **Unicidad:** Cada dato es único y aparece una vez en un conjunto, evitando duplicados.
- **Consistencia:** Los datos no entran en conflicto consigo mismos ni con otros conjuntos de datos.
- **Puntualidad:** La disponibilidad de los datos es cuando se esperan y se necesitan.
- **Validez:** Los datos se ajustan al formato, tipo y rango correctos.

Por otro lado, también existen otras clasificaciones de dimensiones de la calidad del dato. Por ejemplo, Southekal (2023) se basa en el trabajo de DAMA (*Data Management Association*) y define las siguientes dimensiones:


Figura 2.10: Dimensiones de la calidad del dato según Southekal (2023). Tomado de Southekal (2023).

Otra clasificación la encontramos en la propuesta de Buzzelli (2022). Que además de dimensiones, propone el concepto de *atributos*, que describen las características de un dato en concreto, como su tipo, metadatos, propietario, entre otros.

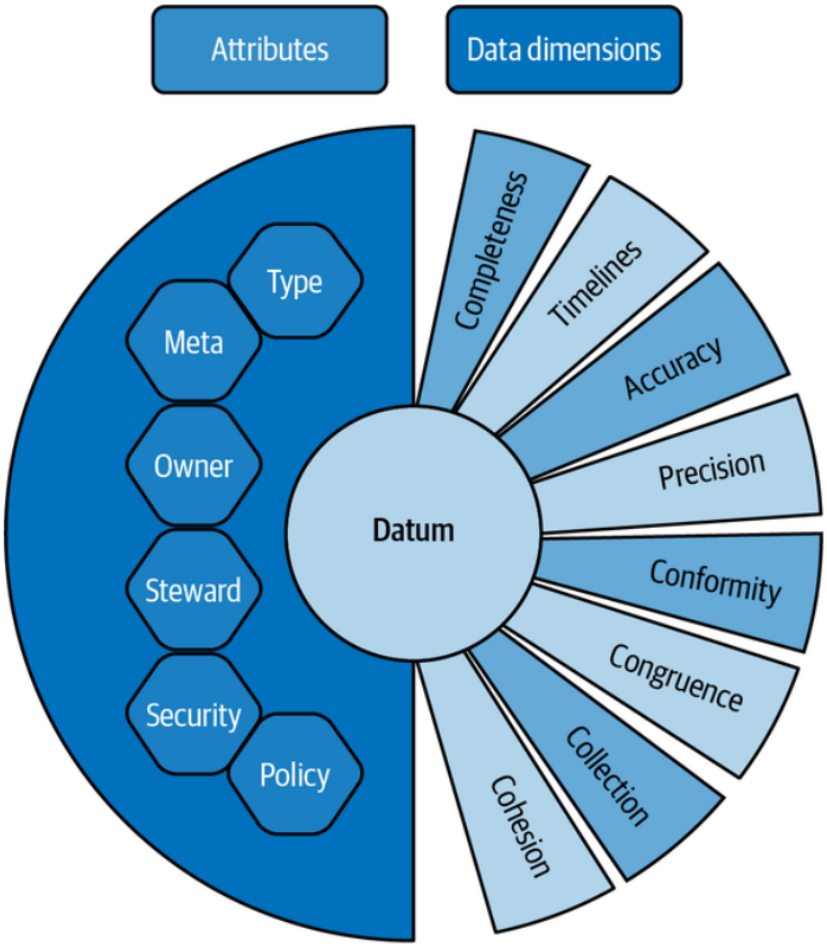


Figura 2.11: Dimensiones y atributos de la calidad del dato según Buzzelli (2022). Tomado de Buzzelli (2022).

### 2.3.2. Mala calidad del dato

De acuerdo con Mertz (2021), George Edward Pelham Box dijo: "*Todos los datos son de mala calidad, y algunos, son útiles*". Esta afirmación nos hace entender que, en la realidad, la regla general es que los datos tengan mala calidad.

La mala calidad del dato se debe a diversos problemas que pueden tener los mismos. Stanley y Schwartz (2024) realizan una clasificación en cuatro tipos, donde cada uno tiene varios problemas:

- **Problemas de tablas:** Afectan a toda la tabla y no son específicos de filas o valores individuales.
    - **Llegadas tardías:** Los datos llegan tarde y no están disponibles en el momento que el consumidor los necesita.
    - **Cambios de esquemas:** Cuando hay cambios estructurales en los datos, columnas nuevas, eliminadas, cambios de nombre o cambios de tipo.

    - **Cambios no trazables:** Cuando los registros de una tabla se actualizan de manera que no se pueden hacer auditorías posteriores.

- **Problemas de filas:** Problemas que afectan a filas enteras dentro de las tablas.
    - **Filas incompletas:** Cuando a determinadas filas les faltan datos por especificar.
    - **Filas duplicadas:** Cuando una misma fila aparece múltiples veces y sólo debería aparecer una vez.
    - **Inconsistencias temporales:** Cuando existen discrepancias o inconsistencias en la secuencia temporal de los registros.
- **Problemas de valores:** Problemas que afectan a valores específicos de los datos.
    - **Valores ausentes:** Cuando no existen datos o aparecen como NULL, ceros, cadenas vacías u otros valores que representan ausencia de datos.
    - **Valores incorrectos:** Cuando los valores no son correctos debido a un problema de codificación o problemas en la creación de los mismos.
    - **Valores inválidos:** Cuando los valores no se ajustan a formatos, restricciones o reglas de negocio.
- **Problemas múltiples:** Problemas sobre como diferentes tablas se relacionan entre si.
    - **Fallos relacionales:** Cuando la integridad relacional entre dos fuentes de datos falla por algunos registros, generando uniones duplicadas o ausentes.
    - **Inconsistencia de fuentes:** Cuando hay diferencias en valores de los datos a lo largo del tiempo o entre fuentes que deberían ser iguales.

Todos estos problemas no son necesariamente independientes, sino que algunos de ellos son las causas de otros y pueden ocurrir al mismo tiempo.

### 2.3.3. Causas de la mala calidad del dato

Muchas de las potenciales causas de la mala calidad del dato estarán relacionadas con la parte del negocio de una organización, empresa o gobierno. Sin embargo, también puede haber problemas técnicos en los sistemas de información por malos diseños o malos usos.

Para resolver estos problemas es importante conocer las causas raíz que los provocan. Tal como indican Southekal (2023) y Hawker (2023), podemos encontrar los siguientes factores:

- Estructuras organizativas aisladas dentro de las organizaciones que provocan gestiones separadas de los datos, creando los llamados silos de datos.

- Interpretación o consumo de datos de diferentes maneras.
- Frecuencia de uso y número de usuarios.
- Casos de uso sin suficiente valor para realizar una generación de datos correcta.
- Búsqueda y recuperación de la información difícil debido a falta de formato, inconsistencias, estándares, falta de gobierno del dato, entre otros.
- Problemas de integración de sistemas y complejidad en las arquitecturas de datos.
- Falta de consciencia del valor de los datos del generador de los mismos.
- Reglas de datos (como longitud, tipo, formato, metadatos, entre otros) que se acaban incumpliendo.
- La disponibilidad del dato en el momento temporal adecuado
- Cambios en el negocio que modifican entidades de datos, relaciones, definiciones, metadatos, estructuras de datos y configuraciones en sistemas.
- Migraciones de datos antiguos o conversiones de bases de datos.
- Cambios en sistemas OLTP *(OnLine Transaction Processing, (OLTP))* y errores de automatización de interfaces.
- Actualizaciones en sistemas de información.
- Errores manuales humanos.
- Diseños de bases de datos pobres con falta de normalización e integridad.
- Mal procesamiento de los datos en proceso que implican limpieza o movimiento de los mismos.
- Falta de cultura de datos en la organización.
- Priorización de la velocidad en los proyectos y beneficios a corto plazo, por encima de un buen gobierno de los datos.
- Unión de datos procedentes de diferentes sistemas de información o fuentes.

### 2.3.4. Calidad del dato en organizaciones, empresas y gobiernos

La calidad del dato en organizaciones, empresas y gobiernos normalmente es un tema que recibe poca atención y no se trata correctamente, a pesar de su gran importancia. En la mayoría de casos, los problemas de calidad del dato no impactan directamente a la supervivencia de la organización y, por eso, el problema no llega a los niveles ejecutivos. Al final, son los técnicos y especialistas quienes realizan correcciones puntuales para presentar datos a los ejecutivos, pero el problema de fondo nunca se acaba solucionando. También, desde los departamentos de negocio tienen otras prioridades, poco tiempo y normalmente existe una desconexión entre negocio y los equipos de datos, que hace que nunca se aborden los problemas de calidad del dato (Hawker, 2023).

Actualmente, las organizaciones están invirtiendo en arquitecturas de datos modernas basadas en la nube que incluyen tecnología y herramientas para garantizar la calidad del dato. Sin embargo, debido a problemas como los mencionados en la sección 2.3.3, las compañías no acaban de materializar la calidad en su información (Stanley y Schwartz, 2024).

Por otro lado, según King y Schwarzenbach (2020), no se tiene una clara visión y organización de la propiedad y responsabilidad de los datos. En muchos casos, desde la generación hasta el consumo de los datos, no existe ningún responsable íntegro de la calidad de los mismos.

#### 2.3.4.1. Impacto de una mala calidad del dato

Conforme a Hawker (2023), cuando la calidad del dato es ignorada en una organización, ésta puede impactar provocando:

- Falta de efectividad, eficiencia o cumplimiento de los procesos de negocio.
- Incapacidad para tomar decisiones de alta calidad basadas en datos.
- Pérdida de ventaja competitiva y la capacidad de la organización de diferenciarse de los competidores.
- Disminución de la reputación de la organización con clientes, proveedores y empleados.
- Imposibilidad de adoptar nuevas tecnologías como la IA.
- Limitaciones en la monetización de productos de datos.
- Incumplimientos de normativas y regulaciones.

A parte de estos factores, la mala calidad del dato drena la productividad de los empleados, que se ven obligados a invertir horas en el entendimiento y corrección de la información, y muchos de ellos acaban dejando de utilizar los datos para su trabajo.

### 2.3.5. Soluciones y correcciones

En esta sección se presentan algunas de las soluciones y acciones correctivas que se realizan para corregir la mala calidad de los datos.

#### 2.3.5.1. Gobierno del dato, datos maestros y datos de referencia

De acuerdo con Eryurek et al. (2021), el gobierno del dato es una pieza clave para garantizar una buena calidad del dato. Éste incluye la organización de la información en dominios, la gestión de metadatos, la asignación de responsabilidades, el control de accesos, la privacidad, la seguridad y la gestión de datos maestros y de referencia.

Los datos maestros son aquellos que se utilizan de manera transversal en una organización y están presentes en múltiples sistemas de información, aplicaciones y procesos. Son esenciales para las operaciones de la compañía y críticos para su negocio. Entre sus características, encontramos que son centralizados, consistentes, cumplen con los criterios de calidad del dato y sirven de base a otros sistemas. Como ejemplo de datos maestros, encontramos: productos, clientes, proveedores, activos, u otra información que sea importante para el negocio.

Así pues, la gestión de datos maestros *(Master Data Management, (MDM))* incluye la definición de arquitecturas tecnológicas, estándares, procesos y políticas para definir y administrar datos maestros (Buzzelli, 2022).

Por otro lado, los datos de referencia son aquellos que se utilizan muy frecuentemente para clasificar o categorizar otros datos. Así pues, los datos de referencia son utilizados por los datos maestros como información estandarizada. Otra característica es que tienen rangos limitados y son muy estables en el tiempo. Como ejemplos, encontramos: códigos de países, nombres de calles, códigos postales, unidades de medida, monedas, categorías de productos, entre otros.

En conclusión, el gobierno del dato y el uso de datos maestros y datos de referencia en los sistemas de información pueden prevenir y ayudar a tener mayor calidad del dato.

#### 2.3.5.2. Reglas de calidad del dato

Las reglas de calidad del dato son procesos y verificaciones lógicas que se aplican a los conjuntos de datos para determinar si tienen suficiente calidad (Hawker, 2023).

Conforme a Stanley y Schwartz (2024), las reglas tienen un alcance, un tipo y, normalmente, un conjunto de restricciones. El alcance define a qué datos se aplica la regla, dónde están los datos y durante qué rango de tiempo. El tipo determina la condición que se quiere comprobar, y las restricciones indican límites fijos que la regla debe respetar o cumplir.

Un ejemplo de regla de calidad sería la verificación de direcciones de correo electrónico. El alcance determinaría que la regla se aplica en la columna 'email' de la tabla 'usuarios' de la base de datos 'clientes'. El tipo es de validación de formato, donde se asegura tener una '@' y terminar en un

dominio. Las restricciones serían que el texto no puede tener espacios y que no debe estar vacío (Stanley y Schwartz, 2024).

#### 2.3.5.3. Perfilado de datos *(data profiling)*

El perfilado de datos *(data profiling)* evalúa un conjunto de datos y determina su estado de calidad, incluyendo información sobre sus valores, niveles de completitud y distribuciones estadísticas sobre cada uno de sus atributos (Hawker, 2023).

Se presenta información sobre: rangos de valores (por ejemplo, mínimos, máximos y cardinalidad), valores ausentes, valores duplicados, valores fuera de rango *(outliers)* y otros aspectos de calidad de datos. A partir de este diagnóstico, se puede tomar acción para corregir los datos necesarios (Eryurek et al., 2021).

#### 2.3.5.4. Modificación de datos

La remediación de datos es la actividad de corregir la calidad de los datos una vez se han identificado los problemas y se han priorizado los más importantes a solucionar. Se incluyen procesos de limpieza de datos, transformaciones, estandarizaciones y de aumento de datos. En algunos contextos, también se le llama *data wrangling*, sin embargo, ésta última se basa más en transformar datos crudos en datos de valor explotables. Por otro lado, cuando realizamos remediación de datos, se recomienda gestionar los problemas de calidad prioritariamente en las fuentes origen (Hawker, 2023), (McGregor, 2021).

De acuerdo con Moses et al. (2022), Southekal (2023) y McGregor (2021), la remediación de datos incluye tareas como:

- Corrección de valores y formatos incorrectos.
- Eliminación, filtrado e imputación de datos ausentes.
- Eliminación de duplicados.
- Aplicación de funciones o mapeados.
- Renombrado de índices o atributos.
- Discretizaciones o agrupamientos.
- Detección y tratamiento de valores fuera de rango *(outliers)*.
- Normalizaciones y escalados.
- Conversiones de formatos temporales.
- Permutaciones y muestreos aleatorios.

- Creación de variables *"dummy"*.
- Manipulación y normalización de texto.
- Aplicación de expresiones regulares.
- Estandarización de datos a partir de datos maestros y de referencia.
- Aumento de datos a partir de importación o generación de datos adicionales.

Por otro lado, Hellerstein et al. (2017), propone una clasificación de tareas en tres grupos: las de estructuración, que cambian la forma o el esquema de los datos; las de enriquecimiento, que añaden nuevos valores a los datos; y las de limpieza, que corrigen irregularidades en el conjunto de datos.

La mayoría de tareas de remediación de datos son manuales, aun así se pueden crear *pipelines* de automatización que apliquen ciertas acciones (Southekal, 2023).

#### 2.3.5.5. Monitorización de métricas

Otra de la soluciones para mejorar la calidad del dato es la monitorización de métricas. Consiste en el proceso de revisar el estado de calidad en los diferentes conjuntos de datos de una organización a través de cuadros de mando e indicadores que agregan los resultados de las reglas de calidad vistas en la sección 2.3.5.2 (Hawker, 2023).

Según Stanley y Schwartz (2024), una estrategia de monitorización debe ser capaz de detectar los problemas de calidad del dato, alertar a los perfiles indicados en el momento adecuado, permitir resolver rápidamente los problemas y escalar eficientemente al aumentar el volumen de datos a monitorizar.

## 2.4. LLMs aplicados a la calidad del dato

Como se ha visto en la sección 2.1, los LLMs han revolucionado las capacidades de procesamiento, entendimiento y generación del lenguaje. Aprovechando estas nuevas habilidades, en los últimos años, se han publicado diferentes estudios que utilizan estos modelos en tareas relacionadas con la calidad de los datos.

En enero de 2022, cuando el modelo más destacado era GPT-3, Jaimovitch-López et al. (2022) ya planteaban que los LLMs presentaban potencial debido a que son capaces de inferir resultados utilizando muy pocos ejemplos, integran grandes capacidades de conocimiento y pueden realizar recuperación de la información. Sin embargo, también destacaban que tenían problemas de fiabilidad.

Unos meses más tarde, Narayan et al. (2022) también investigaron el rendimiento de LLMs en tareas de limpieza e integración, encontrado buenos resultados en varias tareas utilizando *few-shot learning*.

Más recientemente, en agosto de 2023, Zhang et al. (2023) estudiaron diferentes tareas de calidad del dato, demostrando su potencial, pero a la vez alertando de sus limitaciones, que principalmente eran ineficiencias y costes de computación.

### 2.4.1. Potenciales tareas

Conforme a los siguientes autores: Narayan et al. (2022), Jaimovitch-López et al. (2022), Frąckiewicz (2023), Menthière (2023) y Pai (2023), entre las tareas de calidad del dato que podrían llegar a resolver los LLMs, encontramos:

- Estandarización de datos.
- Cambios de formato o sintaxis.
- Clasificación de valores en categorías.
- Detección de inconsistencias.
- Imputación de datos ausentes.
- Detección y corrección de errores en valores.
- Identificación y corrección de discrepancias entre fuentes.
- Validación de datos con datos maestros o de referencia.
- Enriquecimiento y aumento de datos.
- Detección y tratamiento de duplicados.
- Descubrimiento y tratamiento de valores fuera de rango (*outliers*).
- Tareas de etiquetado de datos.
- Aplicación de reglas de calidad.

Además de todas estas tareas, los LLMs más avanzados como GPT-4 y GPT-4 Turbo tienen capacidad de programación y ejecución de código Python. Por lo tanto podrían incluso hacer el perfilado de los datos (*data profiling*) visto en la sección 2.3.5.3.

### 2.4.2. Limitaciones

Hasta ahora, parece que el potencial de los LLMs para la calidad del dato es muy prometedor. Sin embargo, actualmente existen ciertas limitaciones que pueden hacer que no sea viable una implementación real de esta tecnología.

De acuerdo con Jaimovitch-López et al. (2022) y Menthière (2023), existe un problema muy serio que todavía no se ha resuelto: la fiabilidad. La implementación de un LLM puede resultar en un sistema no determinista que cometa errores esporádicos. Así pues, en ciertos sectores, este hecho puede ser inaceptable. Además, en Narayan et al. (2022), se encontró que para datos privados o de un dominio muy específico también surgían limitaciones de efectividad.

Finalmente, según Zhang et al. (2023) el coste de computación puede llegar a ser muy elevado dependiendo de la tarea de calidad a realizar y el volumen de datos que hay que tratar. Ésto acabaría trayendo problemas de escalabilidad a largo plazo (Fernandez et al., 2023).

### 2.4.3. Casos de uso

En esta última sección, se presenta un listado de estudios que han explorado la aplicación de LLMs en calidad del dato:

- **Jaimovitch-López et al. (2022):** Estudiaron la realización de varias tareas de *data wrangling* con GPT-3 y Google BIG-G. Encontraron que los modelos pueden ser muy potentes y flexibles para ciertas tareas, pero presentaban problemas de fiabilidad.
- **Narayan et al. (2022):** Evaluaron a GPT-3 con cinco tareas de limpieza e integración de datos. Encontraron que el rendimiento obtenido se asimilaba a otros métodos clásicos. También encontraron limitaciones al utilizar datos privados o de un dominio muy específico.
- **Hegselmann et al. (2023):** Exploraron la clasificación de datos tabulares mediante los modelos de GPT-3 y T0, con la técnica de *few-shot learning*. Observaron que el rendimiento superaba a antiguas técnicas de aprendizaje automático o profundo.
- **N. Li et al. (2023):** Desarrollaron una API RESTful para la extracción y estandarización de habilidades de descripciones de trabajo y de perfiles profesionales. Se basaron en el modelo de LLaMA, sin embargo, no evaluaron la eficacia de esta aplicación.
- **Arnes y Horsch (2023):** Investigaron la posibilidad de corregir valores y estructuras de datos para cumplir con reglas de calidad. Se generaban datos que seguían esquemas estándar y se evaluaron GPT-3, GPT-3.5 y GPT-4. Al final, obtuvieron buenos resultados sólo con los dos últimos modelos.

- **Ram (2023):** Utilizó GPT-3.5 Turbo en un encuesta para clasificar respuestas abiertas en diferentes categorías. El resultado fue una estandarización de las múltiples respuestas abiertas en unos grupos que le permitieron realizar analítica de datos posteriormente.
- **Sharma et al. (2023):** Exploraron la generación de código SQL con GPT-3.5 Turbo para consultar y transformar datos de fuentes orígenes hasta obtener conjuntos de datos explotables. Obtuvieron una precisión del 96 % realizando 105 tareas.
- **Chen et al. (2023):** Desarrollaron un sistema para crear aplicaciones de gestión de datos utilizando LLMs e hicieron experimentos de imputación de datos, clasificación de temas y generación de consultas SQL.
- **Peeters y Bizer (2023):** Utilizaron GPT-3.5, GPT-4 y LLaMA 2 para encontrar coincidencias de entidades *(entity matching)*, que consiste en determinar si dos entidades hacen referencia a la misma entidad del mundo real. Llevaron a cabo 5 pruebas y GPT-4 obtuvo resultados de entre 76 % y 95 % de *F1 score*.
- **Bolding et al. (2023):** Estudiaron la capacidad de GPT-3.5 para limpiar textos y eliminar diferentes tipos de ruido, por ejemplo eliminación de *emojis* manteniendo el significado semántico o la corrección de jergas a lenguaje estandarizado. Encontraron mejoras en la calidad del texto, pero también limitaciones como la robustez respecto a sesgos o diferencias entre idiomas.

# Capítulo 3

# Metodología

Tal y como se ha introducido en el Capítulo 1, con el objetivo de demostrar si los LLMs son capaces de realizar tareas de calidad del dato, se ha diseñado una metodología para realizar experimentos prácticos que demuestren el potencial de los LLM en tareas de corrección y estandarización de datos.

Así pues, este capítulo presenta la metodología que se ha utilizado para realizar la parte experimental de la investigación. La sección 3.1 expone las fuentes de datos utilizadas, la sección 3.2 presenta el entorno de desarrollo y las tecnologías, la sección 3.4 presenta los costes asociados al uso de diferentes recursos, y la sección 3.5 muestra como se ha llevado a cabo el diseño de los experimentos.

## 3.1. Fuentes de datos

Para realizar los experimentos, se utilizan, por un lado, datos de referencia reales y, por otro, datos transaccionales ficticios generados especialmente para cada tarea. Respecto a los primeros, se han empleado datos de referencia de la Generalitat de Catalunya (Generalitat de Catalunya, 2023), la cual los pone a disposición a través de una plataforma de datos abiertos (Generalitat de Catalunya, 2024) donde publica diferentes datos de interés para el público general. Además, se incluyen otros datos que se utilizan en los sistemas de información públicos.

Respecto a la obtención de los datos, la plataforma dispone de una API que permite importar los datos directamente en el código. Así pues, para este proyecto se ha utilizado esta API para acceder a los datos: https://dev.socrata.com/docs/endpoints.

A continuación, se presentan los conjuntos de datos de referencia que se han seleccionado para realizar los experimentos.

### 3.1.1. Municipios de Catalunya

Se trata del listado oficial de municipios de Catalunya publicado por el Institut d'Estadística de Catalunya (IDESCAT) y Institut Cartogràfic i Geològic de Catalunya (ICGC) (2023). Se ha utilizado

la última actualización de 28 de diciembre de 2023, y los metadatos del conjunto de datos se muestran en la Tabla 3.1.

| Nombre de la columna | Descripción | Tipo de dato |
|---|---|---|
| Codi | Identificador único del municipio. | Texto |
| Nom | Nombre oficial del municipio. | Texto |
| Codi comarca | Identificador único de la comarca. | Texto |
| Nom comarca | Nombre oficial de la comarca. | Texto |
| UTM X | Valor de coordenada X del punto de georeferencia. | Número |
| UTM Y | Valor de coordenada Y del punto de georeferencia. | Número |
| Longitud | Longitud en formato decimal. | Número |
| Latitud | Latitud en formato decimal. | Número |
| Georeferència | Punto de georeferencia. | Punto geográfico |

Tabla 3.1: Metadatos del conjunto de datos de referencia de municipios de Catalunya. Tomado de Institut d'Estadística de Catalunya (IDESCAT) y Institut Cartogràfic i Geològic de Catalunya (ICGC) (2023).

La columna *"Nom"* presenta el nombre oficial de todos los municipios de Catalunya. Así pues, se ha utilizado esta variable como dato de referencia real para hacer los experimentos.

### 3.1.2. Calles de Catalunya

Este conjunto incluye el listado oficial de calles de Catalunya estandarizado y publicado por el Institut Cartogràfic i Geològic de Catalunya (ICGC) (2023). Se usa la última actualización del 28 de diciembre de 2023 y los metadatos del conjunto de datos se muestran en la Tabla 3.2. sintéticos.

| Nombre de la columna | Descripción | Tipo de dato |
|---|---|---|
| Codi Unitat Poblacional | Código de la unidad poblacional de Catalunya. | Texto |
| Codi Tipus Via | Código del tipo de vía. | Texto |
| Nexe Via | Información anexa al nombre de la vía. | Texto |
| Nom Via | Nombre de la vía. | Texto |
| Nom Compost Via | Nombre entero de la vía, formado a partir de los campos anteriores. | Texto |
| Data | Fecha en que se registró la vía. | Texto |
| Nom Municipi | Nombre del municipio donde se encuentra la vía. | Texto |
| Nom Unitat Poblacional | Nombre de la unidad poblacional donde se encuentra la vía. | Texto |
| Tipus Via | Nombre del tipo de vía. | Texto |

Tabla 3.2: Metadatos del conjunto de datos de referencia de calles de Catalunya. Tomado de Institut Cartogràfic i Geològic de Catalunya (ICGC) (2023).

Así pues, este conjunto será la referencia para la generación de datos transaccionales sintéticos.

## 3.2. Entorno de desarrollo

En esta sección se exponen las tecnologías que se han utilizado para desarrollar el código. Concretamente se ha utilizado el lenguaje de programación Python junto con librerías y APIs.

El código se ha programado en el entorno de desarrollo de Visual Studio Code, el cual permite el uso de múltiples lenguajes de programación, incluyendo Python. Además, se pueden utilizar entornos virtuales que contengan todas las librerías necesarias para el proyecto. Gracias a su interfaz intuitiva y personalización mediante extensiones ha proporcionado una buena experiencia de usuario.

## 3.3. Librerías y APIs

Las librerías y APIs de Python utilizadas han sido las siguientes:

- **Pandas (pandas):** Para la importación, manipulación y transformación de datos tabulares.
- **Randint (randint):** Para la generación de números y variables aleatorias.
- **Time (time):** Para monitorizar el tiempo de ejecución.
- **Sócrata (sodapy):** Para la importación de datos abiertos de la Generalitat de Catalunya, concretamente, para la obtención de los datos de referencia.
- **OpenAI (openai):** Incluye el cliente para llamar a la API de inferencia de OpenAI.
- **Google Vertex AI (vertexai):** Incluye el cliente para llamar a la API de inferencia de Google Gemini Pro.
- **Hugging Face (huggingface_hub):** Incluye el cliente para llamar a la API de inferencia de Hugging Face, que permite el uso de diferentes modelos.
- **Scikit-Learn (sklearn):** Para evaluar los modelos con métricas de aprendizaje automático.

## 3.4. Costes y recursos disponibles

El desarrollo y puesta en producción de LLMs para realizar experimentos de inferencia es una tarea compleja que requiere de muchos recursos económicos y computacionales. Se consideró utilizar modelos en local, pero se vió que no se disponía del *hardware* necesario para hacerlo. La falta de GPUs avanzadas hizo que se descartara la ejecución de LLMs en resursos locales.

Por otro lado, se planteó el uso de las infraestructuras de diferentes *clouds* como Google, Microsoft Azure o Amazon Web Services. Sin embargo, se encontró un coste económico demasiado alto para poner LLMs en producción y realizar inferencias.

La solución que se empleó fue el uso de APIs de empresas como OpenAI o Google, que representaban unos costes más comedidos. Además se ha utilizado la API de Hugging Face Pro, que con un coste reducido, permite una inferencia bastante eficiente utilizando su infraestructura.

La Tabla 3.3 muestra los costes asociados al uso de estas herramientas.

| API | LLM | Coste |
|---|---|---|
| OpenAI API | GPT-4 Turbo *(gpt-4-1106-preview)* | Entradas: 0,01 $ / 1K tokens.<br>Salidas: 0,03 $ / 1K tokens. |
| | GPT-3.5 Turbo *(gpt-3.5-turbo-0125)* | Entradas: 0,0005 $ / 1K tokens.<br>Salidas: 0,0015 $ / 1K tokens. |
| Google Vertex AI API | Gemini Pro | Hasta 60 consultas por minuto: Sin coste.<br>Más de 60 consultas por minuto:<br>Entradas: 0,00025 $ / 1K tokens.<br>Salidas: 0,0005 $ / 1K tokens. |
| Hugging Face Pro API | LLMs de código abierto | 9 $ / mes |

Tabla 3.3: Costes económicos de las APIs para realizar inferencias con LLMs.

## 3.5. Diseño de los experimentos

Esta sección expone el procedimiento que se ha llevado a cabo para diseñar los experimentos que validarán la viabilidad de aplicar LLMs para la calidad del dato.

En primer lugar, se decidieron qué tareas de calidad del dato se querían estudiar. Concretamente, se seleccionaron tres experimentos, que representaban tres tareas concretas. El primer experimento consiste en una tarea simple de corrección de datos, el segundo una estandarización en base a datos de referencia y el tercero, una imputación de valores ausentes en un conjunto de datos en base a datos de referencia. Los detalles de los experimentos se presentan en el Capítulo 4.

El Capítulo 5 expone los LLMs que se han evaluado junto con sus características y el Capítulo 6 muestra las métricas de evaluación que se han implementado para valorar el desempeño de los diferentes modelos.

# Capítulo 4

# Experimentos

Con el objetivo de evaluar el desempeño de los LLMs en tareas de calidad del dato, se han diseñado tres experimentos:

- **Experimento 1 - Corrección de valores:** Se evaluará la capacidad que tiene un LLM de corregir un atributo de datos transaccionales en base a datos de referencia.
- **Experimento 2 - Estandarización:** Se probará si un LLM es capaz de estandarizar los campos y valores de unos datos transaccionales que no tienen un formato adecuado.
- **Experimento 3 - Imputación:** Se evaluará la habilidad de imputar datos ausentes en un conjunto de datos basándose en datos maestros.

Las siguientes secciones explican con más detalle cada experimento y sus fases.

## 4.1. Experimento 1 - Corrección de valores

En muchas ocasiones, los datos transaccionales pueden contener atributos que se generan con campos abiertos donde el usuario introduce un valor manualmente. Este hecho provoca que siempre haya algún error humano que se propaga hacia el sistema de información y su base de datos. El resultado es que los datos almacenados contienen problemas e inconsistencias en un determinado campo.

El objetivo de este experimento es comprobar si un LLM es capaz de corregir este tipo de errores en un atributo concreto. Para ello tendrá que modificar unos datos transaccionales en base a unos datos de referencia.

Concretamente, los datos de referencia utilizados han sido el listado de municipios de Catalunya, presentados en la sección 3.1.1. Así pues, en este caso, el LLM tendrá que corregir nombres de municipios mal escritos.

Para realizar la tarea lo más automatizada posible se ha desarrollado un flujo de trabajo o *pipeline* para utilizar diferentes LLMs, que va desde la importación de datos hasta la evaluación de la tarea. La Figura 4.1 muestra este diagrama de flujo que se ha seguido para desarrollar el código.

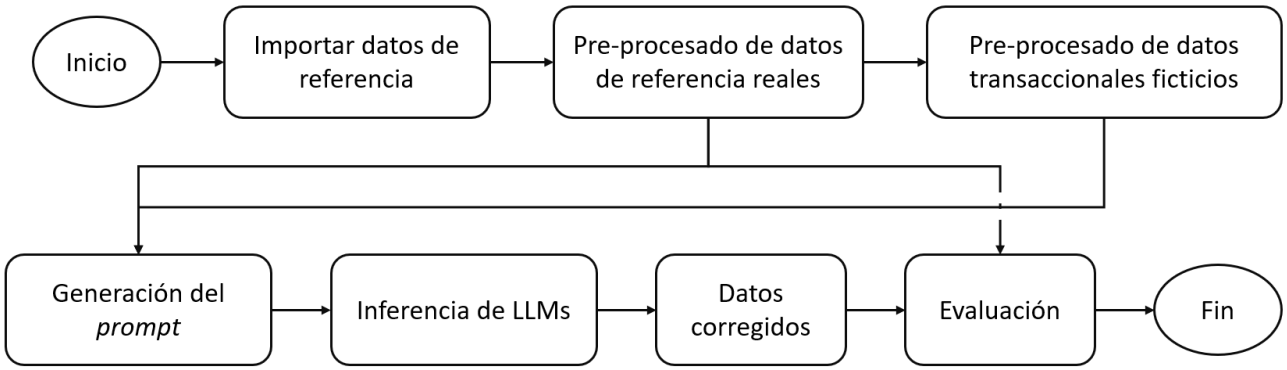


Figura 4.1: Diagrama de flujo del experimento 1.

El código de Python se adjunta en el Anexo B.1 y a continuación se explican los pasos concretos del proceso.

### 4.1.1. Importación de librerías y de datos

El primer paso del flujo de trabajo es la importación de librerías. En particular, se importan las librerías descritas en la sección 3.3. Además, respecto a la librería *random*, se fija la semilla con el comando `random.seed(1)` para garantizar la reproducibilidad en la generación de los datos y la ejecución del código.

Seguidamente, se configura el cliente de la API de Sócrata para acceder a los datos abiertos de la Generalitat de Catalunya (ver sección 3.1). Una vez configurada, podemos obtener los datos de municipios en formato JSON. Sin embargo, estos datos tendrán que ser pre-procesados y adaptados para realizar la tarea.

### 4.1.2. Pre-procesado de datos de referencia reales

El segundo paso consiste en preparar los datos de referencia que utilizará el modelo. Partiendo del JSON con los datos del municipio, se transforman los datos en un *dataframe* de *pandas* y se selecciona la columna *Nom*, es decir, el nombre oficial de los municipios de Catalunya. La Tabla 4.1 muestra una impresión del *dataframe* resultante.

Llegados a este punto, se planteó utilizar todos los municipios para realizar el experimento. Sin embargo, al pasarlos a formato texto para generar un *prompt*, se vio que eran demasiados *tokens*. Utilizando la herramienta de *Tokenización* de OpenAI (2023d), se obtuvieron 5,309 tokens.

Por otro lado, para poder comparar correctamente los modelos, todos ellos debían utilizar la misma ventana de contexto (ver sección 2.2). Así pues, teniendo en cuenta los modelos a utilizar

| | Nom |
|---|---|
| 0 | Salàs de Pallars |
| 1 | Vilabertran |
| 2 | Artesa de Lleida |
| 3 | Tarroja de Segarra |
| 4 | Riumors |
| ... | ... |
| 944 | Ivorra |
| 945 | Sant Martí de Llémena |
| 946 | Ciutadilla |
| 947 | Vendrell, el |
| 948 | Bellcaire d'Empordà |

Tabla 4.1: *Dataframe* de nombres oficiales de municipios de Catalunya.

(ver Capítulo 5), la máxima ventana de contexto era de 4096 *tokens*. Con este límite, se tuvo que reducir el número de municipios a incluir en los datos de referencia.

En consecuencia, los datos de referencia a utilizar en el experimento 1 se generaron a partir de una muestra de los 948 municipios de Catalunya.

### 4.1.3. Pre-procesado de datos transaccionales ficticios

El siguiente paso fue el pre-procesado de los datos transaccionales ficticios. Para generarlos, se seleccionó una muestra aleatoria con valores repetidos de los datos de referencia del experimento uno.

A continuación, se debían introducir errores e inconsistencias en los datos. Para ello se programó una función que producía errores aleatorios de diferentes tipos. Los tipos de errores se escogieron y se diseñaron basándose en los errores más comunes que se pueden encontrar en datos de texto introducido manualmente. Concretamente se introdujeron errores al 70 % de los datos transaccionales, y un 30 % quedaba con valores correctos. De esta manera se podía estudiar la corrección de datos erróneos y la detección y no alteración de los datos correctos. Así pues, cuando entraba un valor en la función, se le aplicaba un error aleatorio de la siguiente lista:

- **Sustitución:** Se sustituye una letra por otra del abecedario de maneara aleatoria.
- **Eliminación:** Se elimina una letra de la palabra de forma aleatoria.
- **Adición:** Se añade una letra aleatoriamente dentro de la palabra.
- **Puntuación:** Se añade un símbolo de puntuación (', . ! ¿ ?') de forma aleatoria en la palabra.
- **Repetición:** Se repite una letra de la palabra aleatoriamente.

- **Espaciado:** Se añade un espacio en un lugar aleatorio de la palabra.
- **Mezcla de mayúsculas:** Se convierten a mayúsculas un número aleatorio de letras.
- **Símbolos:** Se convierten algunas letras en símbolos ('@ 3 ! 0 $').
- **Fonético:** Se generan errores fonéticos de ciertas letras ('c k s z').
- **Abreviación:** Si el dato tiene más de dos palabras, se abrevia una de ellas con la primera letra y un punto.
- **Mezcla de palabras:** Si el dato tiene más de una palabra, se mezclan aleatoriamente las mismas.

Por lo tanto, al aplicarse aleatoriamente, la proporción de cada error era aproximadamente la misma. Para ver el detalle de como se programó y el código, ver el Anexo B.1. Así pues, se aplicó esta función al 70 % de los valores y se obtuvieron los datos transaccionales ficticios a utilizar. Posteriormente, se creó una columna llamada *error*, que indicaba si el dato transaccional contenía un error (con un 1) o no lo contenía (con un 0). El resultado del *dataframe* de datos transaccionales ficticios fue el presentado en la Tabla 4.2.

| | datos_referencia | datos_transaccionales | error |
|---|---|---|---|
| 0 | Ponts | Pots | 1 |
| 1 | Sant Vicenç de Castellet | $@nt v!c3nç d3 c@$t3ll3t | 1 |
| 2 | Verdú | VErDú | 1 |
| 3 | Canejan | Canejon | 1 |
| 4 | Granyena de Segarra | Granyena de Segarr,a | 1 |
| ... | ... | ... | ... |
| 195 | Deltebre | deltebre | 1 |
| 196 | Pobla de Massaluca, la | Pobla de Massaluca, la | 0 |
| 197 | Roca del Vallès, la | Roca del Vallès, la | 0 |
| 198 | Tarrés | Tarré s | 1 |
| 199 | Deltebre | Deltebre | 0 |

Tabla 4.2: Datos transaccionales ficticios del experimento 1.

### 4.1.4. Generación del *Prompt*

El siguiente paso es la generación del *prompt*. Para ello, se han seguido las técnicas de ingeniería de *prompts* vistas en la sección 2.2 y se ha estructurado de la siguiente manera:

- **Presentación:** Se asigna un rol al LLM y se le explican los datos y la tarea de calidad del dato que debe realizar. Además se incluyen aclaraciones para guiar al modelo.

- **Respuesta esperada:** Se indica el formato y características de la respuesta esperada.
- **Instrucciones importantes:** Se informa de mensajes importantes que tiene que cumplir el modelo.
- **Datos de referencia:** Se presentan los datos de referencia.
- **Datos transaccionales a corregir:** Se muestran los datos a corregir en un formato específico.
- **Ejemplo de respuesta:** Se enseña al modelo un ejemplo de cómo debe ser la respuesta.
- **Frase para tomar acción**: Se introduce una última frase para guiar al modelo en la respuesta.

El *prompt* resultante para el experimento 1 se muestra a continuación. Nótese que se han acortado los datos de referencia y los datos transaccionales para un mayor entendimiento del mismo.

***Prompt:***

*Eres un experto en corrección de datos, especializado en la detección y estandarización de errores en los datos de municipios de Cataluña, España. Además, eres experto en la alineación de datos transaccionales con datos de referencia. Tengo dos conjuntos de datos en formato CSV con delimitador '; '. El primero incluye datos de referencia con el formato correcto de municipios de Cataluña, España. El segundo contiene datos transaccionales de municipios que contienen errores. Los errores pueden ser de diferentes tipos, como un mal formato. Tu tarea es corregir meticulosamente los datos transaccionales, basándote en los datos de referencia, de manera que tengan el formato correcto. Aplica tu conocimiento para alinear y estandarizar los datos transaccionales con precisión. Asegúrate de que los valores que ya están estandarizados queden inalterados. No utilices herramientas de análisis ni Python, solo tu conocimiento.*

*Tu respuesta debe contener todos los datos transaccionales, tanto los correctos como los corregidos. El formato de la respuesta debe ser un CSV con delimitador '; ' con dos columnas llamadas: Dato transaccional y Dato corregido. Incluye la cabecera en la respuesta. Cabecera:*

*Dato transaccional; Dato corregido*

*MUY IMPORTANTE: Sólo incluye el CSV en tu respuesta. No incluyas espacios entre cada fila. No incluyas ninguna palabra adicional en tu respuesta. No incluyas explicaciones, informaciones, razonamientos o pasos seguidos. Asegúrate que el número de valores de la respuesta es el mismo que el número de valores de datos transaccionales. No dupliques, ni te dejes valores.*

*Datos de referencia:*
*Penelles; Linyola; Preixens; Corbera de Llobregat; Vall d'en Bas, la; Figaró-Montmany; Miravet; Godall; Sant Ramon; Canejan; Celrà; Granja d'Escarp, la; Sant Vicenç de Castellet; Sant Pere de Ribes; Sabadell; Arsèguel; Albatàrrec; Masies de Roda, les; Caseres; Almacelles; Espolla; Bossòst; Vielha e Mijaran; Fontanilles; Sant Joan Despí; Martorelles; Santa Eulàlia de Riuprimer; Botarell; ...*

*Archivo CSV a completar:*
*Dato transaccional; Dato corregido*
*Pots;*
*$@nt v!c3nç d3 c@$t3ll3t;*
*VErDú;*
*Canejon;*
*Granyena de Segarr,a;*
*Figaró-Montmany;*
*Arbolr;*
*CaBAcéS;*
*Albatàrrec;*
*l!ny0l@;*
*Cornellà de Llobregat;*
*Arres;*
*...*

*Ejemplo de respuesta:*
*Dato transaccional; Dato corregido*
*C. de Llobregat; Corbera de Llobregat*
*Gudall; Godall*
*Ponntss; Ponts*

*Archivo CSV respuesta:*

### 4.1.5. Configuración de las APIs de LLMs

Una vez se tuvo el *prompt* generado, se realizó la configuración de las APIs de LLMs. Cada API requiere de una identificación que se lleva a cabo con llaves de acceso. En caso que el uso del LLM tenga un coste, se asocia la llave y su uso para repercutir los costes.

### 4.1.6. Inferencia

El siguiente paso fue la inferencia, que se refiere a la generación de la respuesta por parte del LLM. Para ello, se siguieron las documentaciones de las APIs de OpenAI, Google Gemini y Hugging Face. A cada modelo, se le enviaba el *prompt* de la sección 4.1.4 y se recibía la respuesta con los datos corregidos. Además, también se midió el tiempo de ejecución de la inferencia para comparar a los modelos.

### 4.1.7. Post-procesado de datos

Las respuesta obtenidas se encontraban en formato texto, por lo que se debía hacer un post-procesado de los datos corregidos. Las respuestas tenían el siguiente formato:

*Dato transaccional; Dato corregido*
*C. de Llobregat; Corbera de Llobregat*
*Gudall; Godall*
*Ponntss; Ponts*
*B@dalona; Badalona*
*...*

Por consiguiente, se realizó un procesado del texto con una separación de líneas y la creación de una columna adicional para cada modelo en el *dataframe* de datos transaccionales. De esta manera los datos quedaban estructurados para evaluarlos.

### 4.1.8. Evaluación e impresión de resultados

El último paso fue la evaluación de resultados siguiendo las métricas presentadas en el Capítulo 6. En particular, se calculó la calidad inicial de los datos transaccionales y, después, para cada modelo, la calidad final, el incremento de calidad, la exactitud con la cual se corrigieron los datos erróneos y se detectaron los datos correctos, y el tiempo de ejecución de la inferencia.

## 4.2. Experimento 2 - Estandarización

Existen casos en los que los datos transaccionales no solo contienen errores, sino que no tienen la estructura y los campos correctos como marcan los datos de referencia. El objetivo de este segundo experimento es ver si un LLM es capaz de corregir datos transaccionales y, además, cambiar su estructura para adecuarla a los campos que se requieren según los datos de referencia.

En particular, los datos de referencia utilizados han sido las calles de Catalunya, presentadas en la sección 3.1.2. El flujo de trabajo o *pipeline* es muy parecido al del experimento 1 (Figura 4.2) y

el código Python se adjunta en el Anexo B.2.

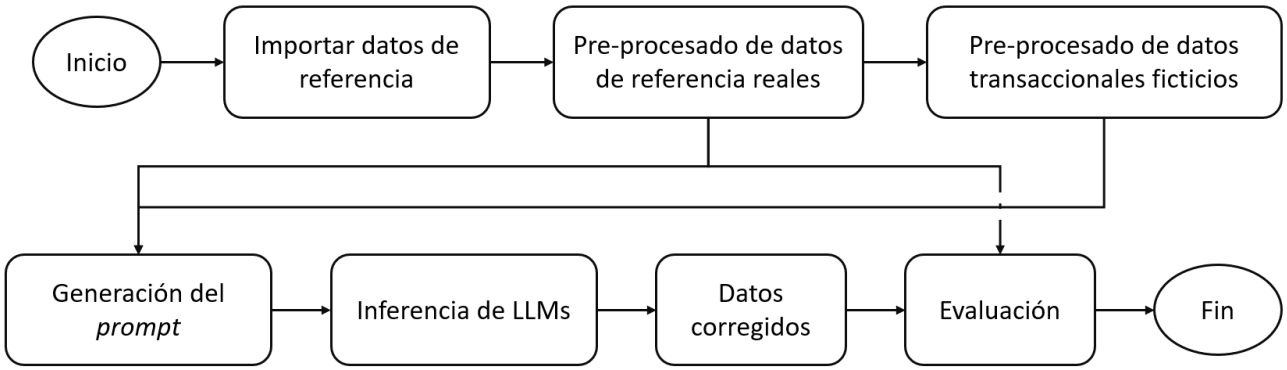


Figura 4.2: Diagrama de flujo del experimento 2.

Debido a que hay pasos que se repiten con el experimento 1, a continuación sólo se exponen las fases en las que hay diferencias con el primer experimento. Estas son: la importación de datos de referencia, el pre-procesado de datos de referencia reales, el pre-procesado de datos transaccionales ficticios, la generación del *prompt*, el post-procesado de datos y la evaluación.

### 4.2.1. Importación de datos de referencia

De manera similar al experimento 1, se utiliza la API de Sócrata para importar los datos de referencia de las calles de Catalunya (ver sección 3.1.2). Una vez importados en formato JSON, se transforman a un *dataframe* que se muestra en la Figura 4.3.

| | nom_compost_via | codi_unitat_poblacional | codi_tipus_via | nom_via | nexe_via | data | nom_municipi | nom_unitat_poblacional | tipus_via |
|---|---|---|---|---|---|---|---|---|---|
| 0 | Camí de Can Socarrats | 0800180013016 | 013 | Can Socarrats | de | 20221117 | Abrera | Can Socarrats i Can Moragues | Camí |
| 1 | Carrer de Can Moragues | 0800180013016 | 016 | Can Moragues | de | 20221117 | Abrera | Can Socarrats i Can Moragues | Carrer |
| 2 | Carrer del Metall | 0800180013016 | 016 | Metall | del | 20221117 | Abrera | Can Socarrats i Can Moragues | Carrer |
| 3 | Carretera d'Ullastrell | 0800180013016 | 021 | Ullastrell | d' | 20221117 | Abrera | Can Socarrats i Can Moragues | Carretera |
| 4 | No consta | 0800180013016 | 998 | NaN | NaN | 20231025 | Abrera | Can Socarrats i Can Moragues | No consta |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 114148 | Travessia de la Pineda | 4390760001017 | 083 | Pineda | de la | 20220425 | Canonja, la | Canonja, la | Travessia |
| 114149 | No consta | 4390760001017 | 998 | NaN | NaN | 20231025 | Canonja, la | Canonja, la | No consta |
| 114150 | Altres/Diversos | 4390760001017 | 999 | NaN | NaN | 20231025 | Canonja, la | Canonja, la | Altres/Diversos |
| 114151 | No consta | 9899899999999 | 998 | NaN | NaN | 20231025 | No consta | No consta | No consta |
| 114152 | Altres/Diversos | 9999999999999 | 999 | NaN | NaN | 20231025 | Altres/Diversos | Altres/Diversos | Altres/Diversos |

114153 rows × 9 columns

Figura 4.3: Datos importados de calles de Catalunya para el experimento 2.

Como se puede observar, existen 114153 filas, es decir, 114153 calles. Si se pasara este número de datos a texto, los *tokens* resultantes excederían del máximo de la ventana de contexto del LLM. Por lo tanto, se ha tenido que pre-procesar y seleccionar una muestra para realizar el experimento.

### 4.2.2. Pre-procesado de datos de referencia reales

Para reducir el número de datos de referencia se han aplicado tres acciones de pre-procesado. La primera es eliminar todas aquellas filas que no son calles reales, por ejemplo, los valores *No consta* u *Otros*. Después, para acotar el estudio, se han seleccionado solamente calles de Barcelona. Por último, se ha seleccionado una muestra de todas las calles de Barcelona. El *dataframe* resultante, es el que se utiliza para el experimento. La Figura 4.4 es un ejemplo con una muestra de diez datos de referencia.

| | nom_compost_via | codi_unitat_poblacional | codi_tipus_via | nom_via | nexe_via | data | nom_municipi | nom_unitat_poblacional | tipus_via |
|---|---|---|---|---|---|---|---|---|---|
| 0 | Carrer de Batista i Roca | 0801930001017 | 016 | Batista i Roca | de | 20130601 | Barcelona | Barcelona | Carrer |
| 1 | Carrer dels Periodistes | 0801930001017 | 016 | Periodistes | dels | 20130601 | Barcelona | Barcelona | Carrer |
| 2 | Carrer de Fraga | 0801930001017 | 016 | Fraga | de | 20130601 | Barcelona | Barcelona | Carrer |
| 3 | Carrer de Jorba | 0801930001017 | 016 | Jorba | de | 20130601 | Barcelona | Barcelona | Carrer |
| 4 | Passatge del Pintor Tapiró | 0801930001017 | 051 | Pintor Tapiró | del | 20130601 | Barcelona | Barcelona | Passatge |
| 5 | Passatge dels Camps Elisis | 0801930001017 | 051 | Camps Elisis | dels | 20130601 | Barcelona | Barcelona | Passatge |
| 6 | Camí de la Reineta | 0801930001017 | 013 | Reineta | de la | 20130601 | Barcelona | Barcelona | Camí |
| 7 | Carrer de Santa Magdalena | 0801930001017 | 016 | Santa Magdalena | de | 20130601 | Barcelona | Barcelona | Carrer |
| 8 | Carrer d'Almacelles | 0801930001017 | 016 | Almacelles | d' | 20130601 | Barcelona | Barcelona | Carrer |
| 9 | Plaça de les Basses d'en Barral | 0801930001017 | 057 | Basses d'en Barral | de les | 20130601 | Barcelona | Barcelona | Plaça |

Figura 4.4: Ejemplo de muestra de datos de referencia del experimento 2.

### 4.2.3. Pre-procesado de datos transaccionales ficticios

El siguiente paso ha sido la creación de datos transaccionales de calles con errores de los datos de referencia. Para ello, se ha utilizado la columna 'nom_compost_via', que contiene el nombre completo de las calles. A continuación, se ha aplicado la función de generación de errores del experimento 1 (ver sección 4.1.3) y se han obtenido unos datos del estilo de la Figura 4.5.

| | datos_transaccionales |
|---|---|
| 0 | Passatgedels Camps Elisis |
| 1 | c@rr3r d'@lm@c3ll3$ |
| 2 | PLaÇa De LEs BASsEs D'eN BArrAl |
| 3 | Passatge dels Camps Elisis |
| 4 | Roca i Batista de Carrer |
| 5 | CarrerdeBatistaiRoca |
| 6 | Carrer dels Periodistes |
| 7 | CarreR De SanTa MagdAlEna |
| 8 | Camí de la Reinehta |
| 9 | Plaça de les Basses d'en Barral |
| 10 | karrer de fraga |

Figura 4.5: Muestra de datos transaccionales con errores del experimento 2.

En este caso, los datos contienen errores y, además, no tienen la estructura que marcan los datos de referencia, es decir, no tienen los atributos correctos.

### 4.2.4. Generación del *Prompt*

El *prompt* resultante para el experimento 2 se muestra a continuación. Se ha seguido la misma estructura que en el experimento 1 (ver sección 4.1.4). Nótese que se han acortado los datos de referencia y los datos transaccionales para un mayor entendimiento del mismo.

***Prompt:***

*Eres un experto en corrección de datos, especializado en la detección y estandarización de errores en los datos de calles de Cataluña, España. Además, eres experto en la alineación de datos transaccionales con datos de referencia. Tengo dos conjuntos de datos en formato CSV con delimitador ";". El primero incluye datos de referencia con el formato y atributos correctos de las calles de Cataluña, España. El segundo contiene datos transaccionales de calles que contienen errores. Los errores pueden ser de diferentes tipos, como un mal formato. Tu tarea es corregir meticulosamente los datos transaccionales, basándote en los datos de referencia, de manera que tengan el formato correcto. Aplica tu conocimiento para alinear y estandarizar los datos transaccionales con precisión. Asegúrate de que los valores que ya están estandarizados queden inalterados. No utilices herramientas de análisis ni Python, solo tu conocimiento.*

*Tu respuesta debe contener todos los datos transaccionales, tanto los correctos como los corregidos. El formato de la respuesta debe ser un CSV con cabecera, con delimitador ";z con las columnas que se muestran a continuación:*

*Dato transaccional; nom_compost_via; codi_unitat_poblacional; codi_tipus_via; nom_via; nexe_via; data; nom_municipi; nom_unitat_poblacional; tipus_via*

*MUY IMPORTANTE: Sólo incluye el CSV en tu respuesta. No incluyas espacios entre cada fila. No incluyas ninguna palabra adicional en tu respuesta. No incluyas explicaciones, informaciones, razonamientos o pasos seguidos.*
*MUY IMPORTANTE: Asegúrate que el número de filas de la respuesta es el correcto. No dupliques, ni te dejes valores.*

*Datos de referencia:*
*nom_compost_via; codi_unitat_poblacional; codi_tipus_via; nom_via; nexe_via; data; nom_municipi; nom_unitat_poblacional; tipus_via*

*Carrer de Batista i Roca; 0801930001017; 016; Batista i Roca; de; 20130601; Barcelona; Barcelona; Carrer*
*Carrer dels Periodistes; 0801930001017; 016; Periodistes; dels; 20130601; Barcelona; Barcelona; Carrer*
*Carrer de Fraga; 0801930001017; 016; Fraga; de; 20130601; Barcelona; Barcelona; Carrer*
*Carrer de Jorba; 0801930001017; 016; Jorba; de; 20130601; Barcelona; Barcelona; Carrer*
*...*

*Archivo CSV a completar:*
*Dato transaccional; nom_compost_via; codi_unitat_poblacional; codi_tipus_via; nom_via; nexe_via; data; nom_municipi; nom_unitat_poblacional; tipus_via*
*Passatgedels Camps Elisis;*
*c@rr3r d'@lm@c3ll3$;*
*PLaÇa De LEs BASsEs D'eN BArrAl;*
*Passatge dels Camps Elisis;*
*Roca i Batista de Carrer;*
*CarrerdeBatistaiRoca;*
*Carrer dels Periodistes;*
*CarreR De SanTa MagdAlEna;*
*Camí de la Reinehta;*
*Plaça de les Basses d'en Barral;*
*karrer de fraga;*
*...*

*Ejemplo de respuesta:*
*Dato transaccional; nom_compost_via; codi_unitat_poblacional; codi_tipus_via; nom_via; nexe_via; data; nom_municipi; nom_unitat_poblacional; tipus_via*
*C. Jorba; Carrer de Jorba; 0801930001017; 016; Jorba; de; 20130601; Barcelona; Barcelona; Carrer*
*Avinguda Almacelles; Carrer d'Almacelles; 0801930001017; 016; Almacelles; d'; 20130601; Barcelona; Barcelona; Carrer*
*C/ Periodistes; Carrer dels Periodistes; 0801930001017; 016; Periodistes; dels; 20130601; Barcelona; Barcelona; Carrer*

*Archivo CSV respuesta:*

### 4.2.5. Post-procesado de datos

Al igual que en el experimento 1, las respuesta obtenidas se encontraban en formato texto, por lo que se debía hacer un post-procesado de los datos corregidos. Las respuestas tenían el siguiente formato:

*Dato transaccional; nom_compost_via; codi_unitat_poblacional; codi_tipus_via; nom_via; nexe_via; data; nom_municipi; nom_unitat_poblacional; tipus_via*
*C. Jorba; Carrer de Jorba; 0801930001017; 016; Jorba; de; 20130601; Barcelona; Barcelona; Carrer*
*Avinguda Almacelles; Carrer d'Almacelles; 0801930001017; 016; Almacelles; d'; 20130601; Barcelona; Barcelona; Carrer*
*C/ Periodistes; Carrer dels Periodistes; 0801930001017; 016; Periodistes; dels; 20130601; Barcelona; Barcelona; Carrer*
*...*

Por consiguiente, se realizó un procesado del texto de manera que los datos quedaban almacenados en un *dataframe* de *pandas*. Así pues, los datos quedaban estructurados para evaluarlos.

### 4.2.6. Evaluación e impresión de resultados

Para terminar el experimento 2, se realizó la evaluación e impresión de resultados siguiendo las métricas del Capítulo 6. Concretamente se calculó la calidad inicial de los datos transaccionales, y después, para cada modelo, la calidad final, el incremento de calidad y el tiempo de ejecución de la inferencia.

## 4.3. Experimento 3 - Imputación

Para el último experimento, se ha querido estudiar un caso muy común que se encuentra en los sistemas de información de organizaciones o empresas. Éstas pueden tener unos datos maestros de muy buena calidad, pero cuando se realizan acciones o eventos en los sistemas de información, es normal que algunos datos no se introduzcan correctamente o, directamente, ni se introduzcan. La consecuencia de este este hecho es que los datos transaccionales acaban siendo incompletos y con datos ausentes.

El objetivo de este experimento es explorar si un LLM es capaz de corregir datos transaccionales ausentes e incorrectos a la vez en base a unos datos maestros que sí que tienen calidad. Así pues, el modelo deberá imputar o rellenar los datos ausentes y corregir aquellos que no sean correctos.

En este caso, se han utilizado datos completamente fictícios o sintéticos, tanto para los datos maestros como para los datos transaccionales. En particular, se ha diseñado un conjunto de datos

que contiene la siguiente información sobre personas: DNI, Nombre, Apellido 1, Apellido 2, Email, Teléfono. En la sección 4.3.1 se muestra como se han generado estos datos.

Por otro lado, el flujo de trabajo o *pipeline* que se ha seguido se muestra en la Figura 4.6.

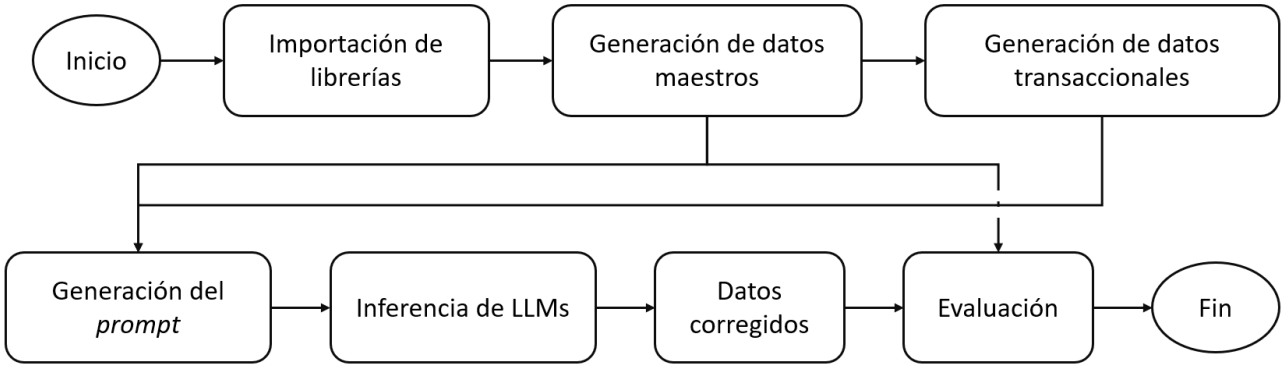


Figura 4.6: Diagrama de flujo del experimento 3.

El código de Python se adjunta en el Anexo B.3 y seguidamente se presentan los pasos que se han seguido y que presentan alguna diferencia con el experimento 1.

### 4.3.1. Generación de datos maestros

Después de importar todas las librerías necesarias, se ha procedido a generar los datos maestros con información de personas (DNI, Nombre, Apellido 1, Apellido 2, Email, Teléfono). Para cada atributo, se ha programado una función de Python que hace uso de la librería *random* y que genera un valor aleatorio para dicho atributo.

| | DNI | Nombre | Apellido_1 | Apellido_2 | Email | Teléfono |
|---|---|---|---|---|---|---|
| 0 | 99087821A | Sergio | Ortiz | Garrido | sergio.ortiz.garrido@outlook.com | 771427803 |
| 1 | 24194632N | Luis | Márquez | Gómez | luis.marquez.gomez@gmail.com | 645274245 |
| 2 | 43512101B | Mateo | Vargas | Nuñez | mateo.vargas.nuñez@gmail.com | 730104512 |
| 3 | 38813711S | Jorge | Bravo | Arias | jorge.bravo.arias@gmail.com | 775033250 |
| 4 | 75779969Y | Angela | Giménez | Moreno | angela.gimenez.moreno@yahoo.com | 790317320 |
| 5 | 97115104L | Alicia | Fernández | Garrido | alicia.fernandez.garrido@gmail.com | 717212448 |
| 6 | 62352697L | Mario | Sanz | Bravo | mario.sanz.bravo@yahoo.com | 757205075 |
| 7 | 32467706D | Hugo | Gutiérrez | Ferrer | hugo.gutierrez.ferrer@gmail.com | 789889192 |
| 8 | 79891324L | David | Carrasco | Carrasco | david.carrasco.carrasco@outlook.com | 715641243 |
| 9 | 14126274L | Paula | Martínez | Santos | paula.martinez.santos@outlook.com | 683543641 |

Figura 4.7: Muestra de datos maestros del experimento 3.

A continuación, se ha creado un *dataframe* llamando a cada una de las funciones para cada

atributo de cada fila. El resultado es un conjunto de datos maestros sintéticos y la Figura 4.7 presenta un ejemplo del *dataframe* resultante.

### 4.3.2. Generación de datos transaccionales

El siguiente paso ha sido la generación de los datos transaccionales. Para crearlos se han aplicado dos pasos:

- **Muestreo de los datos maestros:** Se ha cogido una muestra aleatoria (con datos repetidos) de los datos maestros.
- **Introducción de datos ausentes y errores:** A un porcentaje de filas del conjunto de datos transaccionales, se les han suprimido 2 atributos y se les han introducido errores en 1 atributo con la fórmula de introducción de errores del experimento 1 (ver sección 4.1.3).

El resultado es un conjunto de datos transaccionales con datos ausentes y errores que se puede ver en la Figura 4.8.

| | DNI | Nombre | Apellido_1 | Apellido_2 | Email | Teléfono |
|---|---|---|---|---|---|---|
| 0 | NaN | NaN | F ernández | Garrido | alicia.fernandez.garrido@gmail.com | 717212448 |
| 1 | 79891324L | David | Carraasco | Carrasco | NaN | NaN |
| 2 | 14126274L | NaN | Martínez | NaN | paula.martinez.santos@outlook.com | 68354364¿1 |
| 3 | NaN | Alicia | Fernández | Garrido | aLicia.feRNaNdEZ.garrido@GMAIL.COM | NaN |
| 4 | 9908782 1A | NaN | NaN | Garrido | sergio.ortiz.garrido@outlook.com | 771427803 |
| 5 | NaN | Sergio | Ortiz | Garrido | sergio.ortiz.garrido@outlook.com | NaN |
| 6 | NaN | Luis | Márquez | NaN | luis.marquez.gomez@ggmail.com | 645274245 |
| 7 | NaN | Hugo | Gutiérrez | Ferrer | NaN | 789889192 |
| 8 | 62q352697L | NaN | Sanz | NaN | mario.sanz.bravo@yahoo.com | 757205075 |
| 9 | 14126274L | Paula | MarTÍnEz | Santos | NaN | NaN |
| 10 | NaN | NaN | Varrgas | Nuñez | mateo.vargas.nuñez@gmail.com | 730104512 |
| 11 | 75779969Y | NaN | Giménez | NaN | angela.gimenez.moreno@yahoo.co | 790317320 |
| 12 | 97115104L | NaN | Fernández | Garri?do | alicia.fernandez.garrido@gmail.com | NaN |
| 13 | NaN | Mateo | Vargas | Nuñez | mateo.vargys.nuñez@gmail.com | NaN |
| 14 | 75779969y | Angela | Giménez | NaN | NaN | 790317320 |
| 15 | NaN | Mateo | Varrgas | NaN | mateo.vargas.nuñez@gmail.com | 730104512 |
| 16 | 75779969Y | Angela | Giménez | Moreno | NaN | NaN |
| 17 | NaN | Hugo | Gutiérrez | Ferrer | NaN | 789889192 |
| 18 | 32467706d | NaN | Gutiérrez | NaN | hugo.gutierrez.ferrer@gmail.com | 789889192 |
| 19 | 14126274L | Paula | NaN | NaN | paula.martinez.santos@outlook.com | 68o3543641 |

Figura 4.8: Muestra de datos transaccionales del experimento 3.

El LLM tendrá que, por un lado, imputar los valores ausentes (NaN) basándose en los datos maestros y, por otro, corregir los datos erróneos.

### 4.3.3. Generación del *Prompt*

El *prompt* resultante para el experimento 3 se muestra a continuación. Se ha seguido la misma estructura que en el experimento 1 (ver sección 4.1.4). Nótese que se han acortado los datos maestros y los datos transaccionales para un mayor entendimiento del mismo.

***Prompt:***

*Eres un experto en corrección de datos, especializado en la detección y estandarización de errores en datos transaccionales con información de personas. Además, eres experto en la alineación de datos transaccionales con datos maestros. Tengo dos conjuntos de datos en formato CSV con delimitador ";". El primero incluye datos maestros con el formato y atributos correctos de personas y sus datos personales. El segundo contiene datos transaccionales de personas que contienen errores y datos ausentes (NaN). Los errores pueden ser de diferentes tipos, como un mal formato. Tu tarea es corregir meticulosamente los datos transaccionales, basándote en los datos maestros, de manera que tengan el formato correcto y no haya datos ausentes (NaN). Aplica tu conocimiento para alinear y estandarizar los datos transaccionales con precisión. Asegúrate de que los valores que ya están estandarizados queden inalterados. No utilices herramientas de análisis ni Python, solo tu conocimiento.*

*Tu respuesta debe contener todos los datos transaccionales corregidos. El formato de la respuesta debe ser un CSV con cabecera, con delimitador ";", y con las columnas que se muestran a continuación:*

*DNI; Nombre; Apellido_1; Apellido_2; Email; Teléfono*

*MUY IMPORTANTE: Sólo incluye el CSV en tu respuesta. No incluyas espacios entre cada fila. No incluyas ninguna palabra adicional en tu respuesta. No incluyas explicaciones, informaciones, razonamientos o pasos seguidos.*
*MUY IMPORTANTE: Asegúrate que el número de filas de la respuesta es el correcto. No dupliques, ni te dejes valores.*

*Datos maestros:*
*DNI; Nombre; Apellido_1; Apellido_2; Email; Teléfono*
*99087821A; Sergio; Ortiz; Garrido; sergio.ortiz.garrido@outlook.com; 771427803*
*38813711S; Jorge; Bravo; Arias; jorge.bravo.arias@gmail.com; 775033250*
*75779969Y; Angela; Giménez; Moreno; angela.gimenez.moreno@yahoo.com; 790317320*

*...*

*Archivo CSV a completar:*
*DNI; Nombre; Apellido_1; Apellido_2; Email; Teléfono*
*NaN; NaN; F ernández; Garrido; alicia.fernandez.garrido@gmail.com; 717212448*
*NaN; Sergio; Ortiz; Garrido; sergio.ortiz.garrido@outlook.com; NaN*
*...*

*Ejemplo de respuesta:*
*DNI; Nombre; Apellido_1; Apellido_2; Email; Teléfono*
*91697831M; Miguel; Ortiz; Nieto; miguel.ortiz.nieto@yahoo.com; 738427558*
*81213749J; Raquel; Álvarez; Blanco; raquel.alvarez.blanco@gmail.com; 650538087*

*Archivo CSV respuesta:*

### 4.3.4. Post-procesado de datos

Al igual que en el experimento 2, las respuesta obtenidas se encontraban en formato texto, por lo que se debía hacer un post-procesado de los datos corregidos. Las respuestas tenían el siguiente formato:

*DNI; Nombre; Apellido_1; Apellido_2; Email; Teléfono*
*91697831M; Miguel; Ortiz; Nieto; miguel.ortiz.nieto@yahoo.com; 738427558*
*20102230T; Lorena; Peña; Soto; lorena.peña.soto@outlook.com; 757957493*
*...*

Por consiguiente, se realizó un procesado del texto de manera que los datos quedaban almacenados en un *dataframe* de *pandas*. Así pues, los datos quedaban estructurados para evaluarlos.

### 4.3.5. Evaluación e impresión de resultados

El último paso del experimento 3 fue la evaluación e impresión de resultados siguiendo las métricas del Capítulo 6. Concretamente se calculó la calidad inicial de los datos transaccionales y, después, para cada modelo, la calidad final, el incremento de calidad, la exactitud con la cual se corrigieron los datos erróneos y se detectaron los datos correctos, y el tiempo de ejecución de la inferencia.

# Capítulo 5

# Modelos LLM

Este capítulo presenta los modelos LLM que se han utilizado y evaluado para resolver las tareas de calidad del dato definidas en los experimentos del Capítulo 4. La selección se ha basado en un equilibrio entre la relevancia de los LLMs y los recursos económicos necesarios para ejecutarlos. Así pues, la sección 5.1 presenta a GPT-3.5 Turbo, la sección 5.2 a GPT-4 Turbo, la sección 5.3 a Gemini Pro, y finalmente, la sección 5.4 presenta a LLaMA 2 70B Chat.

## 5.1. GPT-3.5 Turbo (OpenAI)

El modelo GPT-3 de OpenAI, lanzado en junio de 2020 con 175.000 millones de parámetros, presentaba grandes capacidades de modelado del lenguaje. Aun así, su tamaño masivo requería de un coste computacional muy elevado. Esto se traducía en un gran coste económico para los desarrolladores. Además, un requisito para construir aplicaciones a tiempo real era tener baja latencia, factor que GPT-3 carecía.

Por este motivo, en marzo de 2022, OpenAI publicó el modelo GPT-3.5 con una arquitectura mejorada y optimizada respecto a su antecesor GPT-3. Conservaba su rendimiento y mejoraba su eficiencia computacional. Aun así, los desarrolladores encontraban demasiado caro el uso de este modelo con miles o millones de usuarios.

Así pues, en noviembre de 2022, OpenAI anuncia GPT-3.5 Turbo, una iteración de GPT-3.5 que mejoraba todavía más la eficiencia en términos de computación, velocidad y coste económico. Este modelo permitió a los desarrolladores realizar aplicaciones económicamente más viables.

GPT-3.5 Turbo es un modelo de propiedad privada que OpenAI comercializa. Por esto, no se ha revelado ni la arquitectura ni los datos con los que se entrenó. Aun así, se puede deducir que es un modelo que utiliza una arquitectura basada en *transformers* con un pre-entrenamiento autorregresivo, un ajuste con RLHF y una capa de seguridad por encima.

Cuando se lanzó el modelo, tenía una ventana de contexto de 4.096 tokens y se ha ido actualizando hasta día de hoy, que cuenta con 16.385 tokens. Actualmente está accesible a través de la API de

OpenAI con un coste de 0.0005$/1K tokens para las entradas y 0.0015$/1K tokens para las salidas. Concretamente, se ha utilizado la versión llamada *gpt-3.5-turbo-0125*.

## 5.2. GPT-4 Turbo (OpenAI)

GPT-4 Turbo fue lanzado en noviembre de 2023 como una evolución de GPT-4 que mantenía sus capacidades haciéndolo todavía más eficiente (OpenAI, 2023b). Se optimizó la latencia, haciéndolo más rápido, y se redujo el coste computacional, permitiendo reducir el coste económico. Concretamente, la inferencia de GPT-4 Turbo pasó a costar 3 veces menos que la de GPT-4. También se añadieron conocimientos y datos hasta abril de 2023.

GPT-4 Turbo también es un modelo de propiedad privada, por lo que no se ha revelado la arquitectura ni los datos que utilizó para su entrenamiento. Sin embargo, en internet hay rumores que indican que utiliza una arquitectura de mezcla de expertos (MoE) (ver sección 2.1.2.2).

Según el informe técnico de GPT-4 publicado por OpenAI (2023a), se puede deducir que se hace uso de la arquitectura de *transformers* con pre-entrenamiento autorregresivo, un ajuste de RLHF y una capa de seguridad. Sin embargo, no indican nada sobre mezcla de expertos (MoE).

En el momento de realización de esta investigación, el modelo se accede a través de la API de OpenAI, con una ventana de contexto de 128.000 tokens y un coste de 0.01$/1K tokens para las entradas y 0.03$/1K tokens para las salidas. En particular, la versión que se ha utilizado es la *gpt-4-0125-preview*.

## 5.3. Gemini Pro (Google)

En diciembre de 2023, Pichai y Hassabis (2023) anunciaron Gemini 1.0, una família de tres LLMs llamados Gemini Ultra, Gemini Pro y Gemini Nano. El primero es el modelo más grande diseñado para realizar tareas muy complejas, el segundo es un modelo versátil en muchas tareas y a la vez escalable, y el tercero es un modelo muy eficiente pensado para ejecutarse en dispositivos móviles.

En el momento del lanzamiento, sólo se abrió al público Gemini Pro y en fase experimental. Sin embargo, según Pichai y Hassabis (2023), Gemini Ultra superaría a GPT-4 en la mayoría de evaluaciones y *benchmarks*.

La familia Gemini 1.0 es multimodal por defecto, es decir incluye el procesamiento de imágenes, audio y texto. Respecto a la arquitectura, según el informe técnico de Google Gemini Team (2023), utiliza una arquitectura de *transformers* optimizada para la inferencia en las unidades de procesamiento tensorial de Google (TPUs). Sin embargo, no se revela información técnica detallada del proceso de entrenamiento y desarrollo del modelo.

Actualmente, sólo se puede acceder al modelo Gemini Pro a través de la API de VertexAI y, además, se encuentra en una versión experimental que puede presentar errores. Consta de una ventana

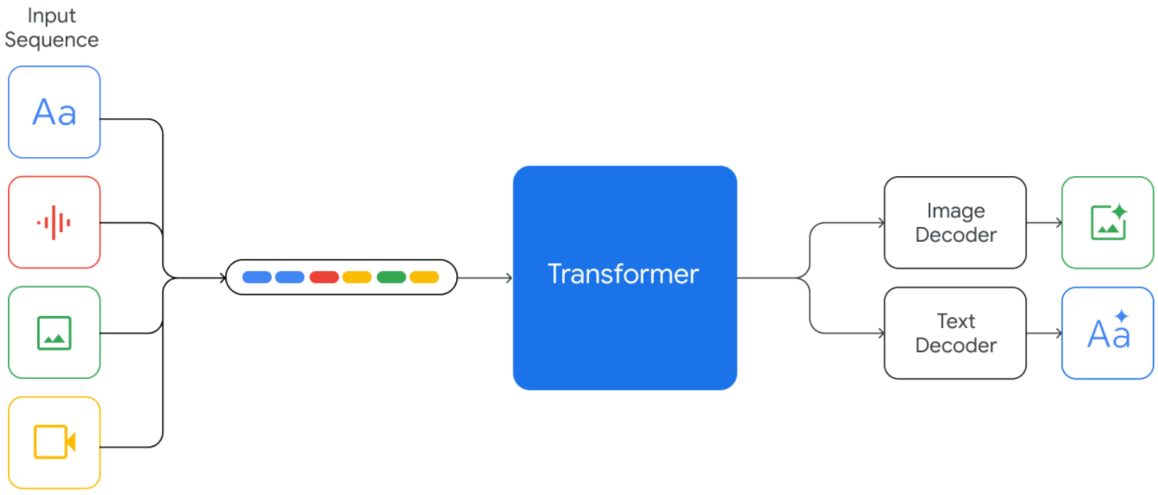


Figura 5.1: Diagrama de la arquitectura de los modelos Gemini de Google. Tomado de Google Gemini Team (2023).

de contexto de 32.000 tokens y su precio es gratuito hasta 60 consultas por minuto. Al superar las 60 consultas por minuto, el coste es de 0.00025$/1K tokens para las entradas y 0.0005$/1K tokens para las salidas. Como se puede comprobar es más económico que los modelos de OpenAI.

## 5.4. LLaMA 2 70B Chat (Meta)

En julio de 2023, Meta publicó LLaMA 2, una colección de LLMs con tres tamaños: 7.000, 13.000 y 70.000 millones de parámetros. También se incluyó LLaMA 2 Chat, los correspondientes modelos ajustados para la tarea de conversación (Touvron et al., 2023). A diferencia de los modelos anteriores, Meta introdujo una licencia de uso comercial que permitía el acceso abierto a estos modelos con ciertas cláusulas de responsabilidad y protección. Además, fueron más transparentes que OpenAI y Google al proporcionar información sobre el proceso de desarrollo de estos modelos.

Los autores, Touvron et al. (2023), mostraron el proceso teórico de cómo se había desarrollado el modelo en un artículo científico. El primer paso fue el pre-entrenamiento de LLaMA 2, utilizando un *transformer* con pre-entrenamiento auto-regresivo, una ventana de contexto de 4096 *tokens*, y una técnica llamada atención de consultas agrupadas. Respecto a las fuentes de información, se utilizaron datos públicos de internet, que se tradujeron en 2 billones (españoles) de *tokens*, utilizando la *tokenización* de Codificación de Pares de Bytes (BPE).

Una vez se pre-entrenó a LLaMA 2, se creó una versión inicial de LLaMA 2 Chat mediante un ajuste fino supervisado con más de 100.000 instrucciones. Posteriormente, el modelo se fue refinando a partir de RLHF con más de 1 millón de preferencias humanas. En particular, se utilizaron las técnicas de RLHF llamadas muestreo de rechazo y optimización de políticas próximas, junto a modelos de recompensa de seguridad y de utilidad.

Respecto a la capa de seguridad, se realizaron acciones tanto en el pre-entrenamiento como en

el ajuste fino. Primero, se garantizó que el conjunto de datos de pre-entrenamiento estuviera libre de contenido peligroso, tóxico o dañino, y luego, se realizó un ajuste fino supervisado de seguridad, un RLHF de seguridad y la inclusión de *prompts* por defecto que bloquearan cierto contenido de las respuestas generadas.

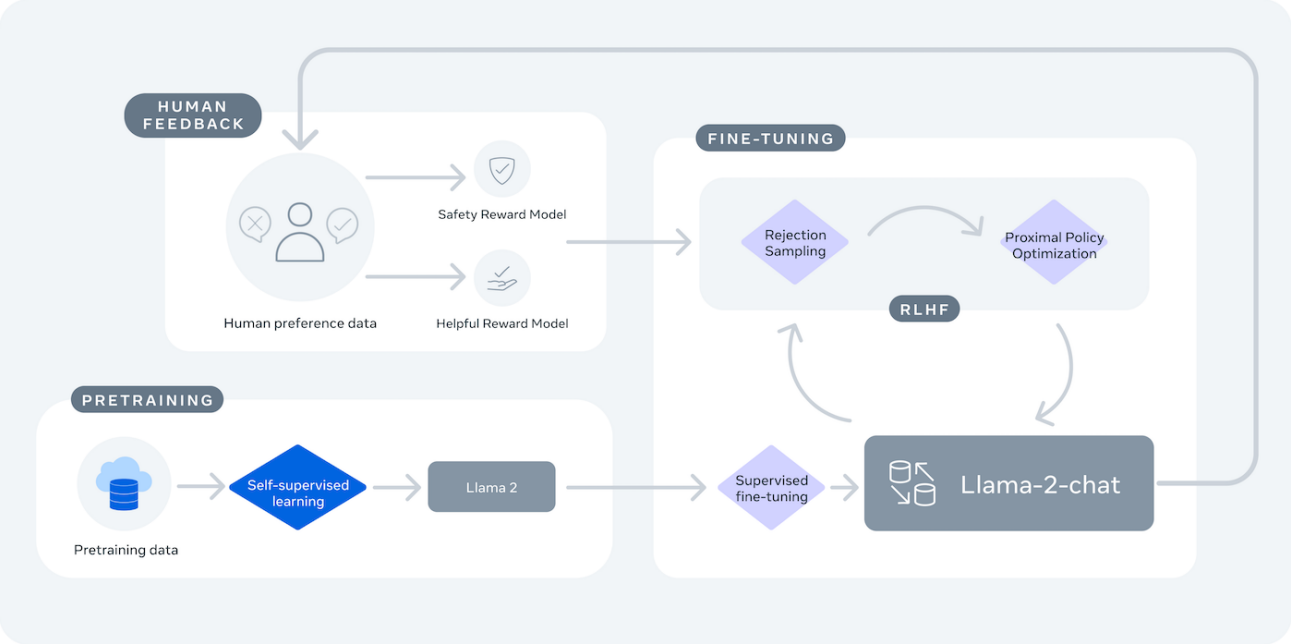


Figura 5.2: Entrenamiento del modelo LLaMA 2 Chat de Meta. Tomado de Touvron et al. (2023).

Los modelos resultantes fueron abiertos por Meta y cualquiera con suficientes recursos computacionales los podía usar bajo ciertos términos de responsabilidad. En paralelo, Hugging Face, adoptó el modelo como un estándar de la industria y lo puso a disposición a través de su API de inferencia. Para este proyecto, se ha utilizado la API de inferencia de Hugging Face Pro, la cual por 9$ al mes, permite realizar inferencias con el modelo de LLaMA 2 70B Chat con un rendimiento aceptable.

# Capítulo 6

# Evaluación

En el Capítulo 2, sección 2.1.2.6, se expusieron diferentes pruebas de evaluación para LLMs. Éstas incluían desde exámenes para humanos hasta *benchmarks* específicos para verificar las habilidades de los modelos. Sin embargo, no se encontró un conjunto de pruebas que evaluase a los LLMs para realizar tareas de calidad del dato.

Por este motivo, este capítulo, presenta la metodología de evaluación que se ha diseñado para verificar la eficacia de los LLMs en tareas de calidad del dato, concretamente los experimentos del Capítulo 4.

En la evaluación de resultados, se ha considerado que el problema es equivalente a uno de aprendizaje supervisado. Es decir, se dispone de las etiquetas correctas del resultado y se comparan con las respuestas predichas por los modelos. Así pues, para las evaluaciones, se han tenido en cuenta los siguientes tipos de datos:

- **Datos de referencia, $y^{ref}$:** Datos que marcan el contenido estándar que deberían tener los datos en los sistemas de información. En la evaluación actúan como las etiquetas correctas a predecir o también llamadas como *ground truth* en inglés.
- **Datos transaccionales, $y^{trans}$:** Datos de los sistemas de información que se encuentran con mala calidad y que hay que corregir. En la evaluación, se comparan con los datos de referencia para saber la calidad inicial que tienen los datos transaccionales.
- **Datos predichos, $y^{pred}$:** Datos que los LLMs habrán corregido, teniendo en cuenta los datos de referencia y los datos transaccionales. En la evaluación actúan como los datos predichos por el modelo y se comparan con los datos de referencia.

Con estos tres tipos de datos, se han obtenido diferentes métricas de evaluación que permiten comparar a los modelos. A continuación, la siguiente sección presenta estas métricas.

## 6.1. Métricas de evaluación

Respecto a las métricas de evaluación, se han definido cuatro tipos: métricas de calidad de los datos, métricas de exactitud de las predicciones de los modelos, métricas de tiempo de ejecución y métricas de formato de la respuesta. Las siguientes secciones presentan las métricas con sus formulas y cómo se ha implementado en el código.

### 6.1.1. Calidad de los datos transaccionales

Se ha definido la calidad de un conjunto de datos como el porcentaje de valores correctos respecto a sus datos de referencia. Teniendo en cuenta que en los experimentos tenemos datos transaccionales y datos predichos, cada uno tendrá una calidad distinta. Por lo tanto, existe una calidad inicial y una calidad posterior a la inferencia del modelo.

#### 6.1.1.1. Calidad inicial

La calidad inicial es la fracción entre los datos transaccionales que son correctos, basándose en los datos de referencia, y el número total de datos transaccionales. Matemáticamente se ha expresado con la siguiente fórmula:

$$C_{ini}(y^{ref}, y^{trans}) = \frac{1}{n_{\text{total}}} \sum_{i=0}^{n_{\text{total}}-1} 1(y_i^{ref} = y_i^{trans})$$

Para implementarla en el código, se ha utilizado la librería de Scikit-learn, la cual incorpora una función de exactitud llamada `sklearn.metrics.accuracy_score(y,ŷ)` (Scikit-Learn, 2024):

$$\texttt{sklearn.metrics.accuracy\_score(}y^{ref}\texttt{,}y^{trans}\texttt{)}$$

#### 6.1.1.2. Calidad posterior

Similar al caso anterior, la calidad posterior es la fracción entre los datos predichos que son correctos, basándose en los datos de referencia, y el número total de datos predichos. Matemáticamente se ha expresado con la siguiente fórmula:

$$C_{post}(y^{ref}, y^{pred}) = \frac{1}{n_{\text{total}}} \sum_{i=0}^{n_{\text{total}}-1} 1(y_i^{ref} = y_i^{pred})$$

De la misma manera, se ha utilizado la librería de Scikit-learn, con la función de exactitud (Scikit-Learn, 2024):

$$\texttt{sklearn.metrics.accuracy\_score(}y^{ref}\texttt{,}y^{pred}\texttt{)}$$

#### 6.1.1.3. Incremento de calidad

El incremento de calidad se ha definido como la diferencia entre la calidad posterior y la calidad inicial. Este valor indica cómo el LLM ha mejorado la calidad de los datos después de realizar la tarea. Matemáticamente, se describe como:

$$\Delta C = C_{post} - C_{ini}$$

y se ha implementado directamente en el código.

### 6.1.2. Exactitud en la corrección de datos transaccionales erróneos

Esta métrica mide la exactitud con la cual el modelo corrige los datos transaccionales erróneos. Para calcularla, se tendrán en cuenta sólo el subconjunto de datos transaccionales que inicialmente contenían errores. Así pues, se define como la fracción entre los datos transaccionales que inicialmente tenían errores y han sido correctamente corregidos, y el número de datos transaccionales erróneos en un principio. Matemáticamente se ha expresado de la siguiente manera:

$$Ex_{err}(y_{err}^{ref}, y_{err}^{pred}) = \frac{1}{n_{\text{err}}} \sum_{i=0}^{n_{\text{err}}-1} 1(y_{err_i}^{ref} = y_{err_i}^{pred})$$

y en el código, también se ha empleado la función de exactitud de (Scikit-Learn, 2024):

$$\texttt{sklearn.metrics.accuracy\_score}(y_{err}^{ref}, y_{err}^{pred})$$

### 6.1.3. Exactitud en la detección de datos transaccionales correctos

Esta métrica mide la exactitud con la cual el modelo detecta los datos transaccionales que estaban bien desde un principio y los deja sin alteraciones, lo que significa que ha realizado correctamente la detección. Para calcularla, se tendrán en cuenta sólo el subconjunto de datos transaccionales que inicialmente eran correctos. Por lo tanto, se define como la fracción entre los datos transaccionales que inicialmente eran correctos y se han dejado inalterados, y el número de datos transaccionales correctos en un principio. Matemáticamente se ha expresado como:

$$Ex_{corr}(y_{corr}^{ref}, y_{corr}^{pred}) = \frac{1}{n_{\text{corr}}} \sum_{i=0}^{n_{\text{corr}}-1} 1(y_{corr_i}^{ref} = y_{corr_i}^{pred})$$

y en el código, también se ha empleado la función de exactitud de (Scikit-Learn, 2024):

$$\texttt{sklearn.metrics.accuracy\_score}(y_{corr}^{ref}, y_{corr}^{pred})$$

### 6.1.4. Tiempo de ejecución

El tiempo de ejecución se ha definido como el tiempo de inferencia del modelo LLM. Este tiempo empieza justo antes de llamar a la API de inferencia y termina cuando se recibe la respuesta. Asi pues, se ha implementado en el código de la siguiente manera:

```
start_time = time.time()
# Inferencia del LLM
...
end_time = time.time()
time_LLM = end_time - start_time
```

### 6.1.5. Formato de salida correcto

La última métrica de evaluación consiste en determinar si la respuesta del modelo tiene el formato de salida que se le ha requerido. Por ejemplo, para el experimento 1, el formato debe ser solamente un CSV con delimitador '; ':

*Dato transaccional; Dato corregido*
*C. de Llobregat; Corbera de Llobregat*
*Gudall; Godall*
*Ponntss; Ponts*
*...*

Esta métrica se evaluará de manera manual y tendrá dos posibles resultados: formato de salida correcto o incorrecto. En caso que el modelo añada alguna línea de información adicional o la estructura no sea la solicitada, se considerará que el formato de salida ha sido incorrecto. Además, se verificará que el número de filas resultante sea el correcto, y en caso de no coincidir, se evaluará el formato como incorrecto.

# Capítulo 7

# Resultados y discusión

Este capítulo presenta los resultados obtenidos al ejecutar cada experimento del Capítulo 4, con los modelos LLM del Capítulo 5, evaluados con las métricas del Capítulo 6. Los resultados se presentan en tablas que contienen información sobre el *prompt*, el modelo utilizado y las métricas de evaluación resultantes. Además, se realiza el análisis de los resultados mediante representaciones gráficas y se comentan los hallazgos e implicaciones de los mismos.

Es importante destacar que Gemini Pro no fue capaz de inferir ninguna respuesta por problemas con la API. Concretamente, se obtenían errores que bloqueaban la consulta. Para más información consultar el Anexo A.

## 7.1. Resultados y discusión del experimento 1

Para el experimento 1 de corrección de valores, se ha utilizado el *prompt* de la sección 4.1.4 y se ha jugado con el número de datos que se le introducen. Es decir, se han estudiado cuatro *prompts* con diferentes números de datos de referencia y datos transaccionales. Las combinaciones evaluadas se muestran en la Tabla 7.1.

| *Prompt* | $N_{referencia}$ | $N_{transaccionales}$ | *Tokens* del *prompt* |
|---|---|---|---|
| *Prompt*_1_A | 10 | 20 | 680 |
| *Prompt*_1_B | 50 | 100 | 1520 |
| *Prompt*_1_C | 100 | 200 | 2537 |
| *Prompt*_1_D | 150 | 350 | 4041 |

Tabla 7.1: *Prompts* evaluados en el experimento 1. Cada *prompt* tiene un número de datos de referencia, $N_{referencia}$, y un número de datos transaccionales, $N_{transaccionales}$. También se presenta el número de *tokens* correspondientes a cada *prompt*.

Como se puede observar, aunque todos los *prompts* comparten la misma estructura, dependiendo del número de datos de referencia y datos transaccionales introducidos, el número de *tokens* resultantes del *prompt* es distinto. Por lo tanto, se ha querido estudiar si el parámetro de longitud del

*prompt* es importante. A continuación, se ha realizado una visualización de los cuatro *prompts* y se muestra en la Figura 7.1.

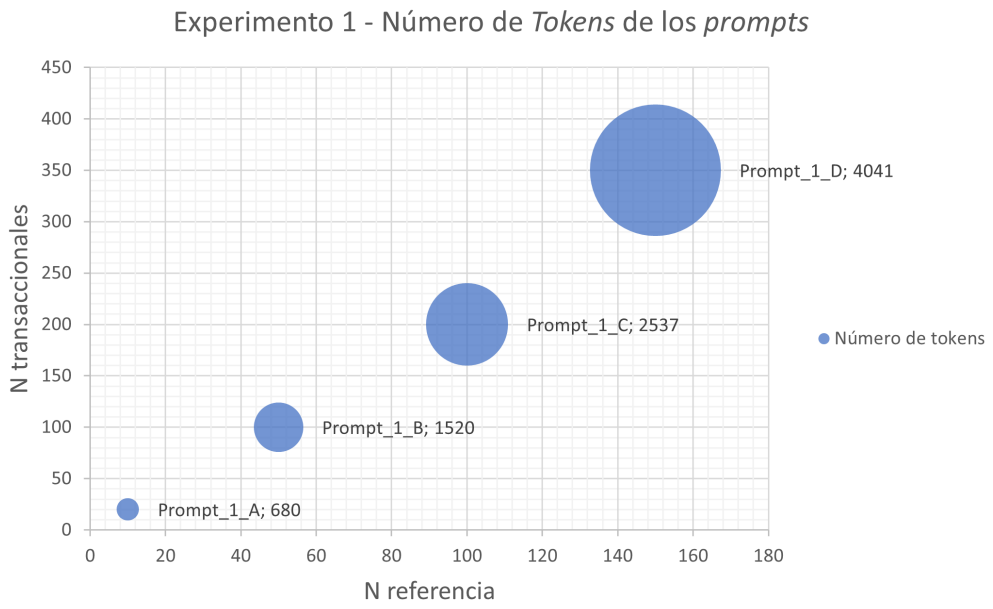


Figura 7.1: Número de *tokens* de cada *prompt* del experimento 1.

Podemos observar el número de *tokens* de cada *prompt* en función del número de datos de referencia y datos transaccionales introducidos. Nótese que el *Prompt_1_D*, con 150 datos de referencia y 350 datos transaccionales, llega al límite de la ventana de contexto seleccionada (4096 *tokens*). Es decir, para este experimento, sólo podríamos introducir hasta unos 500 datos, combinando datos de referencia y transaccionales. Este límite puede ser bastante restrictivo en ciertos casos o aplicaciones, ya que normalmente se trabaja con miles, cientos de miles o millones de datos.

Además, existe otro problema relacionado con el anterior, y es que normalmente los datos de referencia tendrán más de 500 valores. De hecho, pueden llegar a tener muchísimos más. Por ejemplo, en el experimento 1, existen 948 municipios en los datos de referencia. En este caso, ya se ve que si se quisiera implementar una corrección que tuviera en cuenta todos los datos de referencia, se tendría que recurrir a técnicas avanzadas como la RAG (ver sección 2.1.2.4).

Así pues, una potencial estrategia, sería la implementación de una arquitectura RAG, donde todos los datos de referencia se pasaran a *embeddings* y se guardaran en una base de datos vectorial. Seguidamente, la consulta del usuario contendría ciertos datos transaccionales a corregir, y se tendría que realizar una recuperación de la información para encontrar los datos de referencia correspondientes. Esta búsqueda se podría hacer por búsqueda vectorial, por palabras claves o mixta. Una vez seleccionados los datos de referencia necesarios para corregir los transaccionales, se crearía un *prompt* y se haría llegar al LLM. Sin embargo, existen restricciones en el número de datos transaccionales que se podrían introducir en el *prompt* debido a la ventana de contexto. Por lo tanto, para una gran cantidad de datos, se deberían realizar muchas iteraciones.

Por ejemplo, supongamos que todos los datos transaccionales a corregir son distintos. Entonces, cada uno necesitará su dato de referencia correspondiente. Es decir, en el experimento 1, tendríamos aproximadamente unos 250 datos transaccionales a corregir. Seguidamente, se realizaría una búsqueda en la base de datos vectorial y se obtendrían los 250 datos de referencia correspondientes. En consecuencia, con una ventana de contexto de 4096 *tokens*, cada consulta podría corregir un máximo de 250 datos transaccionales. Con un conjunto de datos transaccionales a corregir de un millón de valores, tendríamos que realizar 4000 consultas al LLM. Estos cálculos aproximados, muestran que escalar una solución de este tipo puede ser complicado en términos temporales y económicos.

A continuación, las Tablas 7.2, 7.3, 7.4 y 7.5 muestran los resultados del experimento 1. Nótese que en estas tablas hay valores con una X, que indican que el modelo ha fallado en la métrica correspondiente, o bien, no se ha podido evaluar porque la respuesta tenía un mal formato o el número de valores corregidos por el modelo era erróneo. Los problemas que se han tenido con Gemini Pro se comentan en el Anexo A.

| *Prompt*_1_A ($N_{referencia} = 10; N_{transaccionales} = 20$; *tokens* = 680) | | | | |
|---|---|---|---|---|
| | GPT-4 Turbo | GPT-3.5 Turbo | Gemini Pro | LLaMA 2 70B Chat |
| Calidad inicial [%] | 30.00 | 30.00 | 30.00 | 30.00 |
| Formato de salida | ✓ | ✓ | X | X |
| Número de filas correcto | ✓ | ✓ | X | ✓ |
| Calidad posterior [%] | 100.00 | 95.00 | X | 100.00 |
| Incremento de calidad [%] | 70.00 | 65.00 | X | 70.00 |
| Exactitud en datos erróneos [%] | 100.00 | 92.86 | X | 100.00 |
| Exactitud en datos correctos [%] | 100.00 | 100.00 | X | 100.00 |
| Tiempo de ejecución [s] | 16.21 | 4.05 | X | 15.92 |

Tabla 7.2: Resultados del experimento 1 con el *Prompt*_1_A ($N_{referencia} = 10; N_{transaccionales} = 20$; *tokens* = 680)

| *Prompt*_1_B ($N_{referencia} = 50; N_{transaccionales} = 100$; *tokens* = 1520) | | | | |
|---|---|---|---|---|
| | GPT-4 Turbo | GPT-3.5 Turbo | Gemini Pro | LLaMA 2 70B Chat |
| Calidad inicial [%] | 30.00 | 30.00 | 30.00 | 30.00 |
| Formato de salida | ✓ | ✓ | X | X |
| Número de filas correcto | ✓ | ✓ | X | ✓ |
| Calidad posterior [%] | 100.00 | 98.00 | X | X |
| Incremento de calidad [%] | 70.00 | 68.00 | X | X |
| Exactitud en datos erróneos [%] | 100.00 | 97.14 | X | X |
| Exactitud en datos correctos [%] | 100.00 | 100.00 | X | X |
| Tiempo de ejecución [s] | 74.53 | 19.99 | X | 51.14 |

Tabla 7.3: Resultados del experimento 1 con el *Prompt*_1_B ($N_{referencia} = 50; N_{transaccionales} = 100$; *tokens* = 1520)

| *Prompt*_1_C ($N_{referencia} = 100; N_{transaccionales} = 200$; *tokens* = 2537) | | | | |
|---|---|---|---|---|
| | GPT-4 Turbo | GPT-3.5 Turbo | Gemini Pro | LLaMA 2 70B Chat |
| Calidad inicial [%] | 30.00 | 30.00 | 30.00 | 30.00 |
| Formato de salida | ✓ | ✓ | X | X |
| Número de filas correcto | ✓ | ✓ | X | ✓ |
| Calidad posterior [%] | 99.50 | 95.50 | X | X |
| Incremento de calidad [%] | 69.50 | 65.50 | X | X |
| Exactitud en datos erróneos [%] | 99.30 | 94.37 | X | X |
| Exactitud en datos correctos [%] | 100.00 | 98.28 | X | X |
| Tiempo de ejecución [s] | 156.13 | 33.47 | X | 100.71 |

Tabla 7.4: Resultados del experimento 1 con el *Prompt*_1_C ($N_{referencia} = 100; N_{transaccionales} = 200$; *tokens* = 2537)

| *Prompt*_1_D ($N_{referencia} = 150; N_{transaccionales} = 350$; *tokens* = 4041) | | | | |
|---|---|---|---|---|
| | GPT-4 Turbo | GPT-3.5 Turbo | Gemini Pro | LLaMA 2 70B Chat |
| Calidad inicial [%] | 30.00 | 30.00 | 30.00 | 30.00 |
| Formato de salida | ✓ | ✓ | X | X |
| Número de filas correcto | X | X | X | X |
| Calidad posterior [%] | X | X | X | X |
| Incremento de calidad [%] | X | X | X | X |
| Exactitud en datos erróneos [%] | X | X | X | X |
| Exactitud en datos correctos [%] | X | X | X | X |
| Tiempo de ejecución [s] | 192.27 | 45.23 | X | 19.09 |

Tabla 7.5: Resultados del experimento 1 con el *Prompt*_1_D ($N_{referencia} = 150; N_{transaccionales} = 350$; *tokens* = 4041)

El siguiente paso en el análisis ha sido el estudio de los tiempos de ejecución. La Figura 7.2 presenta, para cada modelo, el tiempo en segundos que ha tardado el modelo en realizar la inferencia para cada *prompt* utilizado.

Podemos observar que, para un mismo modelo, a medida que el *prompt* se hace más grande, el tiempo de inferencia aumenta (a excepción del Prompt_1_D de LLaMA 2 70B Chat, que parece que hubo un problema en la API, ya que la respuesta tampoco fue correcta).

Por otro lado, GPT-4 Turbo es el modelo que más ha tardado en generar las respuestas, seguido de LLaMA 2 70B Chat y GPT-3.5 Turbo. Esto puede ser debido a que GPT-4 Turbo tenga más parámetros que el resto. También hay que destacar a GPT-3.5 Turbo, que consigue tiempos ajustados y puede ser un candidato en caso que el tiempo sea un factor importante, por ejemplo, en caso de que hubiera que hacer muchas consultas al modelo para corregir un conjunto de datos grande.

Otra manera de visualizar estos resultados se presenta en la Figura 7.3, donde se puede comparar mejor a los diferentes LLMs.

Estos tiempos pueden parecer pequeños, pero hay que considerar que, en estos *prompts*, se están

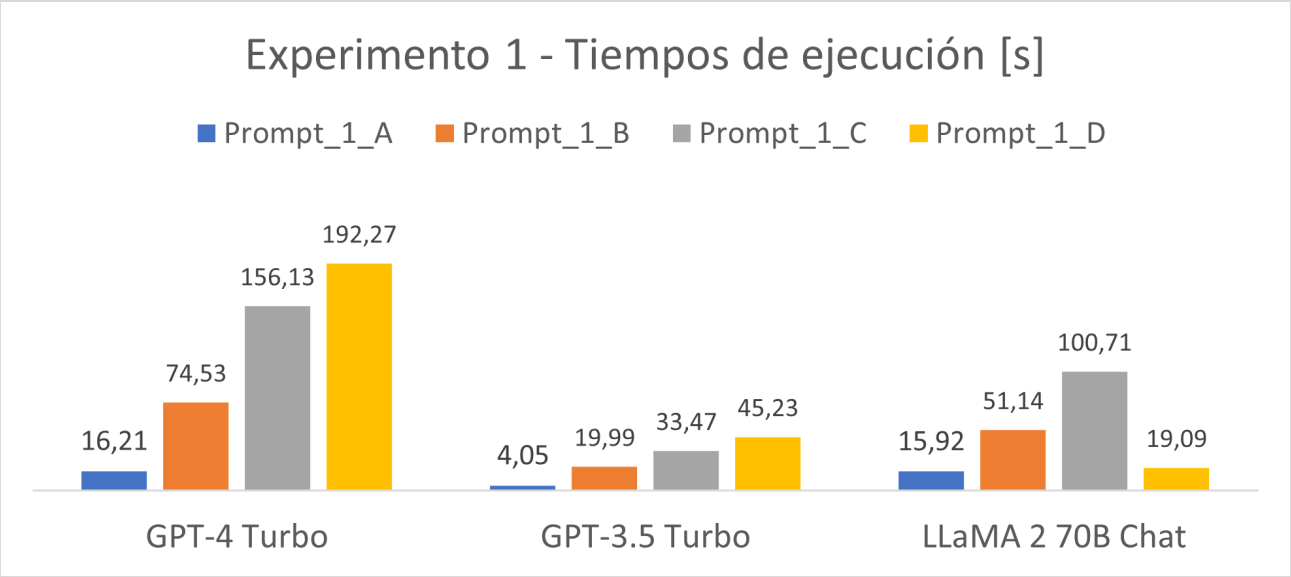


Figura 7.2: Tiempos de ejecución del experimento 1 utilizando cada modelo y *prompt*.

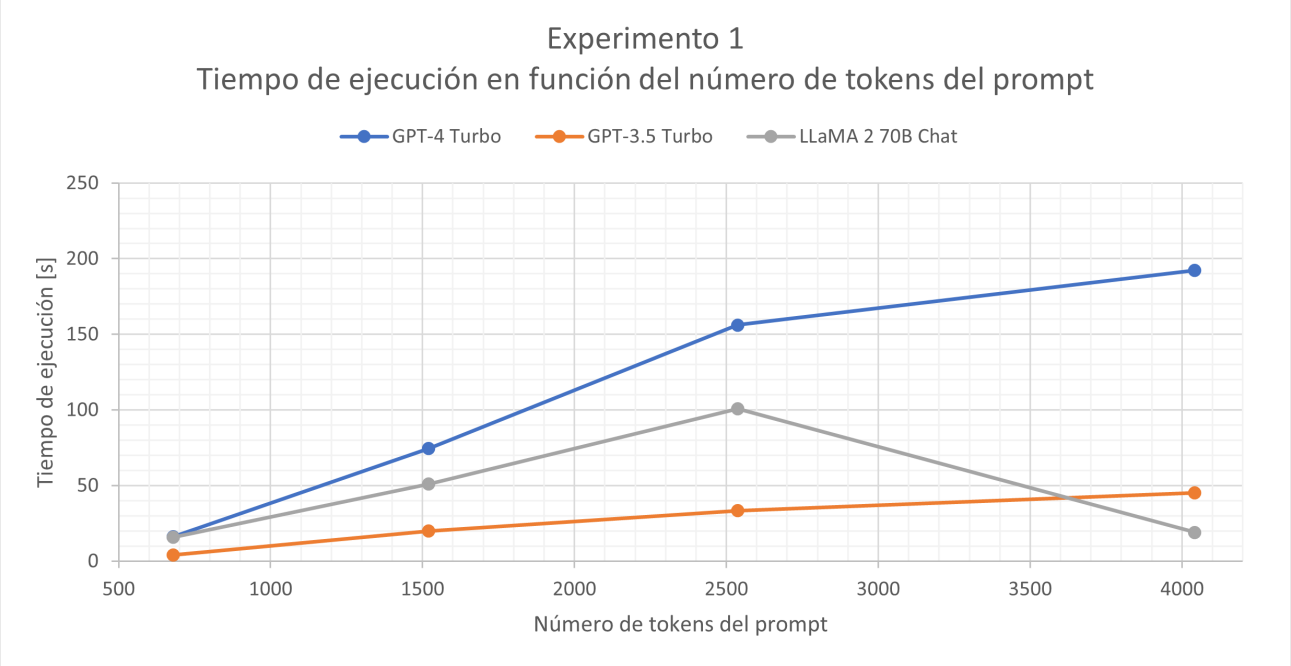


Figura 7.3: Tiempos de ejecución del experimento 1 en función del número de tokens.

corrigiendo relativamente pocos datos transaccionales. Por ejemplo, si volvemos al ejemplo hipotético anterior, con 250 datos transaccionales y 250 datos de referencia, para corregir un conjunto de datos de un millón de registros, necesitaríamos 4000 consultas al LLM, que en tiempo equivaldría a unos 2 días utilizando GPT-3.5 Turbo y unos 9 días si utilizamos GPT-4 Turbo.

Si se utilizara una ventana de contexto más grande, por ejemplo, la ventana máxima de GPT-4 Turbo de 128000 *tokens*, se podrían introducir más datos transaccionales en cada consulta, pero a la vez, el tiempo de cada consulta aumentaría significativamente.

Una vez estudiado el tiempo de ejecución, se ha investigado la exactitud de los modelos. La Figura 7.4 muestra la calidad posterior de los datos transaccionales y la Figura 7.5 presenta el incremento de calidad.

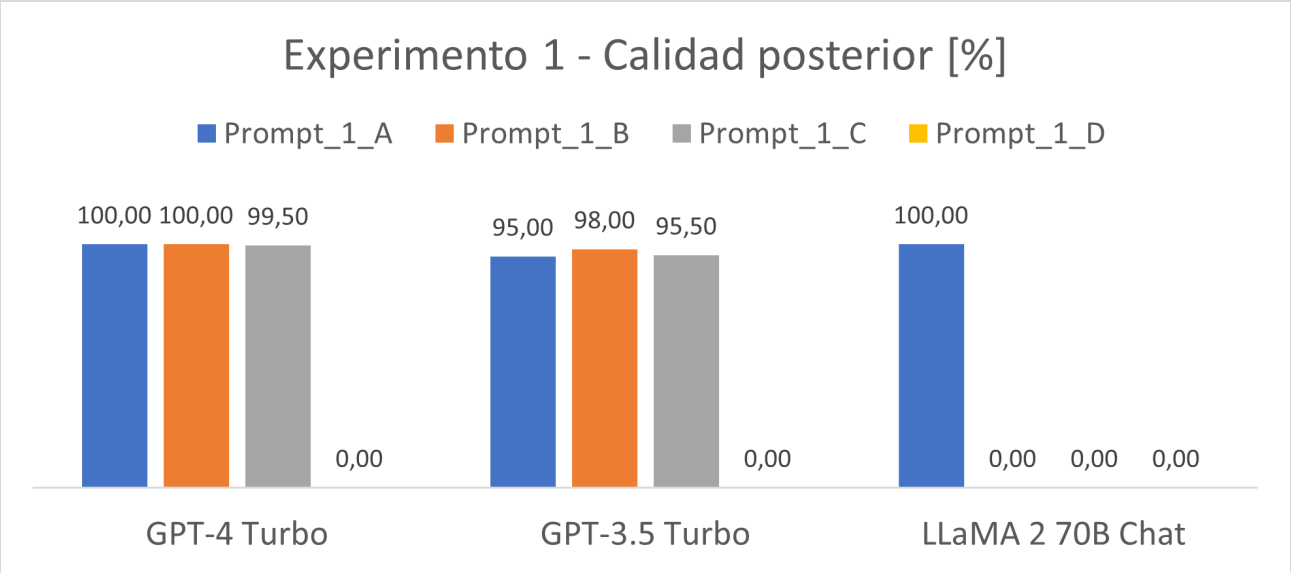


Figura 7.4: Calidad posterior de los datos corregidos del experimento 1.

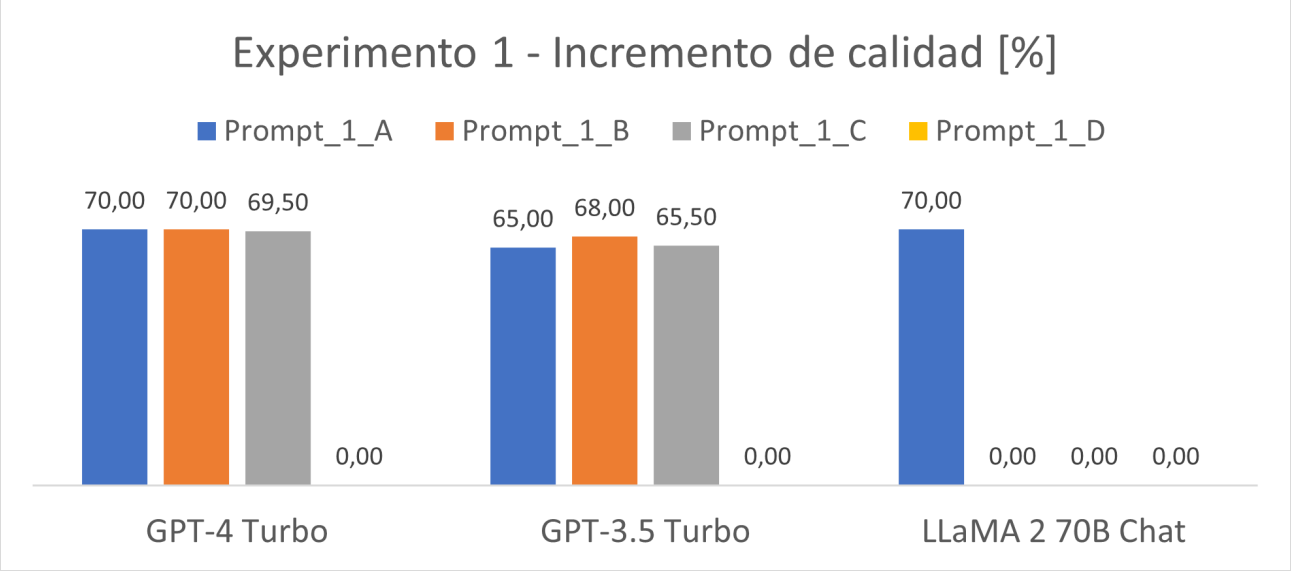


Figura 7.5: Incremento de calidad de los datos corregidos del experimento 1.

Podemos observar que los resultados son bastante extremos, es decir, o son muy buenos, o muy malos. El mejor modelo ha sido GPT-4 Turbo, con calidades posteriores del 100 % para los *prompts Prompt_1_A y Prompt_1_B* y del 99,5 % en el *Prompt_1_C*. Sin embargo, no ha sido capaz de corregir el *Prompt_1_D*, que contenía más datos transaccionales a modificar.

Después, se encuentra GPT-3.5 Turbo, que ha tenido un comportamiento parecido a GPT-4

Turbo, con calidades ligeramente inferiores, de 95 %, 98 % y 95,5 % para los tres primeros *prompts* respectivamente. Aun así, su ratio calidad-tiempo de ejecución es bastante bueno. Por otro lado, LLaMA 2 70B Chat ha demostrado el peor comportamiento. Solamente ha obtenido buenos resultados con un 100 % de calidad posterior con el *Prompt_1_A*.

Así pues, ninguno de los modelos ha sido capaz de corregir todos los *prompts*. Parece existir una relación entre el número de *tokens* del *prompt* de entrada y la calidad de las respuestas. Cuanto mayor es el número de *tokens*, más se confunden los LLMs y cometen errores.

Los errores por los que los modelos no conseguían una respuesta correcta fueron los siguientes:

- **Formato de salida incorrecto:** La respuesta no tenía el formato CSV que se había pedido en el *prompt*, o bien, el modelo añadía palabras o frases explicativas al final del CSV.
- **Número de filas incorrecto:** La respuesta no contenía el mismo número de registros que los datos transaccionales que se le habían introducido en la consulta. Lo más común era que el CSV de la respuesta tuviera menos filas de las que se habían pedido.
- **No completar los valores corregidos:** La respuesta contenía valores ausentes donde debería haber valores corregidos.
- **Combinaciones de los errores anteriores:** La respuesta podía combinar errores de formato, número de filas y valores ausentes.

Cuando aparecían estos errores, no se podía calcular las métricas de calidad y exactitud de las respuestas. Concretamente, GPT-4 Turbo y GPT-3.5 Turbo sufrían del error de número de filas incorrecto. Ambos modelos se olvidaban filas y el CSV resultante contenía menos registros que los datos transaccionales que tenían que corregir. Este error se daba especialmente en los *prompts* con un número de *tokens* más grandes. La causa de este hecho puede ser que cuanta más información tienen que procesar los modelos, más complicado es que sean consistentes con la tarea.

Por otro lado, LLaMA 2 70B Chat sufría de combinaciones de todos los errores. Especialmente, siempre tenía errores de formato de salida, añadiendo frases adicionales. Este hecho puede ser debido a que es un modelo ajustado a tareas de conversación, y siempre tiene la tendencia a explicar la respuesta que da al usuario.

El siguiente análisis ha sido el de la exactitud en datos transaccionales inicialmente erróneos o inicialmente correctos. La Figura 7.6 muestra la exactitud de corrección de datos erróneos y la Figura 7.7 presenta la exactitud de detección y no alteración de datos correctos.

Se puede apreciar un comportamiento similar al análisis anterior. GPT-4 Turbo obtiene los mejores resultados, con exactitudes de corrección de datos erróneos por encima del 99,30 % y de detección de datos del 100 %. GPT-3.5 Turbo muestra una ligera bajada de exactitud en la corrección de datos erróneos con valores de entre 92,86 % y 97 %. Por último, LLaMA 2 70B Chat muestra un 100 %, pero sólo en el primer *prompt*, y falla en el resto.

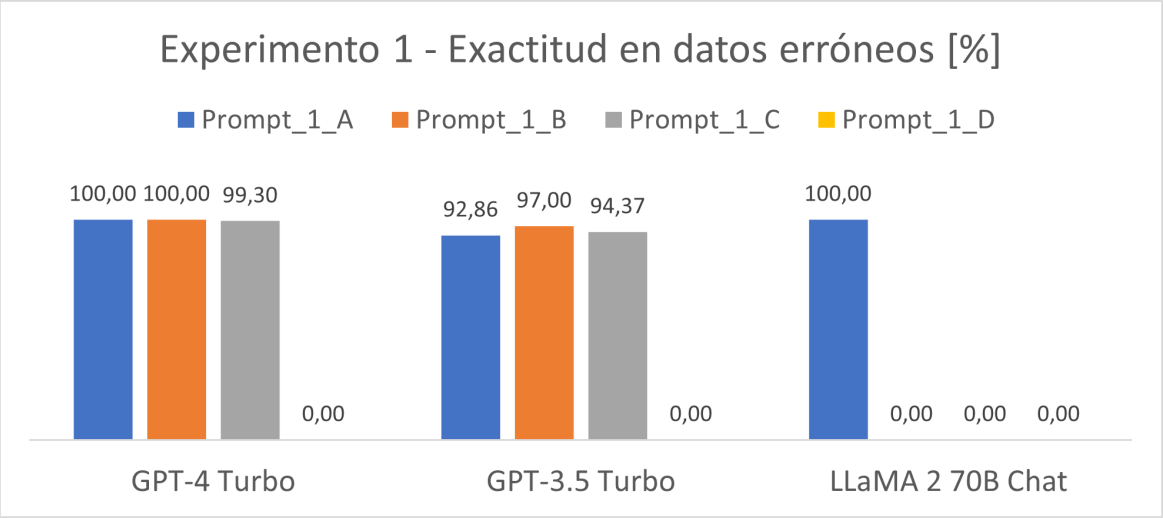


Figura 7.6: Exactitud en la corrección de datos erróneos del experimento 1.

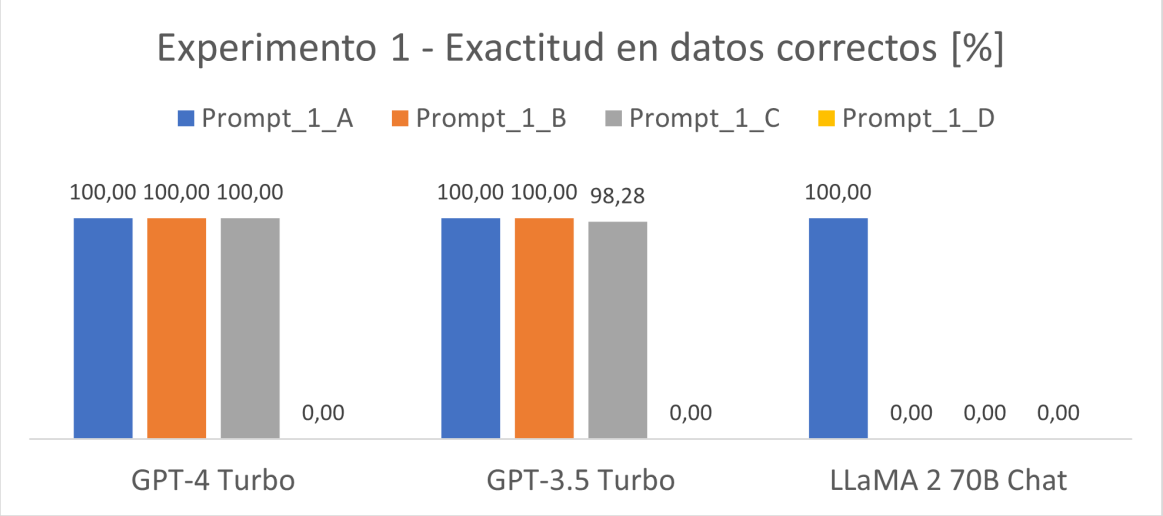


Figura 7.7: Exactitud en la detección de los datos inicialmente correctos del experimento 1.

Una de las conclusiones a las que se puede llegar, es que los modelos tienden a tener menos exactitud en los datos transaccionales inicialmente erróneos, que en los datos inicialmente correctos. Esto puede ser lógico, ya que corregir un dato es más complejo que identificar un dato correcto y dejarlo sin cambios.

## 7.2. Resultados y discusión del experimento 2

Para el experimento 2, de estandarización de datos, también se ha hecho un estudio similar. Se ha utilizado el *prompt* de la sección 4.2.4 y se ha jugado con el número de datos que se le introducen. De manera que se han estudiado cuatro *prompts* con diferentes números de datos de referencia y datos transaccionales. Las combinaciones evaluadas se muestran en la Tabla 7.6.

| *Prompt* | $N_{referencia}$ | $N_{transaccionales}$ | *Tokens* del *prompt* |
|---|---|---|---|
| *Prompt*_2_A | 5 | 10 | 1023 |
| *Prompt*_2_B | 10 | 20 | 1339 |
| *Prompt*_2_C | 25 | 50 | 2292 |
| *Prompt*_2_D | 50 | 100 | 3761 |

Tabla 7.6: *Prompts* evaluados en el experimento 2. Cada *prompt* tiene un número de datos de referencia, $N_{referencia}$, y un número de datos transaccionales, $N_{transaccionales}$. También se presenta el número de *tokens* correspondientes a cada *prompt*.

De manera análoga al experimento 1, se ha realizado una visualización de los cuatro *prompts* estudiados (Tabla 7.6) y se han mostrado en la Figura 7.8.

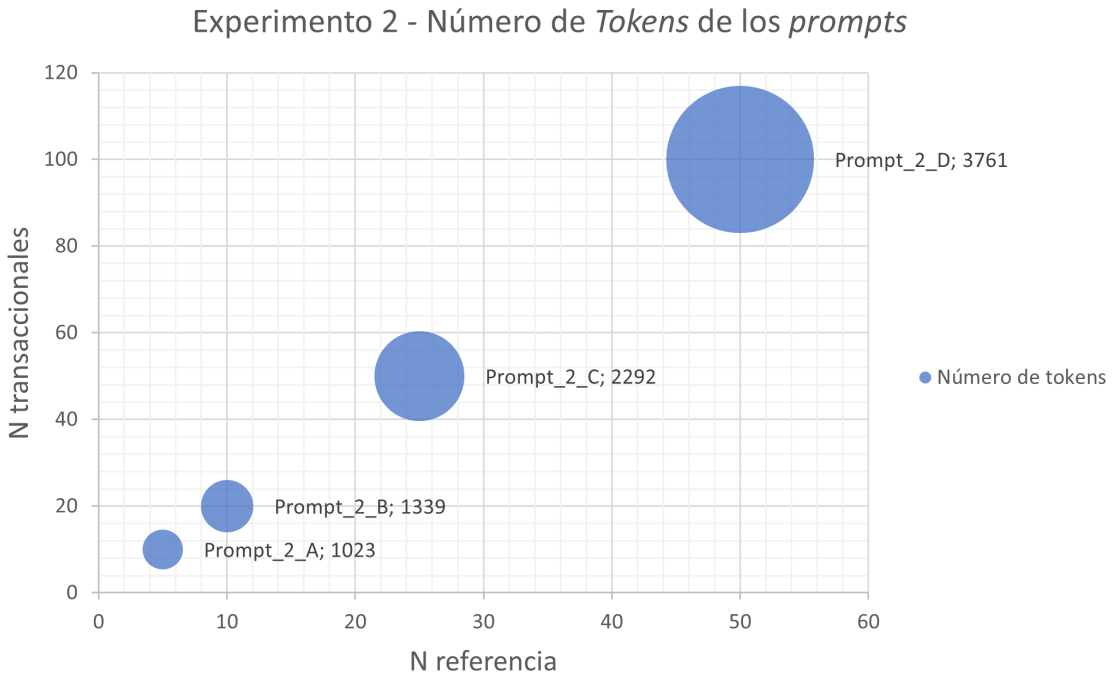


Figura 7.8: Número de *tokens* de cada *prompt* del experimento 2.

La primera observación es que, en el experimento 2, cada registro de los datos de referencia contiene mucha más información que en el experimento 1. Es decir, cada línea del CSV contiene, no solo un campo, sino varios. Esto significa que un solo registro resulta en un número mayor de *tokens* que en el experimento 1. Por lo tanto, con la misma ventana de contexto, se pueden corregir menos datos en una sola consulta.

Con 50 datos de referencia y 100 datos transaccionales, ya se obtienen 3761 *tokens*. Realizando el mismo ejercicio lógico que en el experimento 1, aproximadamente llegamos a los límites de la ventana de contexto con 150 datos (combinando datos de referencia y transaccionales). Esto supone un límite todavía mayor al del experimento 1.

En este caso, si se quisieran utilizar todos los datos de referencia de las calles de Catalunya, que contienen 114,153 registros, estaríamos obligados a adoptar la estrategia de RAG, ya que éstos nunca podrían caber por completo en un *prompt* debido a la restricción de la ventana de contexto.

Por otro lado, si se adoptara la RAG, y todos los datos transaccionales a corregir fueran distintos, podríamos corregir aproximadamente unos 75 datos transaccionales por consulta. Entonces, si quisiéramos corregir un conjunto de datos transaccionales de 1 millón de registros, deberíamos hacer 13,334 consultas. Este número de consultas puede ser realmente muy grande en términos temporales y económicos.

Seguidamente, las Tablas 7.7, 7.8, 7.9 y 7.10 presentan los resultados del experimento 2. Nótese que aquí la calidad inicial siempre es 0.00 % porque, desde un inicio, los datos transaccionales no tienen la estructura correcta que deberían.

| *Prompt*_2_A ($N_{referencia} = 5; N_{transaccionales} = 10$; *tokens* = 1023) | | | | |
|---|---|---|---|---|
| | GPT-4 Turbo | GPT-3.5 Turbo | Gemini Pro | LLaMA 2 70B Chat |
| Calidad inicial [%] | 0.00 | 0.00 | 0.00 | 0.00 |
| Formato de salida | ✓ | ✓ | X | X |
| Número de filas correcto | ✓ | ✓ | X | X |
| Calidad posterior [%] | 100.00 | 100.00 | X | X |
| Incremento de calidad [%] | 100.00 | 100.00 | X | X |
| Tiempo de ejecución [s] | 38.49 | 6.57 | X | 30.61 |

Tabla 7.7: Resultados del experimento 2 con el *Prompt*_2_A ($N_{referencia} = 5; N_{transaccionales} = 10$; *tokens* = 1023)

| *Prompt*_2_B ($N_{referencia} = 10; N_{transaccionales} = 20$; *tokens* = 1339) | | | | |
|---|---|---|---|---|
| | GPT-4 Turbo | GPT-3.5 Turbo | Gemini Pro | LLaMA 2 70B Chat |
| Calidad inicial [%] | 0.00 | 0.00 | 0.00 | 0.00 |
| Formato de salida | ✓ | ✓ | X | X |
| Número de filas correcto | ✓ | ✓ | X | X |
| Calidad posterior [%] | 100.00 | 100.00 | X | X |
| Incremento de calidad [%] | 100.00 | 100.00 | X | X |
| Tiempo de ejecución [s] | 57.29 | 13.83 | X | 82.50 |

Tabla 7.8: Resultados del experimento 2 con el *Prompt*_2_B ($N_{referencia} = 10; N_{transaccionales} = 20$; *tokens* = 1339)

| *Prompt*_2_C ($N_{referencia} = 25$; $N_{transaccionales} = 50$; *tokens* = 2292) | | | | |
|---|---|---|---|---|
| | GPT-4 Turbo | GPT-3.5 Turbo | Gemini Pro | LLaMA 2 70B Chat |
| Calidad inicial [%] | 0.00 | 0.00 | 0.00 | 0.00 |
| Formato de salida | ✓ | ✓ | X | X |
| Número de filas correcto | ✓ | ✓ | X | X |
| Calidad posterior [%] | 100.00 | 100.00 | X | X |
| Incremento de calidad [%] | 100.00 | 100.00 | X | X |
| Tiempo de ejecución [s] | 117.93 | 30.12 | X | 120.08 |

Tabla 7.9: Resultados del experimento 2 con el *Prompt*_2_C ($N_{referencia} = 25$; $N_{transaccionales} = 50$; *tokens* = 2292)

| *Prompt*_2_D ($N_{referencia} = 50$; $N_{transaccionales} = 100$; *tokens* = 3761) | | | | |
|---|---|---|---|---|
| | GPT-4 Turbo | GPT-3.5 Turbo | Gemini Pro | LLaMA 2 70B Chat |
| Calidad inicial [%] | 0.00 | 0.00 | 0.00 | 0.00 |
| Formato de salida | ✓ | ✓ | X | X |
| Número de filas correcto | X | X | X | X |
| Calidad posterior [%] | X | X | X | X |
| Incremento de calidad [%] | X | X | X | X |
| Tiempo de ejecución [s] | 205.45 | 23.19 | X | 122.54 |

Tabla 7.10: Resultados del experimento 2 con el *Prompt*_2_D ($N_{referencia} = 50$; $N_{transaccionales} = 100$; *tokens* = 3761)

El siguiente análisis ha sido el estudio de los tiempos de ejecución del experimento 2. La Figura 7.9 presenta, para cada modelo, el tiempo en segundos que ha tardado el modelo en realizar la inferencia para cada *prompt* utilizado.

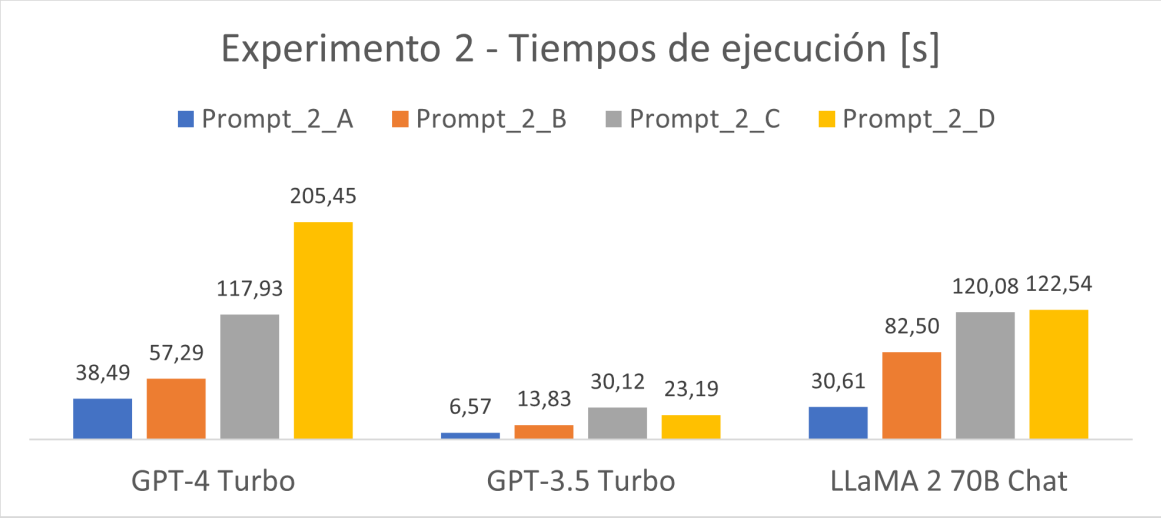


Figura 7.9: Tiempos de ejecución del experimento 2.

Se puede apreciar un comportamiento similar al experimento 1, es decir, a medida que el *prompt* se hace más grande, el tiempo de inferencia aumenta. En este caso, ha habido 2 excepciones que son el GPT-3.5 Turbo y el LLaMA 2 70B Chat con el *Prompt_2_D*, que al no realizar bien la respuesta, han obtenido tiempos más reducidos. Otra vez, GPT-4 Turbo presenta los tiempos de ejecución más elevados, seguido de LLaMA 2 70B Chat y GPT-3.5 Turbo.

También se ha realizado el gráfico comparativo de la Figura 7.10 para evaluar los distintos modelos. Se observa que los tiempos de LLaMA 2 70B Chat son bastante irregulares. Esto puede deberse a fluctuaciones en la latencia de la API de Hugging Face.

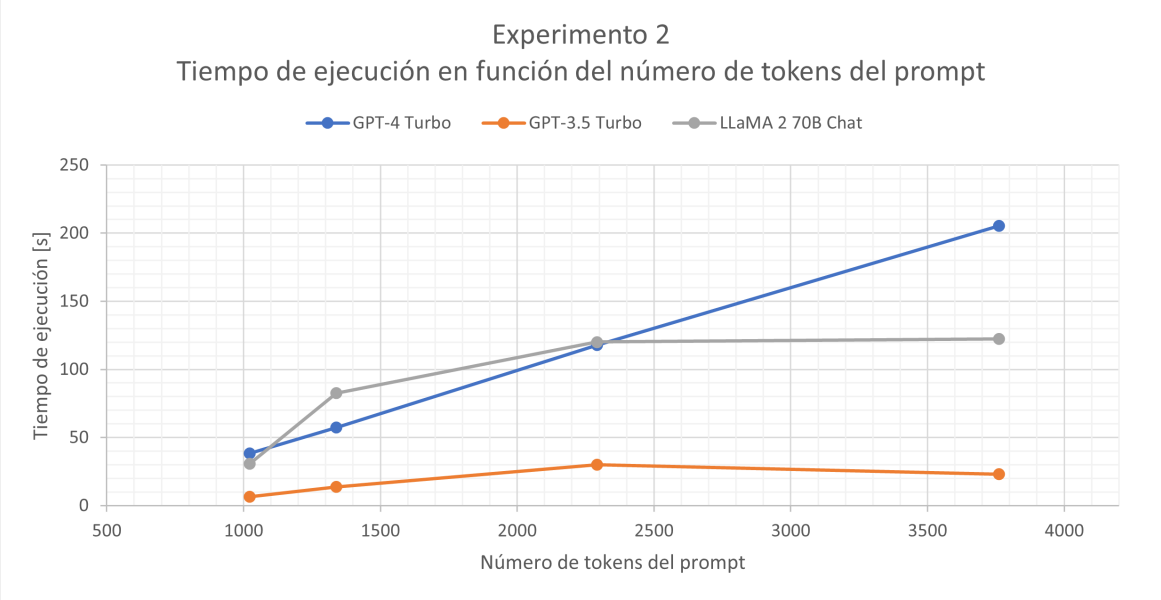


Figura 7.10: Tiempos de ejecución del experimento 2 en función del número de tokens.

Al igual que en el experimento 1, los tiempos pueden parecer ajustados, pero para el conjunto de datos transaccionales de 1 millón de registros que se planteaba anteriormente, 13,334 consultas equivaldrían a unos 31 días de ejecución con GPT-4 Turbo. Esto puede ser demasiado tiempo en determinadas aplicaciones.

A continuación, se estudia la calidad de los datos resultantes del experimento 2. La Figura 7.11 muestra la calidad posterior, que en este caso, coincide con el incremento de calidad. Esto es debido a que la calidad inicial de los datos transaccionales era 0 %, ya que todos los datos eran erróneos al no tener los campos y la estructura correcta.

En este caso, de nuevo los resultados son bastante extremos. Solo GPT-4 Turbo y GPT-3.5 Turbo han obtenido resultados con una calidad del 100 % con los *prompts Prompt_2_A* y *Prompt_2_B*. Todo el resto de combinaciones no han sido capaces de corregir los datos. Además, LLaMA 2 70B Chat no ha sido capaz de realizar ningún resultado correctamente.

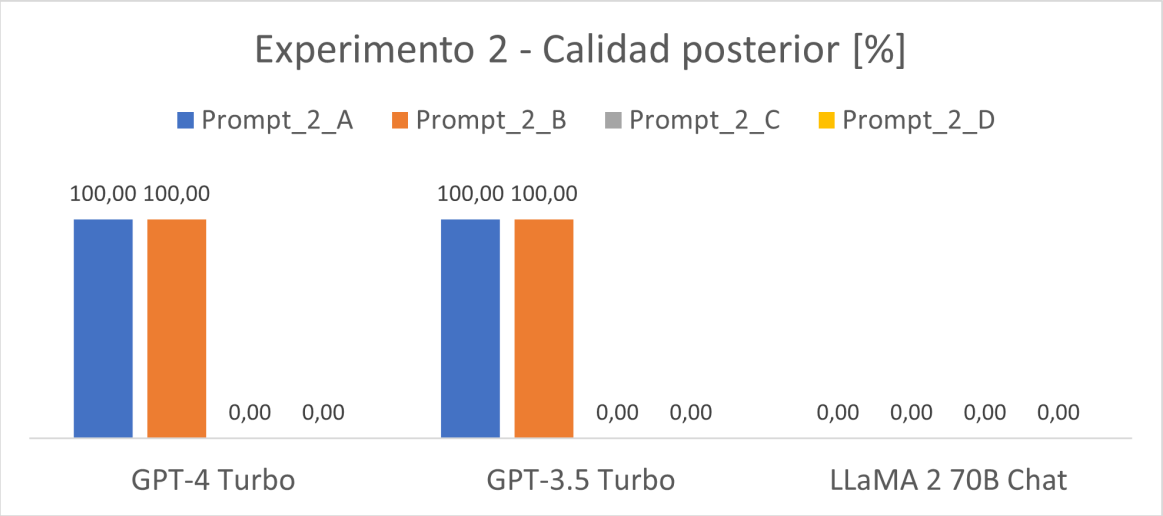


Figura 7.11: Calidad posterior de los datos corregidos del experimento 2.

De nuevo, ambos modelos, GPT-4 Turbo y GPT-3.5 Turbo, cometen el error de olvidarse filas en el resultado. Por otra parte, LLaMA 2 70B Chat comete una combinación de errores incluyendo un mal formato de salida, número de filas incorrecto o valores no completados.

Este empeoramiento generalizado del rendimiento de los modelos puede ser debido a que el experimento 2 es bastante más complejo que el experimento 1, ya que el modelo debe corregir la calle y extraer todos los atributos correctamente de la misma. Con muchos datos en el *prompt*, parece que los modelos confunden datos y olvidan filas que hay que corregir.

## 7.3. Resultados y discusión del experimento 3

De igual forma que en los experimentos anteriores, para el experimento 3 de imputación de datos ausentes, también se ha utilizado su correspondiente *prompt* de la sección 4.3.3 y se ha jugado con el número de datos que se le introducen. De esta manera, se han estudiado cuatro *prompts* con diferentes números de datos de referencia y datos transaccionales. Las combinaciones evaluadas se muestran en la Tabla 7.11.

| *Prompt* | $N_{referencia}$ | $N_{transaccionales}$ | *Tokens* del *prompt* |
|---|---|---|---|
| *Prompt*_3_A | 4 | 8 | 883 |
| *Prompt*_3_B | 10 | 20 | 2169 |
| *Prompt*_3_C | 15 | 30 | 2899 |
| *Prompt*_3_D | 20 | 45 | 3999 |

Tabla 7.11: *Prompts* evaluados en el experimento 3. Cada *prompt* tiene un número de datos de referencia, $N_{referencia}$, y un número de datos transaccionales, $N_{transaccionales}$. También se presenta el número de *tokens* correspondientes a cada *prompt*.

Seguidamente, se ha realizado una visualización de los cuatro *prompts* estudiados (Tabla 7.11) y se han mostrado en la Figura 7.12.

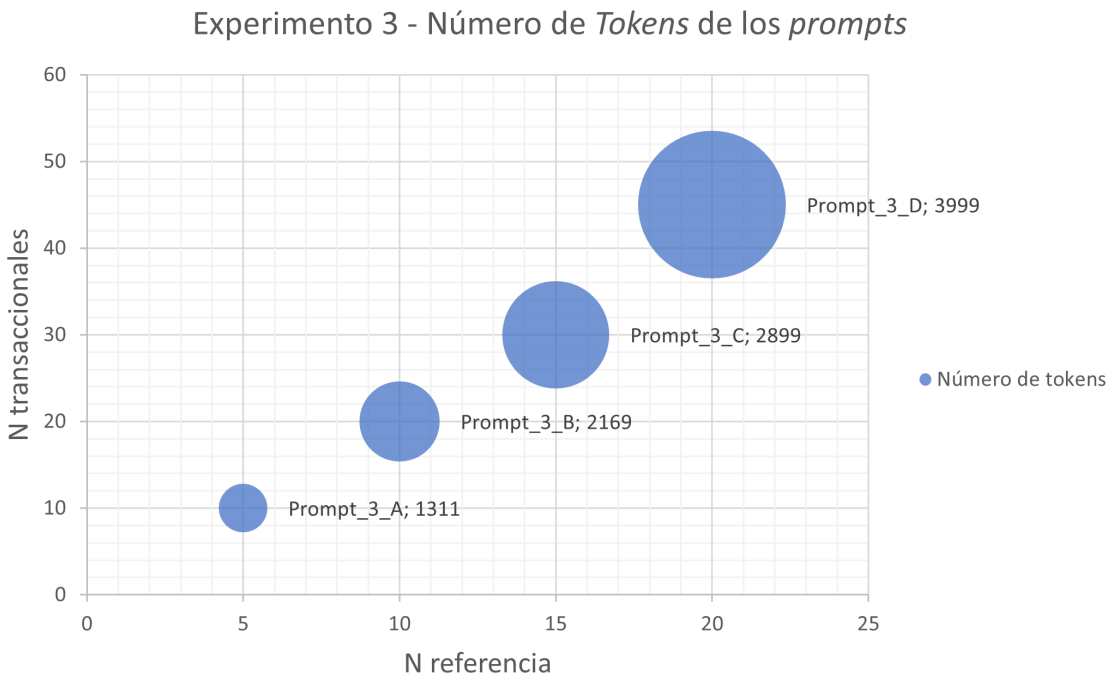


Figura 7.12: Número de *tokens* de cada *prompt* del experimento 3.

De nuevo, se puede observar el número de *tokens* de cada *prompt* en función del número de datos de referencia y datos transaccionales del experimento 3. En este caso, tanto los datos de referencia como los transaccionales, incluyen todos los campos (DNI, Nombre, Apellido 1, Apellido 2, Email y Teléfono), por lo que cada dato se traduce en un número mayor de *tokens*.

Si cada dato se traduce en más *tokens*, con la misma ventana de contexto se podrán corregir menos datos transaccionales en cada consulta. Es decir, a medida que cada dato individual contenga más información, la capacidad del LLM de corregir grandes conjuntos de datos se reduce. Por ejemplo, el *Prompt_3_D*, con 3999 *tokens*, solo contiene 20 datos de referencia y 45 datos transaccionales.

En consecuencia, con unos datos como los del experimento 3 y una ventana de contexto de 4096 *tokens*, solo se podrían introducir unos 65 datos (combinando datos de referencia y transaccionales). Además, si todos los datos transaccionales son diferentes, sólo podríamos incluir 32. Por lo tanto, de nuevo, realizando el ejercicio hipotético de querer corregir un conjunto de datos transaccionales de 1 millón de registros, necesitaríamos aproximadamente 31,250 consultas al LLM. Un valor todavía más elevado que en los experimentos 1 y 2.

En definitiva, a medida que aumenta la complejidad de los datos y la cantidad de información y atributos que contienen, se pueden corregir menos datos por consulta y habría que utilizar un número muy elevado de consultas si queremos corregir un conjunto de datos grande. Este hecho puede ser una limitación que impida escalar la aplicación de LLMs para la corrección de datos a escala masiva.

A continuación, las Tablas 7.12, 7.13, 7.14 y 7.15 presentan los resultados del experimento 3.

| *Prompt*_3_A ($N_{referencia} = 4$; $N_{transaccionales} = 8$; *tokens* = 883) | | | | |
|---|---|---|---|---|
| | GPT-4 Turbo | GPT-3.5 Turbo | Gemini Pro | LLaMA 2 70B Chat |
| Calidad inicial [%] | 30.00 | 30.00 | 30.00 | 30.00 |
| Formato de salida | ✓ | ✓ | X | ✓ |
| Número de filas correcto | X | X | X | X |
| Calidad posterior [%] | X | X | X | X |
| Incremento de calidad [%] | X | X | X | X |
| Exactitud en datos erróneos [%] | X | X | X | X |
| Exactitud en datos correctos [%] | X | X | X | X |
| Tiempo de ejecución [s] | 18.84 | 3.12 | X | 17.37 |

Tabla 7.12: Resultados del experimento 3 con el *Prompt*_3_A ($N_{referencia} = 4$; $N_{transaccionales} = 8$; *tokens* = 883)

| *Prompt*_3_B ($N_{referencia} = 10$; $N_{transaccionales} = 20$; *tokens* = 2169) | | | | |
|---|---|---|---|---|
| | GPT-4 Turbo | GPT-3.5 Turbo | Gemini Pro | LLaMA 2 70B Chat |
| Calidad inicial [%] | 30.00 | 30.00 | 30.00 | 30.00 |
| Formato de salida | ✓ | ✓ | X | X |
| Número de filas correcto | X | X | X | X |
| Calidad posterior [%] | X | X | X | X |
| Incremento de calidad [%] | X | X | X | X |
| Exactitud en datos erróneos [%] | X | X | X | X |
| Exactitud en datos correctos [%] | X | X | X | X |
| Tiempo de ejecución [s] | 45.28 | 5.21 | X | 41.80 |

Tabla 7.13: Resultados del experimento 3 con el *Prompt*_3_B ($N_{referencia} = 10$; $N_{transaccionales} = 20$; *tokens* = 2169)

| *Prompt*_3_C ($N_{referencia} = 15$; $N_{transaccionales} = 30$; *tokens* = 2899) | | | | |
|---|---|---|---|---|
| | GPT-4 Turbo | GPT-3.5 Turbo | Gemini Pro | LLaMA 2 70B Chat |
| Calidad inicial [%] | 30.00 | 30.00 | 30.00 | 30.00 |
| Formato de salida | ✓ | ✓ | X | X |
| Número de filas correcto | X | X | X | X |
| Calidad posterior [%] | X | X | X | X |
| Incremento de calidad [%] | X | X | X | X |
| Exactitud en datos erróneos [%] | X | X | X | X |
| Exactitud en datos correctos [%] | X | X | X | X |
| Tiempo de ejecución [s] | 66.98 | 6.38 | X | 52.81 |

Tabla 7.14: Resultados del experimento 3 con el *Prompt*_3_C ($N_{referencia} = 15$; $N_{transaccionales} = 30$; *tokens* = 2899)

| *Prompt*_3_D ($N_{referencia} = 20; N_{transaccionales} = 45$; *tokens* = 3999) | | | | |
|---|---|---|---|---|
| | GPT-4 Turbo | GPT-3.5 Turbo | Gemini Pro | LLaMA 2 70B Chat |
| Calidad inicial [%] | 30.00 | 30.00 | 30.00 | 30.00 |
| Formato de salida | ✓ | ✓ | X | X |
| Número de filas correcto | X | X | X | X |
| Calidad posterior [%] | X | X | X | X |
| Incremento de calidad [%] | X | X | X | X |
| Exactitud en datos erróneos [%] | X | X | X | X |
| Exactitud en datos correctos [%] | X | X | X | X |
| Tiempo de ejecución [s] | 103.83 | 10.23 | X | 97.95 |

Tabla 7.15: Resultados del experimento 3 con el *Prompt*_3_D ($N_{referencia} = 20; N_{transaccionales} = 45$; *tokens* = 3999)

Llegados a este punto, se presenta el análisis de los tiempos de ejecución del experimento 3. La Figura 7.13 muestra los tiempos para cada modelo y la Figura 7.14 los compara.

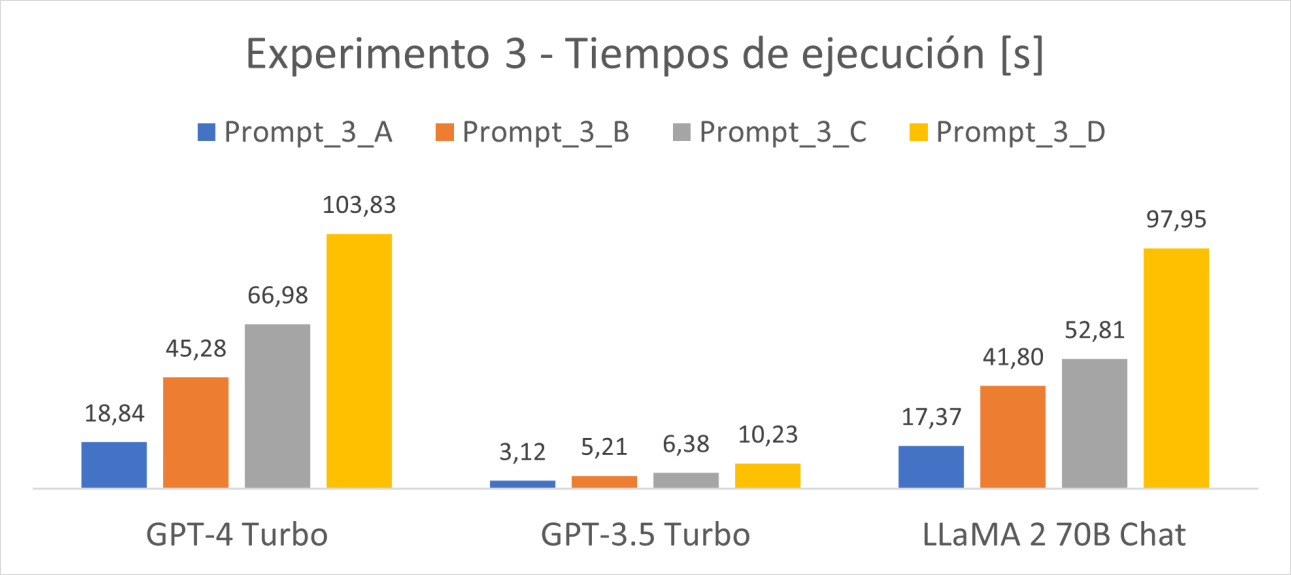


Figura 7.13: Tiempos de ejecución del experimento 3.

Una vez más, como en los experimentos anteriores, se muestra como el tiempo de ejecución aumenta con el número de *tokens* del *prompt*. También, GPT-4 Turbo presenta los tiempos más elevados, seguido de LLaMA 2 70B Chat, y GPT-3.5 Turbo es el modelo más rápido.

Por último, los resultados de calidad del experimento 3 han demostrado que ningún modelo ha sido capaz de resolver la tarea con ningún *prompt*. Esto quiere decir que la tarea era demasiado compleja. De nuevo, GPT-4 Turbo y GPT-3.5 Turbo se olvidaban datos transaccionales, y LLaMA 2 70B Chat tenía una combinación de los errores expuestos en el experimento 1.

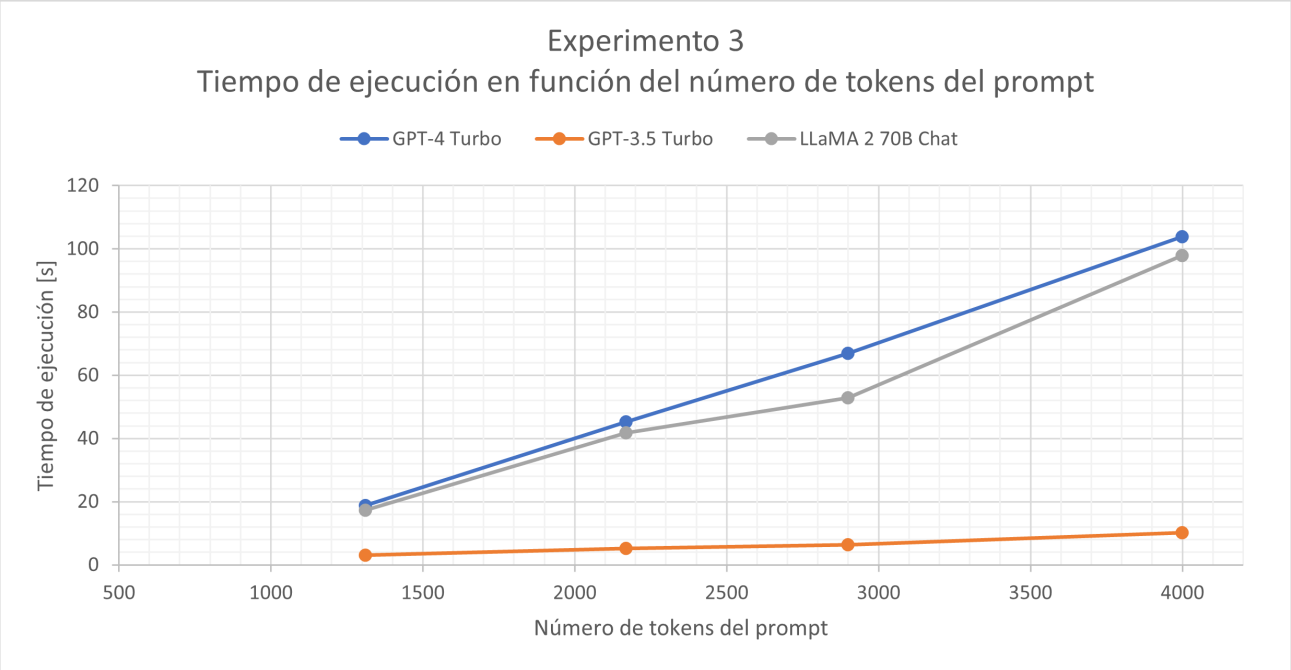


Figura 7.14: Tiempos de ejecución del experimento 3 en función del número de tokens.

## 7.4. Costes de los experimentos

Esta última sección presenta un análisis de los costes económicos que se han pagado para llevar a cabo los experimentos de esta investigación. Concretamente, se han tenido costes asociados a la inferencia de los modelos LLM utilizados: GPT-4 Turbo, GPT-3.5 Turbo y LLaMA 2 70B Chat. En el Capítulo 3, la Tabla 3.3 muestra los precios de las APIs de estos modelos.

Respecto al modelo LLaMA 2 70B Chat, simplemente se han pagado 9$ para hacer uso de la API de Hugging Face Pro. Para realizar este tipo de pruebas es suficiente, pero si se quisiera escalar la solución, se debería utilizar otra solución de inferencia.

En cuanto a la API de OpenAI, tal como muestra la Figura 7.15, se compraron créditos por valor de 10$, que después de incluir tasas, acabaron siendo 12,10$. Sin embargo, el coste de los modelos de OpenAI ha resultado ser más barato de lo esperado. En particular, el coste de GPT-4 Turbo ha sido de 1,29$ y el de GPT-3.5 Turbo de 0,06$.

Estos costes pueden parecer baratos, pero los experimentos planteados han sido pequeñas pruebas comparado con lo que podrían ser en un caso de uso real con millones de datos transaccionales a corregir.

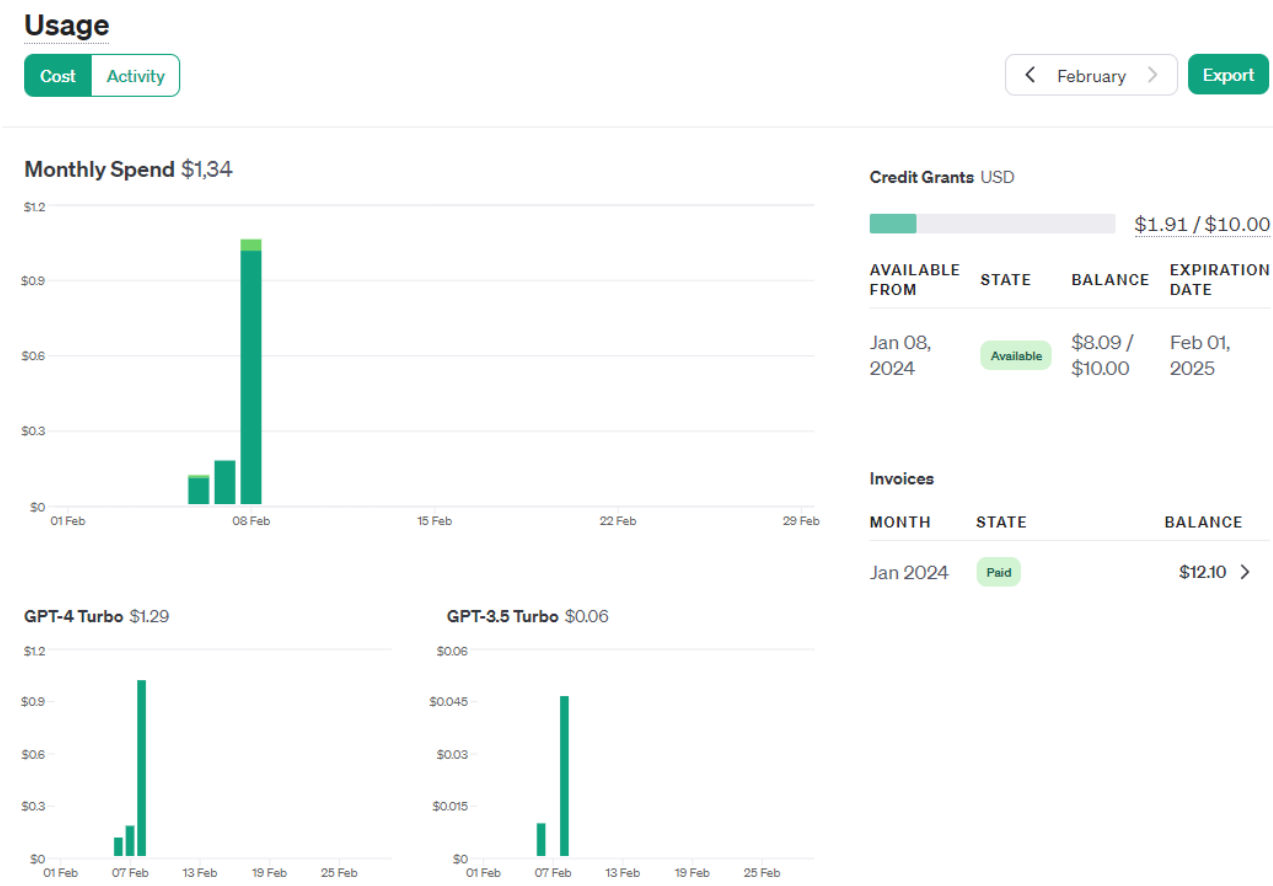


Figura 7.15: Resumen de costes de GPT-4 Turbo y GPT-3.5 Turbo.

# Capítulo 8

# Conclusiones y trabajos futuros

Este último capítulo resume los hallazgos más significativos de esta investigación y propone diferentes líneas de trabajo por donde continuar el estudio. La sección 8.1 presenta las conclusiones del proyecto y la sección 8.2 los posibles trabajos futuros.

## 8.1. Conclusiones

En este trabajo se han estudiado los modelos LLM, incluyendo su historia y su estado del arte. Se ha visto que los mejores modelos utilizan arquitecturas basadas en *transformers* y son pre-entrenados con masivas cantidades de datos. Seguidamente se les ajusta para realizar tareas más específicas, como tener conversaciones naturales. Aun así, todavía tienen retos y limitaciones por superar, por ejemplo, las alucinaciones, las inconsistencias en las respuestas, la explicabilidad o las implicaciones sociales y éticas.

También se ha investigado sobre la ingeniería de *prompts* y cómo optimizar las instrucciones que se proporcionan a este tipo de modelos. Los *prompts* deben ser concisos, estar alineados con la tarea y tener suficiente contexto. Además, se han aplicado distintas técnicas, estrategias y buenas prácticas para realizar los *prompts* de este trabajo.

Por otro lado, se han estudiado los principales problemas de calidad del dato que tienen las empresas, organizaciones y gobiernos, y se ha investigado sobre la aplicación de LLMs para tareas de calidad del dato. Se han encontrado potenciales tareas a resolver, pero a la vez, algunos estudios ya alertaban de problemas de fiabilidad y de coste computacional.

El siguiente paso ha sido el planteamiento de una metodología para realizar experimentos de calidad del dato con LLMs. El resultado ha sido un flujo, desde la importación de datos hasta la evaluación de resultados, que se puede aplicar y reutilizar para distintas tareas.

Seguidamente, se han planteado tres experimentos para comprobar el potencial de los LLMs. El primero ha explorado la corrección de valores en datos transaccionales, el segundo ha probado la viabilidad de la estandarización de campos y atributos, y el tercero ha explorado si es posible realizar

imputación de datos ausentes en un conjunto de datos transaccionales.

Una vez ejecutados los experimentos y analizados los resultados, se han obtenido varias conclusiones. Primero, se ha visto que dependiendo del tipo de dato transaccional a corregir, el número de *tokens* necesarios para representar un dato puede ser distinto. Por ejemplo, si el dato a corregir es simplemente un valor, éste se traducirá en pocos *tokens*. Por el contrario, si el dato es muy complejo, contiene muchos atributos o tiene valores de gran longitud, éste se traducirá en una gran cantidad de *tokens*.

Este hecho puede parecer irrelevante, pero si cada dato transaccional a corregir, necesita muchos *tokens* y la ventana de contexto del modelo LLM es fija, para una misma consulta, un LLM puede corregir una mayor cantidad de datos simples que de datos complejos o largos. Por ejemplo, con la misma venta de contexto de 4096 *tokens*, el experimento 1 ha podido incluir 500 datos (filas) en el *prompt*, el experimento 2, 150 datos, y el experimento 3, 65 datos. Dependiendo del caso de uso, este hecho puede llegar a afectar a la viabilidad de la tarea.

Por otro lado, existen casos en los que hay una cantidad enorme de datos de referencia, como en el experimento 2, con 114,153 registros de calles. Así pues, es muy común que los datos de referencia no quepan en un solo *prompt*. En estos casos, se ha llegado a la conclusión de que es necesario aplicar la técnica de RAG. Los datos de referencia se transformarían en *embeddings* y se guardarían en una base de datos vectorial. Después, el usuario haría una consulta con ciertos datos transaccionales a corregir, y se buscarían los datos de referencia correspondientes. A partir de ambos, se construiría el *prompt* y el LLM corregiría los datos transaccionales.

Sin embargo, tal como se ha comentado, existe un límite de datos a incluir en un *prompt*, por lo que si el conjunto de datos transaccionales a corregir es muy grande, se deberán realizar muchas consultas al LLM. Así pues, un gran número de consultas implicaría elevados tiempos de inferencia y costes económicos, llegando a comprometer la viabilidad temporal y económica de la tarea.

Respecto a los tiempos de inferencia, se han obtenido valores del orden de magnitud de segundos y pocos minutos. No obstante, los experimentos realizados han corregido relativamente pocos datos, del orden de decenas y pocos cientos de datos transaccionales. En consecuencia, si se corrigieran conjuntos de datos transaccionales de millones de registros, estos tiempos se dispararían hasta llegar a ordenes de magnitud de días. Este podría ser otro inconveniente a la hora de aplicar estos modelos.

En cuanto a la calidad y exactitud de las respuestas de los modelos, GPT-4 Turbo ha destacado como el mejor modelo, seguido de GPT-3.5 Turbo y LLaMA 2 70B Chat. Aun así, los resultados han sido muy extremos. Es decir, o bien, se obtenían exactitudes entorno al 90-100 %, o bien, las respuestas no eran generadas correctamente y no se podían evaluar. Además, se ha demostrado que para tareas sencillas (como el experimento 1) y *prompts* cortos, los LLMs obtienen buenos resultados, pero para tareas complejas (como el experimento 3) o utilizando *prompts* largos, los modelos no han sido capaces de corregir correctamente los datos. Por lo tanto, dependiendo de la tarea y de la longitud del *prompt*, puede ser que no sea viable utilizar un LLM.

También, se ha observado que, a mayor número de *tokens* tiene un *prompt*, mayor cantidad

de fallos realiza el LLM. En particular, los errores encontrados han sido fallos por un formato de salida incorrecto, por un número de filas incorrecto, por dejar datos en blanco y combinaciones de los anteriores.

En cuanto el coste económico de los experimentos, ha sido de unos pocos dólares, pero en caso de que se tuvieran que corregir millones de datos transaccionales, se deberían hacer miles de consultas y el coste se dispararía hasta los miles de dólares.

En conclusión, la aplicación de LLMs en tareas de calidad del dato puede obtener buenos resultados en determinados escenarios muy concretos. Por ejemplo, cuando la cantidad de datos transaccionales a corregir es pequeña y la tarea es sencilla. Sin embargo, para grandes cantidades de datos o tareas complejas, surgen dificultades relacionadas con los tiempos de ejecución, los costes económicos y la fiabilidad de las respuestas.

## 8.2. Trabajos futuros

Este trabajo ha sido un primer paso en la investigación del uso de LLMs para tareas de calidad del dato. A partir de aquí se puenden abrir varias líneas de estudio para continuar el proyecto.

Primero, dada la importancia de la ventana de contexto en los experimentos, se podría estudiar la aplicación de ventanas de contexto más grandes, por ejemplo, con GPT-4 Turbo y su ventana de 128,000 *tokens*. Esto permitiría la introducción de más datos en cada *prompt*. Aun así, se tendría que comprobar la fiabilidad de las respuestas con *prompts* tan grandes. También se podrían estudiar los tiempos de inferencia y ver si utilizar *prompts* grandes es rentable.

Por otro lado, también se podrían estudiar otros *prompts* para los experimentos realizados. Podría darse el caso de que, por ejemplo, el experimento 3, utilizando otro *prompt* más optimizado, si que funcionara.

Otra línea de trabajo es probar más tareas de calidad del dato. Por ejemplo, se podrían estudiar la eliminación de duplicados, la clasificación de valores en categorías, la identificación de discrepancias entre varias fuentes, el descubrimiento de valores fuera de rango (*outliers*), entre otras tareas.

También, se podría estudiar si es posible reducir los problemas de fiabilidad e inconsistencias de los LLMs. En particular, recientemente la API de OpenAI ha introducido un modo llamado JSON, con el cual las respuestas se devuelven en formato JSON, y un modo con un parámetro semilla que mejora la reproducibilidad de las respuestas. Se podría investigar si estos modos mejoran los resultados obtenidos.

Finalmente, como se ha visto en el estado del arte, los LLMs están en plena evolución y, posiblemente, en los próximos años mejoren sus capacidades y surjan nuevas aplicaciones y líneas de investigación.

# Referencias


Arnes, J. I., y Horsch, A. (2023). Schema-Based Priming of Large Language Model for Data Object Validation Compliance. *Journal of Systems and Software*. Descargado de https://papers.ssrn.com/sol3/papers.cfm?abstract_id=4453361

Bahdanau, D., y Cho, K. (2014). Neural Machine Translation by Jointly Learning to Align and Translate. *arXiv*. Descargado de https://arxiv.org/pdf/1409.0473v1.pdf

Bolding, Q., Liao, B., Denis, B. J., Luo, J., y Monz, C. (2023). Ask Language Model to Clean Your Noisy Translation Data. *arXiv*. Descargado de https://arxiv.org/pdf/2310.13469.pdf

Bsharat, S. M., Myrzakhan, A., y Shen, Z. (2023). Principled Instructions Are All You Need for Questioning LLaMA-1/2, GPT-3.5/4. *arXiv*. Descargado de https://arxiv.org/pdf/2312.16171.pdf

Buzzelli, B. (2022). *Data Quality Engineering in Financial Services*. O'Reilly Media, Inc. Descargado de https://learning.oreilly.com/library/view/data-quality-engineering/9781098136925/

Chen, Z., Cao, L., Madden, S., Fan, J., Tang, N., Gu, Z., ... Kraska, T. (2023, 10). SEED: Simple, Efficient, and Effective Data Management via Large Language Models. *arXiv*. Descargado de https://arxiv.org/abs/2310.00749v1

Eryurek, E., Gilad, U., Lakshmanan, V., Kibunguchy-Grant, A., y Ashdown, J. (2021). *Data Governance: The Definitive Guide*. O'Reilly Media, Inc. Descargado de https://learning.oreilly.com/library/view/data-governance-the/9781492063483/

Fernandez, R. C., Elmore, A. J., Franklin, M. J., Krishnan, S., y Tan, C. (2023, 7). How Large Language Models Will Disrupt Data Management. *Proceedings of the VLDB Endowment*, *16*(11), 3302–3309. Descargado de https://dl.acm.org/doi/10.14778/3611479.3611527 doi: 10.14778/3611479.3611527

Frąckiewicz, M. (2023). *The Role of ChatGPT in Enhancing Data Quality and Integrity.* Descar-

gado de https://ts2.pl/en/the-role-of-chatgpt-in-enhancing-data-quality-and-integrity/#gsc.tab=0

Gao, Y., Xiong, Y., Gao, X., Jia, K., Pan, J., Bi, Y., ... Wang, H. (2024). Retrieval-Augmented Generation for Large Language Models: A Survey. *arXiv*. Descargado de https://arxiv.org/pdf/2312.10997.pdf

Generalitat de Catalunya. (2023). *Dades de Referència.* Descargado de https://canigo.ctti.gencat.cat/plataformes/dadesref/dadesref/

Generalitat de Catalunya. (2024). *Dades obertes de Catalunya.* Descargado de https://analisi.transparenciacatalunya.cat/ca/

Google Gemini Team. (2023). Gemini: A Family of Highly Capable Multimodal Models. *Google DeepMind*. Descargado de https://storage.googleapis.com/deepmind-media/gemini/gemini_1_report.pdf

Greyling, C. (2023). *12 Prompt Engineering Techniques.* Descargado de https://cobusgreyling.medium.com/12-prompt-engineering-techniques-644481c857aa

Hadi, M. U., Al-Tashi, Q., Qureshi, R., Shah, A., Muneer, A., Irfan, M., ... Mirjalili, S. (2023). A survey on large language models: Applications, challenges, limitations, and practical usage. *TechRxiv*. Descargado de https://www.techrxiv.org/users/618307/articles/682263-large-language-models-a-comprehensive-survey-of-its-applications-challenges-limitations-and-future-prospects doi: 10.36227/techrxiv.23589741.v4

Hawker, R. (2023). *Practical Data Quality*. Packt Publishing Ltd. Descargado de https://learning.oreilly.com/library/view/practical-data-quality/9781804610787/

Hegselmann, S., Buendia, A., Lang, H., Agrawal, M., Jiang, X., y Sontag, D. (2023). TabLLM: Few-shot Classification of Tabular Data with Large Language Models. En *Proceedings of the 26th international conference on artificial intelligence and statistics (aistats).* Valencia, Spain.

Hellerstein, J. M., Rattenbury, T., Heer, J., Kandel, S., y Carreras, C. (2017). *Principles of Data Wrangling*. O'Reilly Media, Inc. Descargado de https://learning.oreilly.com/library/view/principles-of-data/9781491938911/

Institut Cartogràfic i Geològic de Catalunya (ICGC). (2023). *Nom de carrers de Catalunya | Dades obertes de Catalunya.* Descargado de https://analisi.transparenciacatalunya.cat/Urbanisme-infraestructures/Nom-de-carrers-de-Catalunya/2e2b-4ksy/about_data

Institut d'Estadística de Catalunya (IDESCAT), y Institut Cartogràfic i Geològic de Catalunya (ICGC). (2023). *Municipis Catalunya Geo | Dades obertes de Catalunya.* Descargado de https://analisi.transparenciacatalunya.cat/Urbanisme-infraestructures/Municipis-Catalunya-Geo/9aju-tpwc/about_data

Jaimovitch-López, G., Ferri, C., Hernández-Orallo, J., Martínez-Plumed, F., y Ramírez-Quintana, M. J. (2022, 6). Can language models automate data wrangling? *Machine Learning*, *112*(6), 2053–2082. Descargado de https://link.springer.com/article/10.1007/s10994-022-06259-9

King, T., y Schwarzenbach, J. (2020). *Managing Data Quality*. BCS, The Chartered Institute for IT. Descargado de https://learning.oreilly.com/library/view/managing-data-quality/9781780174617/

Li, C. (2020). *OpenAI's GPT-3 Language Model: A Technical Overview.* Descargado de https://lambdalabs.com/blog/demystifying-gpt-3

Li, N., Kang, B., y De Bie, T. (2023, 4). SkillGPT: a RESTful API service for skill extraction and standardization using a Large Language Model. *arXiv*. Descargado de https://arxiv.org/abs/2304.11060v2

Liu, Y., He, H., Han, T., Zhang, X., Liu, M., Tian, J., ... Ge, B. (2024). Understanding LLMs: A Comprehensive Overview from Training to Inference. *Elsevier*.

Lowe, R., y Leike, J. (2022). *Aligning language models to follow instructions.* Descargado de https://openai.com/research/instruction-following#sample5

Mayo, M. (2024). *Prompt Engineering 101: Mastering Effective LLM Communication - KDnuggets.* Descargado de https://www.kdnuggets.com/prompt-engineering-101-mastering-effective-llm-communication

McGregor, S. E. (2021). *Practical Python Data Wrangling and Data Quality*. O'Reilly Media, Inc. Descargado de https://learning.oreilly.com/library/view/practical-python-data/9781492091493/

Menthière, B. d. (2023, 6). *How we use GPT to improve data quality and entity extraction.* Descargado de https://medium.com/inex-blog/how-we-use-gpt-to-improve-data-quality-and-entity-extraction-b35e21b05ef4

Mertz, D. (2021). *Cleaning Data for Effective Data Science*. Packt Publishing. Descargado de https://learning.oreilly.com/library/view/cleaning-data-for/

9781801071291/

Moses, B., Gavish, L., y Vorwerck, M. (2022). *Data Quality Fundamentals*. O'Reilly Media, Inc. Descargado de https://learning.oreilly.com/library/view/data-quality-fundamentals/9781098112035/

Narayan, A., Chami, I., Orr, L., Arora, S., y Ré, C. (2022). Can Foundation Models Wrangle Your Data?; Can Foundation Models Wrangle Your Data? *arXiv*. Descargado de https://arxiv.org/pdf/2205.09911.pdf

OpenAI. (2023a). GPT-4 Technical Report. *arXiv*. Descargado de https://cdn.openai.com/papers/gpt-4.pdf

OpenAI. (2023b). *New models and developer products announced at DevDay.* Descargado de https://openai.com/blog/new-models-and-developer-products-announced-at-devday

OpenAI. (2023c). *Prompt engineering - OpenAI API.* Descargado de https://platform.openai.com/docs/guides/prompt-engineering

OpenAI. (2023d). *Tokenizer - OpenAI Platform.* Descargado de https://platform.openai.com/tokenizer

Pai, A. (2023). *How GenerativeAI and LLM's will Improve Data Quality in Banking - 6 High Impact Areas.* Descargado de https://www.linkedin.com/pulse/generativeai-llms-applied-data-quality-improvement-6-high-ahson-pai/

Peeters, R., y Bizer, C. (2023, 10). Entity Matching using Large Language Models. *arXiv*. Descargado de https://arxiv.org/abs/2310.11244v1

Pichai, S., y Hassabis, D. (2023). *Introducing Gemini: Google's most capable AI model yet.* Descargado de https://blog.google/technology/ai/google-gemini-ai/#sundar-note

Ram, N. (2023, 6). *From Chaos to Clarity: Streamlining Data Cleansing Using Large Language Models.* Descargado de https://towardsdatascience.com/from-chaos-to-clarity-streamlining-data-cleansing-using-large-language-models-a539fa0b2d90

Ray, P. P. (2023, 1). ChatGPT: A comprehensive review on background, applications, key challenges, bias, ethics, limitations and future scope. *Internet of Things and Cyber-Physical Systems*, *3*, 121–154. doi: 10.1016/J.IOTCPS.2023.04.003

Scikit-Learn. (2024). *Metrics and scoring: quantifying the quality of predictions.*

Descargado de https://scikit-learn.org/stable/modules/model_evaluation.html#accuracy-score

Sharma, A., Li, X., Guan, H., Sun, G., Zhang, L., Wang, L., ... Zou, J. (2023, 9). Automatic Data Transformation Using Large Language Model: An Experimental Study on Building Energy Data. *arXiv*. Descargado de https://arxiv.org/abs/2309.01957v2

Southekal, P. (2023). *Data Quality*. Wiley. Descargado de https://learning.oreilly.com/library/view/data-quality/9781394165230/

Stack Overflow. (2023). *Gemini Pro API Blocking Replies - Stack Overflow.* Descargado de https://stackoverflow.com/questions/77723993/gemini-pro-api-blocking-replies

Stack Overflow. (2024). *VertexAIException - list index out of range Error when calling Gemini-Pro API - Stack Overflow.* Descargado de https://stackoverflow.com/questions/77930819/vertexaiexception-list-index-out-of-range-error-when-calling-gemini-pro-api

Stanley, J., y Schwartz, P. (2024). *Automating Data Quality Monitoring at Scale*. O'Reilly Media, Inc. Descargado de https://learning.oreilly.com/library/view/automating-data-quality/9781098145927/

Touvron, H., Martin, L., y Stone, K. (2023). *Llama 2: Open Foundation and Fine-Tuned Chat Models | Research - AI at Meta.* Descargado de https://ai.meta.com/research/publications/llama-2-open-foundation-and-fine-tuned-chat-models/

UK Government Data Quality Hub. (2021, 6). *Meet the data quality dimensions.* Descargado de https://www.gov.uk/government/news/meet-the-data-quality-dimensions

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... Polosukhin, I. (2017). Attention Is All You Need. En *31st conference on neural information processing systems.* Descargado de https://papers.nips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

Wei, J., Tay, Y., Bommasani, R., Raffel, C., Zoph, B., Borgeaud, S., ... Fedus, W. (2022). Emergent Abilities of Large Language Models. *Transactions on Machine Learning Research*. Descargado de https://arxiv.org/pdf/2206.07682.pdf

Wu, T., He, S., Liu, J., Sun, S., Liu, K., Han, Q. L., y Tang, Y. (2023, 5). A Brief Overview of ChatGPT: The History, Status Quo and Potential Future Development. *IEEE/CAA Journal of Automatica Sinica*, *10*(5), 1122–1136. doi: 10.1109/JAS.2023.123618

Yenduri, G., Murugan, R., Selvi Govardanan, C., Srivastava, G., Selvi, C. G., Kumar Reddy Maddikunta, P., ... Reddy Gadekallu, T. (2023). GPT (Generative Pre-trained Transformer)-A Comprehensive Review on Enabling Technologies, Potential Applications, Emerging Challenges, and Future Directions. *arXiv*. Descargado de `https://arxiv.org/pdf/2305.10435.pdf`

Zhang, H., Dong, Y., Xiao, C., y Oyamada, M. (2023, 8). Large Language Models as Data Preprocessors. *arXiv*. Descargado de `https://arxiv.org/abs/2308.16361v1`

Zhao, W. X., Zhou, K., Li, J., Tang, T., Wang, X., Hou, Y., ... Wen, J.-R. (2023). A Survey of Large Language Models. *arXiv*. Descargado de `https://arxiv.org/pdf/2303.18223.pdf`

# Anexo A

# Problemas con las APIs de inferencia

## A.1. Gemini Pro

Tal y como se ha mostrado en las tablas del Capítulo 7, Gemini Pro no fue capaz de inferir ninguna respuesta en ninguno de los tres experimentos. Esto fue debido a que se tuvieron problemas con la API. Según el propio Google, en el momento de realización de esta investigación, la API de Gemini Pro estaba en *preview*, tal y como muestra la Figura A.1. Por lo tanto, probablemente este sea el motivo de los errores. Concretamente, al querer realizar las inferencias, se han obtenido dos tipos de errores.

**Error de tipo 1: *'ResponseBlockedError: The response was blocked'***

Este error bloqueaba la generación de la respuesta cuando se le pedía la inferencia de un *prompt*. Parece estar relacionado con la capa de seguridad que tiene Gemini Pro, la cual detecta si un *prompt* tiene contenido peligroso, dañino, de odio o sexual. Sin embargo, ninguno de los *prompts* de los experimentos presentaban estas características.

Este error no es un caso aislado, ya que también le sucedió a varias personas (Stack Overflow, 2023). La Figura A.2 muestra la solución que se intentó para que Gemini Pro funcionase. Se trataba de configurar los parámetros de seguridad del modelo para que no bloquease ningún tipo de *prompt* por motivos de seguridad. Sin embargo, aun implementando estos parámetros, el modelo seguía bloqueando las consultas.

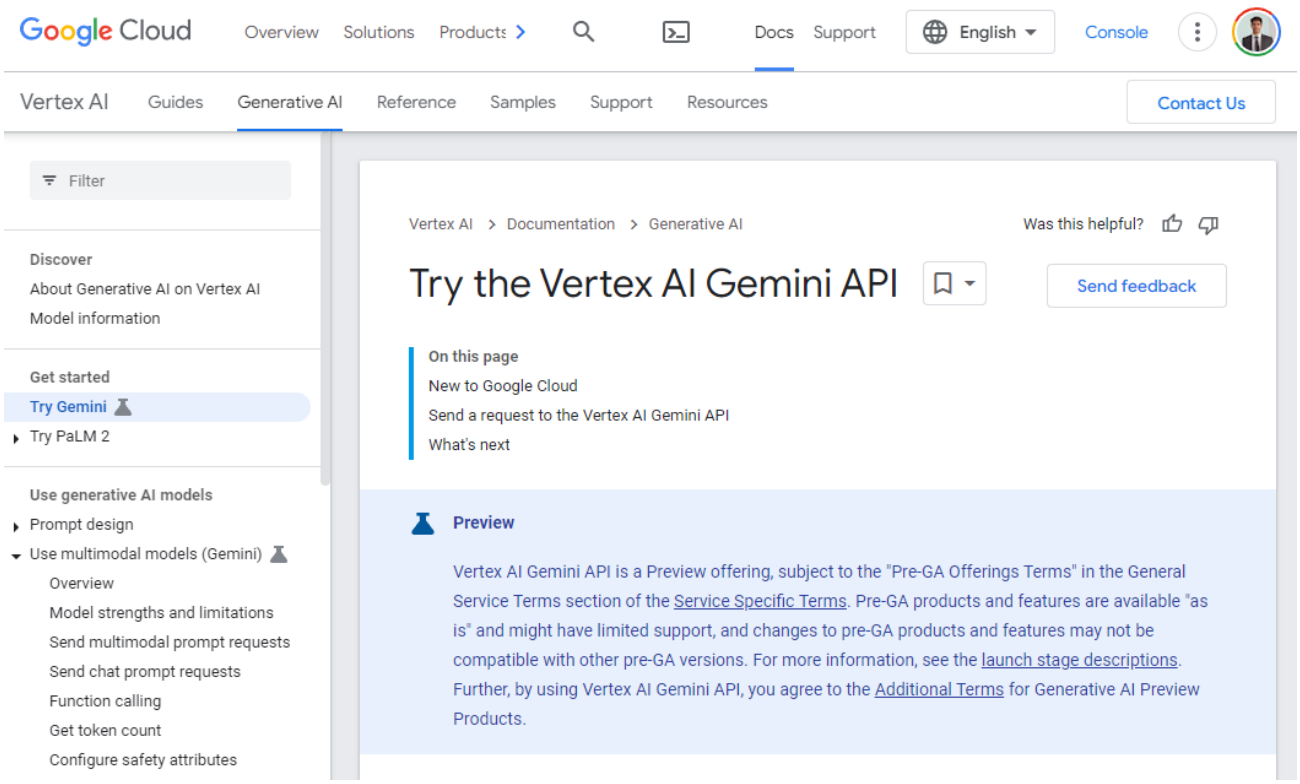


Figura A.1: Captura de pantalla de la documentación de Gemini Pro en estado experimental.

```
# Inferencia de Gemini Pro (Google)
if use_Gemini_Pro:
    generation_config = {
        "max_output_tokens": 4000
    }
    safety_settings ={
        HarmCategory.HARM_CATEGORY_UNSPECIFIED: HarmBlockThreshold.BLOCK_NONE,
        HarmCategory.HARM_CATEGORY_HATE_SPEECH: HarmBlockThreshold.BLOCK_NONE,
        HarmCategory.HARM_CATEGORY_DANGEROUS_CONTENT: HarmBlockThreshold.BLOCK_NONE,
        HarmCategory.HARM_CATEGORY_HARASSMENT: HarmBlockThreshold.BLOCK_NONE,
        HarmCategory.HARM_CATEGORY_SEXUALLY_EXPLICIT: HarmBlockThreshold.BLOCK_NONE,
    }
    model = GenerativeModel("gemini-pro", generation_config=generation_config, safety_settings=safety_settings)
    chat = model.start_chat()
    start_time = time.time()

    response = chat.send_message(prompt,safety_settings=safety_settings)

    end_time = time.time()
    time_Gemini_Pro = end_time - start_time

    text_datos_corregidos_Gemini_Pro = response.text
    print(text_datos_corregidos_Gemini_Pro)
```

Figura A.2: Código utilizado para intentar resolver el problema con la inferencia de Gemini Pro.

### Error de tipo 2: *'IndexError: list index out of range'*

Este error ocurría cuando se introducía un *prompt* con un formato en CSV. Al parecer entraba en conflicto con alguna acción interna de la API. Al igual que el error anterior, este error también le sucedió a más personas (Stack Overflow, 2024). Por ejemplo, se obtenía el error si se enviaba un *prompt* como el siguiente:

*Ponntss; Ponts*
*Gudall; Godall*

Así pues, se tuvo que descartar el uso del modelo Gemini Pro en los experimentos debido a estos dos errores.

# Anexo B

# Código

## B.1. Experimento 1

```
# -------- EXPERIMENTO 1 --------

# LIBRERÍAS
import random
import time
import pandas as pd
from sklearn.metrics import accuracy_score
from sodapy import Socrata
from openai import OpenAI
import vertexai
from vertexai.preview.generative_models import GenerativeModel, HarmCategory,
    HarmBlockThreshold
from huggingface_hub import InferenceClient


# NOTA: En la terminal, hay que hacer log-in en tu cuenta de Google con el
   comando:
# gcloud auth application-default login


# FIJACIÓN DE SEMILLA DE LA LIBRERÍA RANDOM
random.seed(1)


# IMPORTACIÓN DE DATOS
# Configuración de la API de Sócrata
client = Socrata("analisi.transparenciacatalunya.cat", None)

# Datos en formato JSON importados con la API de Sócrata (límite fijado en
```

```
    500,000 registros)
data_municipios = client.get("9aju-tpwc", limit=500000)

# Conversión de JSON a DataFrame
df_data_municipios = pd.DataFrame.from_records(data_municipios)
df_data_municipios


# PREPARACIÓN DE DATOS

# DATOS DE REFERENCIA
# Seleccionamos la columna municipio
df_datos_referencia = df_data_municipios[["nom"]].copy()
df_datos_referencia.rename(columns={"nom": "datos_referencia"}, inplace=True)
df_datos_referencia


# Seleccionamos una muestra de municipios
n_muestra_referencia = 100
df_datos_referencia_muestra = df_datos_referencia.sample(n=
    n_muestra_referencia, replace=False, random_state=1).reset_index(drop=True
    )
df_datos_referencia_muestra


# DATOS TRANSACCIONALES

# Seleccionamos una muestra aleatoria de la muestra de datos de referencia
n_muestra_transaccional = 300
df_datos_transaccionales = df_datos_referencia_muestra.sample(n=
    n_muestra_transaccional, replace=True, random_state=1).reset_index(drop=
    True)
df_datos_transaccionales


# Introducción de errores en los datos transaccionales
# Función para generar datos erróneos
def generar_datos_erroneos(texto):
    # Aleatoriamente, aplica error a un porcentaje de los datos entrantes
    if random.random() < 0.80:

        # Elección aleatoria del tipo de error
        tipo_error = random.choice([
            "sustitución", "eliminación", "adición", "puntuación",
            "repetición", "espaciado", "mezcla_mayúsculas",
```

```
            "símbolos", "fonético", "abreviación", "mezcla_palabras"
        ])

        # Error de sustitución
        if tipo_error == "sustitución":
            i = random.randint(0, len(texto) - 1)
            letra_aleatoria = random.choice("abcdefghijklmnopqrstuvwxyz")
            return texto[:i] + letra_aleatoria + texto[i + 1:]

        # Error de eliminación
        elif tipo_error == "eliminación":
            i = random.randint(0, len(texto) - 1)
            return texto[:i] + texto[i + 1:]

        # Error de adición
        elif tipo_error == "adición":
            i = random.randint(0, len(texto) - 1)
            letra_aleatoria = random.choice("abcdefghijklmnopqrstuvwxyz")
            return texto[:i] + letra_aleatoria + texto[i:]

        # Error de puntuación
        elif tipo_error == "puntuación":
            i = random.randint(0, len(texto) - 1)
            puntuacion_aleatoria = random.choice(",.!¿?")
            return texto[:i] + puntuacion_aleatoria + texto[i:]

        # Error de repetición
        elif tipo_error == "repetición":
            i = random.randint(0, len(texto) - 1)
            return texto[:i] + texto[i] + texto[i:]

        # Error de espaciado
        elif tipo_error == "espaciado":
            i = random.randint(0, len(texto) - 1)
            if " " in texto:
                return texto.replace(" ", "")
            else:
                return texto[:i] + " " + texto[i:]

        # Error de mezcla de mayúsculas
        elif tipo_error == "mezcla_mayúsculas":
            return "".join(random.choice([c.upper(), c.lower()]) for c in
texto)

        # Error de símbolos
```

```
112         elif tipo_error == "símbolos":
113             simbolos = {"a": "@", "e": "3", "i": "!", "o": "0", "s": "$"}
114             return "".join(simbolos.get(c, c) for c in texto.lower())
115 
116         # Error fonético
117         elif tipo_error == "fonético":
118             foneticos = {"c": "k", "k": "c", "s": "z", "z": "s"}
119             return "".join(foneticos.get(c, c) for c in texto.lower())
120 
121         # Error de abreviación
122         elif tipo_error == "abreviación":
123             palabras = texto.split()
124             if len(palabras) > 2:
125                 return palabras[0][0] + ". " + " ".join(palabras[1:])
126             return texto
127 
128         # Error de mezcla de palabras
129         elif tipo_error == "mezcla_palabras":
130             palabras = texto.split()
131             random.shuffle(palabras)
132             return " ".join(palabras)
133 
134     else:
135         return texto
136 
137 
138 # Aplicación de la función para generar datos erróneos
139 df_datos_transaccionales["datos_transaccionales"] = df_datos_transaccionales[
    "datos_referencia"].apply(generar_datos_erroneos)
140 df_datos_transaccionales
141 
142 
143 # Generación de la columna "error", que indica si el dato transaccional tiene
     un error
144 df_datos_transaccionales["error"] = (df_datos_transaccionales["
    datos_transaccionales"] != df_datos_transaccionales["datos_referencia"]).
    astype(int)
145 df_datos_transaccionales
146 
147 
148 # Calidad inicial
149 calidad_inicial = accuracy_score(df_datos_transaccionales["datos_referencia"]
    ,df_datos_transaccionales["datos_transaccionales"])*100
150 
151 print("Calidad inicial de los datos transaccionales: ",f"{calidad_inicial:.2f
```

```
    }","%")
152
153
154 # GENERACIÓN DEL PROMPT
155
156 # String de la Muestra de datos de referencia
157 text_datos_referencia_muestra = ""
158 for i, municipio in enumerate(df_datos_referencia_muestra["datos_referencia"
    ]):
159     if i != (len(df_datos_referencia_muestra["datos_referencia"])-1):
160         text_datos_referencia_muestra += municipio + "; "
161     else:
162         text_datos_referencia_muestra += municipio
163
164 print(text_datos_referencia_muestra)
165
166
167 # String de los Datos transaccionales
168 text_datos_transaccionales = ""
169 for i, municipio in enumerate(df_datos_transaccionales["datos_transaccionales
    "]):
170     if i != (len(df_datos_transaccionales["datos_transaccionales"])-1):
171         text_datos_transaccionales += municipio + "; \n"
172     else:
173         text_datos_transaccionales += municipio
174
175 print(text_datos_transaccionales)
176
177
178 # Generación del prompt
179 # Parte A
180 prompt_parte_A = """Eres un experto en corrección de datos, especializado en
    la detección y estandarización de errores en los datos de municipios de
    Cataluña, España. Además, eres experto en la alineación de datos
    transaccionales con datos de referencia. Tengo dos conjuntos de datos en
    formato CSV con delimitador "; ". El primero incluye datos de referencia
    con el formato correcto de municipios de Cataluña, España. El segundo
    contiene datos transaccionales de municipios que contienen errores. Los
    errores pueden ser de diferentes tipos, como un mal formato. Tu tarea es
    corregir meticulosamente los datos transaccionales, basándote en los datos
     de referencia, de manera que tengan el formato correcto. Aplica tu
    conocimiento para alinear y estandarizar los datos transaccionales con
    precisión. Asegúrate de que los valores que ya están estandarizados queden
     inalterados. El primer valor de los datos transaccionales requiere de
    especial atención para evitar confusiones con el primer valor de los datos
```

```
    transaccionales. No utilices herramientas de análisis ni Python, solo tu
    conocimiento.

Tu respuesta debe contener todos los datos transaccionales, tanto los
    correctos como los corregidos. El formato de la respuesta debe ser un CSV
    con delimitador "; " con dos columnas llamadas: Dato transaccional y Dato
    corregido. Incluye la cabecera en la respuesta. Cabecera:

Dato transaccional; Dato corregido

MUY IMPORTANTE: Sólo incluye el CSV en tu respuesta. No incluyas espacios
    entre cada fila. No incluyas ninguna palabra adicional en tu respuesta. No
    incluyas explicaciones, informaciones, razonamientos o pasos seguidos.
    Asegúrate que el número de valores de la respuesta es el mismo que el nú
    mero de valores de datos transaccionales. No dupliques, ni te dejes
    valores.

Datos de referencia:
"""

# Parte B
prompt_parte_B = """

Archivo CSV a completar:
Dato transaccional; Dato corregido
"""

# Parte C
prompt_parte_C = """

Ejemplo de respuesta:
Dato transaccional; Dato corregido
C. de Llobregat; Corbera de Llobregat
Gudall; Godall
Ponntss; Ponts

Archivo CSV respuesta:"""

# Agrupamiento de strings
prompt = prompt_parte_A + text_datos_referencia_muestra +  prompt_parte_B +
    text_datos_transaccionales + prompt_parte_C
print(prompt)


# CONFIGURACIÓN DE APIS DE LLM
```

```
# (Se han anonimizado los datos de las APIs)
# OpenAI
OPENAI_API_KEY = XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
client = OpenAI(api_key=OPENAI_API_KEY)

# Google Vertex AI
vertexai.init(project=XXXXXXXXXXXXX, location=XXXXXXXXXXXXXXXXX)

# Hugging Face
HUGGING_FACE_API_KEY = XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX #


# INFERENCIA
use_GPT_4_Turbo = True     # Utilizar (True) o no (False) GPT-4 Turbo (OpenAI)
use_GPT_3_5_Turbo = True   # Utilizar (True) o no (False) GPT-3.5 Turbo (OpenAI)
use_Gemini_Pro = False      # Utilizar (True) o no (False) Gemini Pro (Google)
use_LLaMA_2_70B = False    # Utilizar (True) o no (False) LLaMA 2 70B Chat (Meta)


# Inferencia de GPT-4-Turbo (OpenAI)
if use_GPT_4_Turbo:
  start_time = time.time()
  response = client.chat.completions.create(
    model="gpt-4-1106-preview",
    messages=[
      {"role": "user", "content": prompt}
    ],
    max_tokens=4000
  )
  end_time = time.time()
  time_GPT_4_Turbo = end_time - start_time
  text_datos_corregidos_GPT_4_Turbo = response.choices[0].message.content
  print(text_datos_corregidos_GPT_4_Turbo)


# Inferencia de GPT-3.5 Turbo (OpenAI)
if use_GPT_3_5_Turbo:
  start_time = time.time()
  response = client.chat.completions.create(
    model="gpt-3.5-turbo-1106",
    messages=[
```

```
256       {"role": "user", "content": prompt}
257     ],
258     max_tokens=4000
259   )
260   end_time = time.time()
261   time_GPT_3_5_Turbo = end_time - start_time
262   text_datos_corregidos_GPT_3_5_Turbo = response.choices[0].message.content
263   print(text_datos_corregidos_GPT_3_5_Turbo)
264
265
266 # Inferencia de Gemini Pro (Google)
267 if use_Gemini_Pro:
268     generation_config = {
269         "max_output_tokens": 4000
270     }
271     safety_settings ={
272         HarmCategory.HARM_CATEGORY_UNSPECIFIED: HarmBlockThreshold.BLOCK_NONE
    ,
273         HarmCategory.HARM_CATEGORY_HATE_SPEECH: HarmBlockThreshold.BLOCK_NONE
    ,
274         HarmCategory.HARM_CATEGORY_DANGEROUS_CONTENT: HarmBlockThreshold.
    BLOCK_NONE,
275         HarmCategory.HARM_CATEGORY_HARASSMENT: HarmBlockThreshold.BLOCK_NONE,
276         HarmCategory.HARM_CATEGORY_SEXUALLY_EXPLICIT: HarmBlockThreshold.
    BLOCK_NONE,
277     }
278     model = GenerativeModel("gemini-pro", generation_config=generation_config
    , safety_settings=safety_settings)
279     chat = model.start_chat()
280     start_time = time.time()
281     response = chat.send_message(prompt,safety_settings=safety_settings)
282     end_time = time.time()
283     time_Gemini_Pro = end_time - start_time
284     text_datos_corregidos_Gemini_Pro = response.text
285     print(text_datos_corregidos_Gemini_Pro)
286
287
288 # Inferencia de LLaMA 2 70B Chat (Meta)
289 if use_LLaMA_2_70B:
290     client_llama_2 = InferenceClient(model="meta-llama/Llama-2-70b-chat-hf",
    token=HUGGING_FACE_API_KEY)
291     start_time = time.time()
292     response = client_llama_2.text_generation(prompt, max_new_tokens=4000)
293     end_time = time.time()
294     time_LLaMA_2_70B = end_time - start_time
```

```
295     text_datos_corregidos_LLaMA_2_70B = response
296     print(text_datos_corregidos_LLaMA_2_70B)
297
298
299 # GUARDADO DE RESULTADOS EN DATAFRAME
300
301 # Resultados - GPT-4-Turbo (OpenAI)
302 if use_GPT_4_Turbo:
303     lineas = text_datos_corregidos_GPT_4_Turbo.strip().split("\n")[1:]
304     df_datos_transaccionales["GPT_4_Turbo"] = [linea.split("; ")[1] for linea
        in lineas]
305
306 # Resultados - GPT-3.5 Turbo (OpenAI)
307 if use_GPT_3_5_Turbo:
308     lineas = text_datos_corregidos_GPT_3_5_Turbo.strip().split("\n")[1:]
309     df_datos_transaccionales["GPT_3_5_Turbo"] = [linea.split("; ")[1] for
    linea in lineas]
310
311 # Resultados - Gemini Pro (Google)
312 if use_Gemini_Pro:
313     lineas = text_datos_corregidos_Gemini_Pro.strip().split("\n")[1:]
314     df_datos_transaccionales["Gemini_Pro"] = [linea.split("; ")[1] for linea
    in lineas]
315
316 # Resultados - LLaMA 2 70B Chat (Meta)
317 if use_LLaMA_2_70B:
318     lineas = text_datos_corregidos_LLaMA_2_70B.strip().split("\n")[1:-2]
319     df_datos_transaccionales["LLaMA_2_70B"] = [linea.split("; ")[1] for linea
        in lineas]
320
321 df_datos_transaccionales
322
323
324 # EVALUACIÓN
325
326 # Evaluación de GPT-4-Turbo (OpenAI)
327 if use_GPT_4_Turbo:
328     calidad_posterior_GPT_4_Turbo = accuracy_score(df_datos_transaccionales["
    datos_referencia"],df_datos_transaccionales["GPT_4_Turbo"])*100
329     exactitud_datos_erroneos_GPT_4_Turbo = accuracy_score(
    df_datos_transaccionales[df_datos_transaccionales["error"]==1]["
    datos_referencia"],df_datos_transaccionales[df_datos_transaccionales["
    error"]==1]["GPT_4_Turbo"])*100
330     exactitud_datos_correctos_GPT_4_Turbo = accuracy_score(
    df_datos_transaccionales[df_datos_transaccionales["error"]==0]["
```

```
    datos_referencia"],df_datos_transaccionales[df_datos_transaccionales["
    error"]==0]["GPT_4_Turbo"])*100
331
332 # Evaluación de GPT-3.5 Turbo (OpenAI)
333 if use_GPT_3_5_Turbo:
334     calidad_posterior_GPT_3_5_Turbo = accuracy_score(df_datos_transaccionales
    ["datos_referencia"],df_datos_transaccionales["GPT_3_5_Turbo"])*100
335     exactitud_datos_erroneos_GPT_3_5_Turbo = accuracy_score(
    df_datos_transaccionales[df_datos_transaccionales["error"]==1]["
    datos_referencia"],df_datos_transaccionales[df_datos_transaccionales["
    error"]==1]["GPT_3_5_Turbo"])*100
336     exactitud_datos_correctos_GPT_3_5_Turbo = accuracy_score(
    df_datos_transaccionales[df_datos_transaccionales["error"]==0]["
    datos_referencia"],df_datos_transaccionales[df_datos_transaccionales["
    error"]==0]["GPT_3_5_Turbo"])*100
337
338 # Evaluación de Gemini Pro (Google)
339 if use_Gemini_Pro:
340     calidad_posterior_Gemini_Pro = accuracy_score(df_datos_transaccionales["
    datos_referencia"],df_datos_transaccionales["Gemini_Pro"])*100
341     exactitud_datos_erroneos_Gemini_Pro = accuracy_score(
    df_datos_transaccionales[df_datos_transaccionales["error"]==1]["
    datos_referencia"],df_datos_transaccionales[df_datos_transaccionales["
    error"]==1]["Gemini_Pro"])*100
342     exactitud_datos_correctos_Gemini_Pro = accuracy_score(
    df_datos_transaccionales[df_datos_transaccionales["error"]==0]["
    datos_referencia"],df_datos_transaccionales[df_datos_transaccionales["
    error"]==0]["Gemini_Pro"])*100
343
344 # Evaluación de LLaMA 2 70B Chat (Meta)
345 if use_LLaMA_2_70B:
346     calidad_posterior_LLaMA_2_70B = accuracy_score(df_datos_transaccionales["
    datos_referencia"],df_datos_transaccionales["LLaMA_2_70B"])*100
347     exactitud_datos_erroneos_LLaMA_2_70B = accuracy_score(
    df_datos_transaccionales[df_datos_transaccionales["error"]==1]["
    datos_referencia"],df_datos_transaccionales[df_datos_transaccionales["
    error"]==1]["LLaMA_2_70B"])*100
348     exactitud_datos_correctos_LLaMA_2_70B = accuracy_score(
    df_datos_transaccionales[df_datos_transaccionales["error"]==0]["
    datos_referencia"],df_datos_transaccionales[df_datos_transaccionales["
    error"]==0]["LLaMA_2_70B"])*100
349
350
351 # IMPRESIÓN DE RESULTADOS
352
```

```
353 print("Calidad inicial de los datos transaccionales: ",calidad_inicial,"%\n")
354
355 print("Resultados de los modelos: ")
356
357 # GPT-4-Turbo (OpenAI)
358 if use_GPT_4_Turbo:
359     print("GPT-4 Turbo:\t\tCalidad posterior:",calidad_posterior_GPT_4_Turbo,
360           "%    Incremento calidad:",calidad_posterior_GPT_4_Turbo-
    calidad_inicial,
361           "%    Exactitud (accuracy) en datos erroneos:",
    exactitud_datos_erroneos_GPT_4_Turbo,
362           "%    Exactitud (accuracy) en datos correctos:",
    exactitud_datos_correctos_GPT_4_Turbo,
363           "%    Tiempo:",time_GPT_4_Turbo,"s")
364 # GPT-3.5 Turbo (OpenAI)
365 if use_GPT_3_5_Turbo:
366     print("GPT-3.5 Turbo:\t\tCalidad posterior:",
    calidad_posterior_GPT_3_5_Turbo,
367           "%    Incremento calidad:",calidad_posterior_GPT_3_5_Turbo-
    calidad_inicial,
368           "%    Exactitud (accuracy) en datos erroneos:",
    exactitud_datos_erroneos_GPT_3_5_Turbo,
369           "%    Exactitud (accuracy) en datos correctos:",
    exactitud_datos_correctos_GPT_3_5_Turbo,
370           "%    Tiempo:",time_GPT_3_5_Turbo,"s")
371 # Gemini Pro (Google)
372 if use_Gemini_Pro:
373     print("Gemini Pro:\tCalidad posterior:",calidad_posterior_Gemini_Pro,
374           "%    Incremento calidad:",calidad_posterior_Gemini_Pro-
    calidad_inicial,
375           "%    Exactitud (accuracy) en datos erroneos:",
    exactitud_datos_erroneos_Gemini_Pro,
376           "%    Exactitud (accuracy) en datos correctos:",
    exactitud_datos_correctos_Gemini_Pro,
377           "%    Tiempo:",time_Gemini_Pro,"s")
378
379 # LLaMA 2 70B Chat (Meta)
380 if use_LLaMA_2_70B:
381     print("LLaMA_2_70B_Chat:\tCalidad posterior:",
    calidad_posterior_LLaMA_2_70B,
382           "%    Incremento calidad:",calidad_posterior_LLaMA_2_70B-
    calidad_inicial,
383           "%    Exactitud (accuracy) en datos erroneos:",
    exactitud_datos_erroneos_LLaMA_2_70B,
384           "%    Exactitud (accuracy) en datos correctos:",
```

```
    exactitud_datos_correctos_LLaMA_2_70B,
385         "%    Tiempo:",time_LLaMA_2_70B,"s")
```

## B.2. Experimento 2

```
# -------- EXPERIMENTO 2 --------

# LIBRERÍAS
import random
import time
import pandas as pd
from sklearn.metrics import accuracy_score
from sodapy import Socrata
from openai import OpenAI
import vertexai
from vertexai.preview.generative_models import GenerativeModel, HarmCategory,
    HarmBlockThreshold
from huggingface_hub import InferenceClient
from io import StringIO

# NOTA: En la terminal, hay que hacer log-in en tu cuenta de Google con el
   comando:
# gcloud auth application-default login

# FIJACIÓN DE SEMILLA DE LA LIBRERÍA RANDOM
random.seed(1)

# IMPORTACIÓN DE DATOS
# Configuración de la API de Sócrata
client = Socrata("analisi.transparenciacatalunya.cat", None)

# Datos en formato JSON importados con la API de Sócrata (límite fijado en
   500,000 registros)
data_calles = client.get("2e2b-4ksy", limit=500000)

# Conversión de JSON a DataFrame
df_data_calles = pd.DataFrame.from_records(data_calles)
df_data_calles

# PREPARACIÓN DE DATOS
# DATOS DE REFERENCIA
df_datos_referencia = df_data_calles[df_data_calles["nom_municipi"]=="
   Barcelona"].dropna().copy().reset_index(drop=True) # Eliminamos las filas
   que no tienen nombre de calle
```

```
df_datos_referencia

# Seleccionamos una muestra de municipios
n_muestra_referencia = 5
df_datos_referencia_muestra = df_datos_referencia.sample(n=
    n_muestra_referencia, replace=False, random_state=1).reset_index(drop=True
    )
df_datos_referencia_muestra

# DATOS TRANSACCIONALES

# Seleccionamos una muestra aleatoria de la muestra de datos de referencia
n_muestra_transaccional = 10
df_datos_transaccionales_correctos = df_datos_referencia_muestra.sample(n=
    n_muestra_transaccional, replace=True, random_state=1).reset_index(drop=
    True)
df_datos_transaccionales_correctos

# Datos transaccionales
df_datos_transaccionales = df_datos_transaccionales_correctos[["
    nom_compost_via"]].copy()
df_datos_transaccionales.rename(columns={"nom_compost_via": "
    datos_transaccionales"}, inplace=True)
df_datos_transaccionales

# Introducción de errores en los datos transaccionales
# Función para generar datos erróneos
def generar_datos_erroneos(texto):
    # Aleatoriamente, aplica error a un porcentaje de los datos entrantes
    if random.random() < 0.80:

        # Elección aleatoria del tipo de error
        tipo_error = random.choice([
            "sustitución", "eliminación", "adición", "puntuación",
            "repetición", "espaciado", "mezcla_mayúsculas",
            "símbolos", "fonético", "abreviación", "mezcla_palabras"
        ])

        # Error de sustitución
        if tipo_error == "sustitución":
            i = random.randint(0, len(texto) - 1)
            letra_aleatoria = random.choice("abcdefghijklmnopqrstuvwxyz")
            return texto[:i] + letra_aleatoria + texto[i + 1:]

        # Error de eliminación
```

```
74          elif tipo_error == "eliminación":
75              i = random.randint(0, len(texto) - 1)
76              return texto[:i] + texto[i + 1:]
77
78          # Error de adición
79          elif tipo_error == "adición":
80              i = random.randint(0, len(texto) - 1)
81              letra_aleatoria = random.choice("abcdefghijklmnopqrstuvwxyz")
82              return texto[:i] + letra_aleatoria + texto[i:]
83
84          # Error de puntuación
85          elif tipo_error == "puntuación":
86              i = random.randint(0, len(texto) - 1)
87              puntuacion_aleatoria = random.choice(",.!¿?")
88              return texto[:i] + puntuacion_aleatoria + texto[i:]
89
90          # Error de repetición
91          elif tipo_error == "repetición":
92              i = random.randint(0, len(texto) - 1)
93              return texto[:i] + texto[i] + texto[i:]
94
95          # Error de espaciado
96          elif tipo_error == "espaciado":
97              i = random.randint(0, len(texto) - 1)
98              if " " in texto:
99                  return texto.replace(" ", "")
100             else:
101                 return texto[:i] + " " + texto[i:]
102
103         # Error de mezcla de mayúsculas
104         elif tipo_error == "mezcla_mayúsculas":
105             return "".join(random.choice([c.upper(), c.lower()]) for c in
    texto)
106
107         # Error de símbolos
108         elif tipo_error == "símbolos":
109             simbolos = {"a": "@", "e": "3", "i": "!", "o": "0", "s": "$"}
110             return "".join(simbolos.get(c, c) for c in texto.lower())
111
112         # Error fonético
113         elif tipo_error == "fonético":
114             foneticos = {"c": "k", "k": "c", "s": "z", "z": "s"}
115             return "".join(foneticos.get(c, c) for c in texto.lower())
116
117         # Error de abreviación
```

```
        elif tipo_error == "abreviación":
            palabras = texto.split()
            if len(palabras) > 2:
                return palabras[0][0] + ". " + " ".join(palabras[1:])
            return texto

        # Error de mezcla de palabras
        elif tipo_error == "mezcla_palabras":
            palabras = texto.split()
            random.shuffle(palabras)
            return " ".join(palabras)

    else:
        return texto

# Aplicación de la función para generar datos erróneos
df_datos_transaccionales["datos_transaccionales"] = df_datos_transaccionales["datos_transaccionales"].apply(generar_datos_erroneos)
df_datos_transaccionales

# Generación de la columna "error", que indica si el dato transaccional tiene un error
df_datos_transaccionales["error"] = (df_datos_transaccionales["datos_transaccionales"] != df_datos_transaccionales_correctos["nom_compost_via"]).astype(int)
df_datos_transaccionales

# Calidad inicial
calidad_inicial = accuracy_score(df_datos_transaccionales_correctos["nom_compost_via"],df_datos_transaccionales["datos_transaccionales"])*100

print("Calidad inicial de los datos transaccionales: ",f"{calidad_inicial:.2f}","%")

# GENERACIÓN DEL PROMPT

# String de la Muestra de datos de referencia
text_datos_referencia_muestra = df_datos_referencia_muestra.to_csv(sep=';', index=False)
print(text_datos_referencia_muestra)

# String de los Datos transaccionales
text_datos_transaccionales = ""
for i, municipio in enumerate(df_datos_transaccionales["datos_transaccionales"]):
```

```
    if i != (len(df_datos_transaccionales["datos_transaccionales"])-1):
        text_datos_transaccionales += municipio + "; \n"
    else:
        text_datos_transaccionales += municipio

print(text_datos_transaccionales)

# Generación del prompt
# Parte A
prompt_parte_A = """Eres un experto en corrección de datos, especializado en la detección y estandarización de errores en los datos de calles de Cataluña, España. Además, eres experto en la alineación de datos transaccionales con datos de referencia. Tengo dos conjuntos de datos en formato CSV con delimitador ";". El primero incluye datos de referencia con el formato y atributos correctos de las calles de Cataluña, España. El segundo contiene datos transaccionales de calles que contienen errores. Los errores pueden ser de diferentes tipos, como un mal formato. Tu tarea es corregir meticulosamente los datos transaccionales, basándote en los datos de referencia, de manera que tengan el formato correcto. Aplica tu conocimiento para alinear y estandarizar los datos transaccionales con precisión. Asegúrate de que los valores que ya están estandarizados queden inalterados. El primer valor de los datos transaccionales requiere de especial atención para evitar confusiones con el primer valor de los datos transaccionales. No utilices herramientas de análisis ni Python, solo tu conocimiento.

Tu respuesta debe contener todos los datos transaccionales, tanto los correctos como los corregidos. El formato de la respuesta debe ser un CSV con cabecera, con delimitador ";" y con las columnas que se muestran a continuación:

Dato transaccional;nom_compost_via;codi_unitat_poblacional;codi_tipus_via;nom_via;nexe_via;data;nom_municipi;nom_unitat_poblacional;tipus_via

MUY IMPORTANTE: Sólo incluye el CSV en tu respuesta. No incluyas espacios entre cada fila. No incluyas ninguna palabra adicional en tu respuesta. No incluyas explicaciones, informaciones, razonamientos o pasos seguidos.
MUY IMPORTANTE: Asegúrate que el número de filas de la respuesta es el correcto. No dupliques, ni te dejes valores.

Datos de referencia:
"""

# Parte B
prompt_parte_B = """
```

```
178
179 Archivo CSV a completar:
180 Dato transaccional;nom_compost_via;codi_unitat_poblacional;codi_tipus_via;
    nom_via;nexe_via;data;nom_municipi;nom_unitat_poblacional;tipus_via
181 """
182
183 # Parte C
184 prompt_parte_C = """
185
186 Ejemplo de respuesta:
187 Dato transaccional;nom_compost_via;codi_unitat_poblacional;codi_tipus_via;
    nom_via;nexe_via;data;nom_municipi;nom_unitat_poblacional;tipus_via
188 C. Jorba;Carrer de Jorba;0801930001017;016;Jorba;de;20130601;Barcelona;
    Barcelona;Carrer
189 Avinguda Almacelles;Carrer d'Almacelles;0801930001017;016;Almacelles;d'
    ;20130601;Barcelona;Barcelona;Carrer
190 C/ Periodistes;Carrer dels Periodistes;0801930001017;016;Periodistes;dels
    ;20130601;Barcelona;Barcelona;Carrer
191
192 Archivo CSV respuesta:"""
193
194 # Agrupamiento de strings
195 prompt = prompt_parte_A + text_datos_referencia_muestra +  prompt_parte_B +
    text_datos_transaccionales + prompt_parte_C
196 print(prompt)
197
198 # CONFIGURACIÓN DE APIS DE LLM
199
200 # OpenAI
201 OPENAI_API_KEY = XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
202 client = OpenAI(api_key=OPENAI_API_KEY)
203
204 # Google Vertex AI
205 vertexai.init(project=XXXXXXXXXX, location=XXXXXXXXXXXXXXX)
206
207 # Hugging Face
208 HUGGING_FACE_API_KEY = XXXXXXXXXXXXXXXXXXXXXXXXXXXX
209
210 # INFERENCIA
211 use_GPT_4_Turbo = True     # Utilizar (True) o no (False) GPT-4 Turbo (OpenAI
    )
212 use_GPT_3_5_Turbo = True   # Utilizar (True) o no (False) GPT-3.5 Turbo (
    OpenAI)
213 use_Gemini_Pro = True      # Utilizar (True) o no (False) Gemini Pro (Google)
214 use_LLaMA_2_70B = True    # Utilizar (True) o no (False) LLaMA 2 70B Chat (
```

```
    Meta)
215
216 # Inferencia de GPT-4-Turbo (OpenAI)
217 if use_GPT_4_Turbo:
218   start_time = time.time()
219   response = client.chat.completions.create(
220     model="gpt-4-1106-preview",
221     messages=[
222       {"role": "user", "content": prompt}
223     ],
224     max_tokens=4000
225   )
226   end_time = time.time()
227   time_GPT_4_Turbo = end_time - start_time
228   text_datos_corregidos_GPT_4_Turbo = response.choices[0].message.content
229   print(text_datos_corregidos_GPT_4_Turbo)
230
231 # Inferencia de GPT-3.5 Turbo (OpenAI)
232 if use_GPT_3_5_Turbo:
233   start_time = time.time()
234   response = client.chat.completions.create(
235     model="gpt-3.5-turbo-0125",
236     messages=[
237       {"role": "user", "content": prompt}
238     ],
239     max_tokens=4000
240   )
241   end_time = time.time()
242   time_GPT_3_5_Turbo = end_time - start_time
243   text_datos_corregidos_GPT_3_5_Turbo = response.choices[0].message.content
244   print(text_datos_corregidos_GPT_3_5_Turbo)
245
246 # Inferencia de Gemini Pro (Google)
247 if use_Gemini_Pro:
248     generation_config = {
249         "max_output_tokens": 4000
250     }
251     safety_settings ={
252         HarmCategory.HARM_CATEGORY_UNSPECIFIED: HarmBlockThreshold.BLOCK_NONE
    ,
253         HarmCategory.HARM_CATEGORY_HATE_SPEECH: HarmBlockThreshold.BLOCK_NONE
    ,
254         HarmCategory.HARM_CATEGORY_DANGEROUS_CONTENT: HarmBlockThreshold.
    BLOCK_NONE,
255         HarmCategory.HARM_CATEGORY_HARASSMENT: HarmBlockThreshold.BLOCK_NONE,
```

```
256         HarmCategory.HARM_CATEGORY_SEXUALLY_EXPLICIT: HarmBlockThreshold.
    BLOCK_NONE,
257     }
258     model = GenerativeModel("gemini-pro", generation_config=generation_config
    , safety_settings=safety_settings)
259     chat = model.start_chat()
260     start_time = time.time()
261     response = chat.send_message(prompt,safety_settings=safety_settings)
262     end_time = time.time()
263     time_Gemini_Pro = end_time - start_time
264     text_datos_corregidos_Gemini_Pro = response.text
265     print(text_datos_corregidos_Gemini_Pro)
266
267 # Inferencia de LLaMA 2 70B Chat (Meta)
268 if use_LLaMA_2_70B:
269     client_llama_2 = InferenceClient(model="meta-llama/Llama-2-70b-chat-hf",
    token=HUGGING_FACE_API_KEY)
270     start_time = time.time()
271     response = client_llama_2.text_generation(prompt, max_new_tokens=4096)
272     end_time = time.time()
273     time_LLaMA_2_70B = end_time - start_time
274     text_datos_corregidos_LLaMA_2_70B = response
275     print(text_datos_corregidos_LLaMA_2_70B)
276
277 # GUARDADO DE RESULTADOS EN DATAFRAME
278
279 # Resultados - GPT-4-Turbo (OpenAI)
280 if use_GPT_4_Turbo:
281     df_datos_corregidos_GPT_4_Turbo = pd.read_csv(StringIO(
    text_datos_corregidos_GPT_4_Turbo), delimiter=';',dtype=str)
282     df_datos_corregidos_GPT_4_Turbo["concat"] =
    df_datos_corregidos_GPT_4_Turbo["nom_compost_via"] + ' ' +
    df_datos_corregidos_GPT_4_Turbo["codi_unitat_poblacional"] + ' ' +
    df_datos_corregidos_GPT_4_Turbo["codi_tipus_via"] + ' ' +
    df_datos_corregidos_GPT_4_Turbo["nom_via"] + ' ' +
    df_datos_corregidos_GPT_4_Turbo["nexe_via"] + ' ' +
    df_datos_corregidos_GPT_4_Turbo["data"] + ' ' +
    df_datos_corregidos_GPT_4_Turbo["nom_municipi"] + ' ' +
    df_datos_corregidos_GPT_4_Turbo["nom_unitat_poblacional"] + ' ' +
    df_datos_corregidos_GPT_4_Turbo["tipus_via"]
283
284 # Resultados - GPT-3.5 Turbo (OpenAI)
285 if use_GPT_3_5_Turbo:
286     df_datos_corregidos_GPT_3_5_Turbo = pd.read_csv(StringIO(
    text_datos_corregidos_GPT_3_5_Turbo), delimiter=';',dtype=str)
```

```
287     df_datos_corregidos_GPT_3_5_Turbo["concat"] =
    df_datos_corregidos_GPT_3_5_Turbo["nom_compost_via"] + ' ' +
    df_datos_corregidos_GPT_3_5_Turbo["codi_unitat_poblacional"] + ' ' +
    df_datos_corregidos_GPT_3_5_Turbo["codi_tipus_via"] + ' ' +
    df_datos_corregidos_GPT_3_5_Turbo["nom_via"] + ' ' +
    df_datos_corregidos_GPT_3_5_Turbo["nexe_via"] + ' ' +
    df_datos_corregidos_GPT_3_5_Turbo["data"] + ' ' +
    df_datos_corregidos_GPT_3_5_Turbo["nom_municipi"] + ' ' +
    df_datos_corregidos_GPT_3_5_Turbo["nom_unitat_poblacional"] + ' ' +
    df_datos_corregidos_GPT_3_5_Turbo["tipus_via"]
288
289 # Resultados - Gemini Pro (Google)
290 if use_Gemini_Pro:
291     df_datos_corregidos_Gemini_Pro = pd.read_csv(StringIO(
    text_datos_corregidos_Gemini_Pro), delimiter=';',dtype=str)
292     df_datos_corregidos_Gemini_Pro["concat"] = df_datos_corregidos_Gemini_Pro
    ["nom_compost_via"] + ' ' + df_datos_corregidos_Gemini_Pro["
    codi_unitat_poblacional"] + ' ' + df_datos_corregidos_Gemini_Pro["
    codi_tipus_via"] + ' ' + df_datos_corregidos_Gemini_Pro["nom_via"] + ' ' +
     df_datos_corregidos_Gemini_Pro["nexe_via"] + ' ' +
    df_datos_corregidos_Gemini_Pro["data"] + ' ' +
    df_datos_corregidos_Gemini_Pro["nom_municipi"] + ' ' +
    df_datos_corregidos_Gemini_Pro["nom_unitat_poblacional"] + ' ' +
    df_datos_corregidos_Gemini_Pro["tipus_via"]
293
294 # Resultados - LLaMA 2 70B Chat (Meta)
295 if use_LLaMA_2_70B:
296     df_datos_corregidos_LLaMA_2_70B = pd.read_csv(StringIO(
    text_datos_corregidos_LLaMA_2_70B), delimiter=';',dtype=str)
297     df_datos_corregidos_LLaMA_2_70B["concat"] =
    df_datos_corregidos_LLaMA_2_70B["nom_compost_via"] + ' ' +
    df_datos_corregidos_LLaMA_2_70B["codi_unitat_poblacional"] + ' ' +
    df_datos_corregidos_LLaMA_2_70B["codi_tipus_via"] + ' ' +
    df_datos_corregidos_LLaMA_2_70B["nom_via"] + ' ' +
    df_datos_corregidos_LLaMA_2_70B["nexe_via"] + ' ' +
    df_datos_corregidos_LLaMA_2_70B["data"] + ' ' +
    df_datos_corregidos_LLaMA_2_70B["nom_municipi"] + ' ' +
    df_datos_corregidos_LLaMA_2_70B["nom_unitat_poblacional"] + ' ' +
    df_datos_corregidos_LLaMA_2_70B["tipus_via"]
298
299
300 # Concatenación de las filas
301 df_datos_transaccionales_correctos = df_datos_transaccionales_correctos.
    astype(str)
302 df_datos_transaccionales_correctos["concat"] =
```

```
    df_datos_transaccionales_correctos["nom_compost_via"] + ' ' +
    df_datos_transaccionales_correctos["codi_unitat_poblacional"] + ' ' +
    df_datos_transaccionales_correctos["codi_tipus_via"] + ' ' +
    df_datos_transaccionales_correctos["nom_via"] + ' ' +
    df_datos_transaccionales_correctos["nexe_via"] + ' ' +
    df_datos_transaccionales_correctos["data"] + ' ' +
    df_datos_transaccionales_correctos["nom_municipi"] + ' ' +
    df_datos_transaccionales_correctos["nom_unitat_poblacional"] + ' ' +
    df_datos_transaccionales_correctos["tipus_via"]
303
304 # EVALUACIÓN
305
306 # Evaluación de GPT-4-Turbo (OpenAI)
307 if use_GPT_4_Turbo:
308     calidad_posterior_GPT_4_Turbo = accuracy_score(
    df_datos_transaccionales_correctos["concat"],
    df_datos_corregidos_GPT_4_Turbo["concat"])*100
309
310 # Evaluación de GPT-3.5 Turbo (OpenAI)
311 if use_GPT_3_5_Turbo:
312     calidad_posterior_GPT_3_5_Turbo = accuracy_score(
    df_datos_transaccionales_correctos["concat"],
    df_datos_corregidos_GPT_3_5_Turbo["concat"])*100
313
314 # Evaluación de Gemini Pro (Google)
315 if use_Gemini_Pro:
316     calidad_posterior_Gemini_Pro = accuracy_score(
    df_datos_transaccionales_correctos["concat"],
    df_datos_corregidos_Gemini_Pro["concat"])*100
317
318 # Evaluación de LLaMA 2 70B Chat (Meta)
319 if use_LLaMA_2_70B:
320     calidad_posterior_LLaMA_2_70B = accuracy_score(
    df_datos_transaccionales_correctos["concat"],
    df_datos_corregidos_LLaMA_2_70B["concat"])*100
321
322 # IMPRESIÓN DE RESULTADOS
323
324 print("Calidad inicial de los datos transaccionales: ",calidad_inicial,"%\n")
325
326 print("Resultados de los modelos: ")
327
328 # GPT-4-Turbo (OpenAI)
329 if use_GPT_4_Turbo:
330     print("GPT-4 Turbo:\t\tCalidad posterior:",calidad_posterior_GPT_4_Turbo,
```

```
331           "%    Incremento calidad:",calidad_posterior_GPT_4_Turbo-
    calidad_inicial,
332           "%    Tiempo:",time_GPT_4_Turbo,"s")
333 # GPT-3.5 Turbo (OpenAI)
334 if use_GPT_3_5_Turbo:
335     print("GPT-3.5 Turbo:\t\tCalidad posterior:",
    calidad_posterior_GPT_3_5_Turbo,
336           "%    Incremento calidad:",calidad_posterior_GPT_3_5_Turbo-
    calidad_inicial,
337           "%    Tiempo:",time_GPT_3_5_Turbo,"s")
338 # Gemini Pro (Google)
339 if use_Gemini_Pro:
340     print("Gemini Pro:\tCalidad posterior:",calidad_posterior_Gemini_Pro,
341           "%    Incremento calidad:",calidad_posterior_Gemini_Pro-
    calidad_inicial,
342           "%    Tiempo:",time_Gemini_Pro,"s")
343
344 # LLaMA 2 70B Chat (Meta)
345 if use_LLaMA_2_70B:
346     print("LLaMA_2_70B_Chat:\tCalidad posterior:",
    calidad_posterior_LLaMA_2_70B,
347           "%    Incremento calidad:",calidad_posterior_LLaMA_2_70B-
    calidad_inicial,
348           "%    Tiempo:",time_LLaMA_2_70B,"s")
```

## B.3. Experimento 3

```
1  # -------- EXPERIMENTO 3 --------
2
3  # LIBRERÍAS
4  import random
5  import time
6  import pandas as pd
7  from sklearn.metrics import accuracy_score
8  from sodapy import Socrata
9  from openai import OpenAI
10 import vertexai
11 from vertexai.preview.generative_models import GenerativeModel, HarmCategory,
       HarmBlockThreshold
12 from huggingface_hub import InferenceClient
13 from io import StringIO
14
15 # NOTA: En la terminal, hay que hacer log-in en tu cuenta de Google con el
    comando:
```

```
# gcloud auth application-default login

# FIJACIÓN DE SEMILLA DE LA LIBRERÍA RANDOM
#random.seed(1)

# GENERACIÓN DE DATOS MAESTROS

# Nombres
def generar_nombre():
    nombres =  [
        # Hombres
        "Santiago", "Mateo", "Matías", "Nicolás", "Alejandro", "Diego", "Samuel", "Sebastián", "Daniel", "Lucas",
        "Benjamín", "Pablo", "Alex", "David", "Emiliano", "Andrés", "Joaquín", "Carlos", "José", "Adrián",
        "Gabriel", "Miguel", "Ángel", "Javier", "Fernando", "Christian", "Felipe", "Rodrigo", "Francisco", "Leo",
        "Martín", "Marco", "Iván", "Gustavo", "Rafael", "Julio", "Victor", "Hugo", "Salvador", "Antonio",
        "Luis", "Pedro", "Jorge", "Alberto", "Sergio", "Eduardo", "Mario", "Ernesto", "Ricardo", "Manuel",
        # Mujeres
        "Sofía", "Valentina", "Isabella", "Camila", "Valeria", "Mariana", "Gabriela", "Daniela", "Lucía", "Victoria",
        "Martina", "Jimena", "Elena", "Laura", "Sara", "Ana", "Carla", "Paula", "Julia", "Clara",
        "Carmen", "Lorena", "Alba", "Alicia", "Teresa", "Rosa", "Patricia", "Andrea", "Silvia", "Mónica",
        "Natalia", "Angela", "Raquel", "Sonia", "Miriam", "Beatriz", "Bárbara", "Dolores", "Luisa", "Cristina",
        "Elena", "Irene", "Virginia", "Susana", "Adriana", "Juliana", "Esther", "Carolina", "Inés", "Lidia"
    ]
    return random.choice(nombres)

# Apellidos
def generar_apellidos():
    apellidos = ["García","Rodríguez","González","Fernández","López","Martínez","Sánchez","Pérez","Gómez","Martín",
                 "Jimenez","Hernández","Ruiz","Díaz","Moreno","Muñoz","Álvarez","Romero","Gutiérrez","Alonso","Navarro",
                 "Torres","Domínguez","Ramos","Vázquez","Ramírez","Gil","Serrano","Morales","Molina","Blanco","Suárez",
                 "Castro","Ortega","Delgado","Ortiz","Marin","Rubio","Nuñez","Medina","Sanz","Castillo","Iglesias","Cortés",
```

```
47                 "Garrido","Santos","Guerrero","Lozano","Cano","Cruz","Méndez"
   ,"Flores","Prieto","Herrera","Peña","León",
48                 "Márquez","Cabrera","Gallego","Calvo","Vidal","Campos","Reyes
   ","Vega","Fuentes","Carrasco","Diez","Aguilar",
49                 "Caballero","Nieto","Santana","Vargas","Pascual","Giménez","
   Herrero","Hidalgo","Montero","Lorenzo","Santiago",
50                 "Benitez","Duran","Ibáñez","Arias","Mora","Ferrer","Carmona",
   "Vicente","Rojas","Soto","Crespo","Roman","Pastor",
51                 "Velasco","Parra","Sáez","Moya","Bravo","Rivera","Gallardo","
   Soler"]
52     return random.choice(apellidos)
53 
54 # Email
55 def quitar_acentos(str):
56     acentos = {
57         "á": "a", "é": "e", "í": "i", "ó": "o", "ú": "u",
58         "Á": "A", "É": "E", "Í": "I", "Ó": "O", "Ú": "U"
59     }
60     return "".join(acentos.get(char, char) for char in str)
61 
62 def generar_email(nombre,apellido_1,apellido_2):
63     dominios = ["gmail.com","outlook.com","yahoo.com"]
64     return quitar_acentos(nombre.lower()) + "." + quitar_acentos(apellido_1.
   lower()) + "." + quitar_acentos(apellido_2.lower()) + "@" + random.choice(
   dominios)
65 
66 # DNIs
67 def generar_dni():
68     numero = random.randint(10000000, 99999999)
69     letras = "TRWAGMYFPDXBNJZSQVHLCKE"
70     letra = letras[numero % 23]
71     return f"{numero}{letra}"
72 
73 # Número de teléfono
74 def generar_telefono():
75     primer_digito = random.choice(["6", "7"])
76     digitos_restantes = "".join(random.choices("0123456789", k=8))
77     numero_telefono = primer_digito + digitos_restantes
78     return numero_telefono
79 
80 
81 # Número de filas que quieres en tu DataFrame
82 num_filas = 4
83 
84 lista_dni = []
```

```
lista_nombre = []
lista_apellido_1 = []
lista_apellido_2 = []
lista_email = []
lista_telefono = []

for _ in range(num_filas):
    dni = generar_dni()
    nombre = generar_nombre()
    apellido_1 = generar_apellidos()
    apellido_2 = generar_apellidos()
    email = generar_email(nombre,apellido_1,apellido_2)
    telefono = generar_telefono()

    lista_dni.append(dni)
    lista_nombre.append(nombre)
    lista_apellido_1.append(apellido_1)
    lista_apellido_2.append(apellido_2)
    lista_email.append(email)
    lista_telefono.append(telefono)

df_datos_maestros = pd.DataFrame({
    "DNI": lista_dni,
    "Nombre": lista_nombre,
    "Apellido_1": lista_apellido_1,
    "Apellido_2": lista_apellido_2,
    "Email": lista_email,
    "Teléfono": lista_telefono,
})

df_datos_maestros["concat"] = df_datos_maestros["DNI"] + " " +
    df_datos_maestros["Nombre"] + " " + df_datos_maestros["Apellido_1"] + " "
    + df_datos_maestros["Apellido_2"] + " " + df_datos_maestros["Email"] + " "
     + df_datos_maestros["Teléfono"]

df_datos_maestros

# DATOS TRANSACCIONALES

# Seleccionamos una muestra aleatoria de la muestra de datos maestros
n_muestra_transaccional = 8
df_datos_transaccionales_correctos = df_datos_maestros.sample(n=
    n_muestra_transaccional, replace=True, random_state=1).reset_index(drop=
    True)
df_datos_transaccionales_correctos["concat"] =
```

```
    df_datos_transaccionales_correctos["DNI"] + " " +
    df_datos_transaccionales_correctos["Nombre"] + " " +
    df_datos_transaccionales_correctos["Apellido_1"] + " " +
    df_datos_transaccionales_correctos["Apellido_2"] + " " +
    df_datos_transaccionales_correctos["Email"] + " " +
    df_datos_transaccionales_correctos["Teléfono"]
df_datos_transaccionales_correctos

# Introducción de errores en los datos transaccionales
# Función para generar datos erróneos
def generar_datos_erroneos(texto):

    # Elección aleatoria del tipo de error
    tipo_error = random.choice([
        "sustitución", "eliminación", "adición", "puntuación",
        "repetición", "espaciado", "mezcla_mayúsculas",
        "símbolos", "fonético", "abreviación", "mezcla_palabras"
    ])

    # Error de sustitución
    if tipo_error == "sustitución":
        i = random.randint(0, len(texto) - 1)
        letra_aleatoria = random.choice("abcdefghijklmnopqrstuvwxyz")
        return texto[:i] + letra_aleatoria + texto[i + 1:]

    # Error de eliminación
    elif tipo_error == "eliminación":
        i = random.randint(0, len(texto) - 1)
        return texto[:i] + texto[i + 1:]

    # Error de adición
    elif tipo_error == "adición":
        i = random.randint(0, len(texto) - 1)
        letra_aleatoria = random.choice("abcdefghijklmnopqrstuvwxyz")
        return texto[:i] + letra_aleatoria + texto[i:]

    # Error de puntuación
    elif tipo_error == "puntuación":
        i = random.randint(0, len(texto) - 1)
        puntuacion_aleatoria = random.choice(",.!¿?")
        return texto[:i] + puntuacion_aleatoria + texto[i:]

    # Error de repetición
    elif tipo_error == "repetición":
        i = random.randint(0, len(texto) - 1)
```

```
164         return texto[:i] + texto[i] + texto[i:]
165
166     # Error de espaciado
167     elif tipo_error == "espaciado":
168         i = random.randint(0, len(texto) - 1)
169         if " " in texto:
170             return texto.replace(" ", "")
171         else:
172             return texto[:i] + " " + texto[i:]
173
174     # Error de mezcla de mayúsculas
175     elif tipo_error == "mezcla_mayúsculas":
176         return "".join(random.choice([c.upper(), c.lower()]) for c in texto)
177
178     # Error de símbolos
179     elif tipo_error == "símbolos":
180         simbolos = {"a": "@", "e": "3", "i": "!", "o": "0", "s": "$"}
181         return "".join(simbolos.get(c, c) for c in texto.lower())
182
183     # Error fonético
184     elif tipo_error == "fonético":
185         foneticos = {"c": "k", "k": "c", "s": "z", "z": "s"}
186         return "".join(foneticos.get(c, c) for c in texto.lower())
187
188     # Error de abreviación
189     elif tipo_error == "abreviación":
190         palabras = texto.split()
191         if len(palabras) > 2:
192             return palabras[0][0] + ". " + " ".join(palabras[1:])
193         return texto
194
195     # Error de mezcla de palabras
196     elif tipo_error == "mezcla_palabras":
197         palabras = texto.split()
198         random.shuffle(palabras)
199         return " ".join(palabras)
200
201 # Datos transaccionales
202 df_datos_transaccionales = df_datos_transaccionales_correctos.drop("concat",
        axis=1).copy()
203
204 # Añadimos los errores y valores ausentes
205 for index, row in df_datos_transaccionales.iterrows():
206     if random.random() < 0.70:
207         selected_columns = random.sample(list(df_datos_transaccionales.
```

```
    columns), 3)
208         df_datos_transaccionales.at[index, selected_columns[0]] =
    generar_datos_erroneos(row[selected_columns[0]])
209         df_datos_transaccionales.at[index, selected_columns[1]] = "NaN"
210         df_datos_transaccionales.at[index, selected_columns[2]] = "NaN"
211
212 # Concatenamos la información de cada fila
213 df_datos_transaccionales["concat"] = df_datos_transaccionales["DNI"] + " " +
    df_datos_transaccionales["Nombre"] + " " + df_datos_transaccionales["
    Apellido_1"] + " " + df_datos_transaccionales["Apellido_2"] + " " +
    df_datos_transaccionales["Email"] + " " + df_datos_transaccionales["Telé
    fono"]
214 df_datos_transaccionales
215
216 # Generación de la columna "error", que indica si el dato transaccional tiene
     un error
217 df_datos_transaccionales["error"] = (df_datos_transaccionales["concat"] !=
    df_datos_transaccionales_correctos["concat"]).astype(int)
218 df_datos_transaccionales
219
220 # Calidad inicial
221 calidad_inicial = accuracy_score(df_datos_transaccionales["concat"],
    df_datos_transaccionales_correctos["concat"])*100
222
223 print("Calidad inicial de los datos transaccionales: ",f"{calidad_inicial:.2f
    }","%")
224
225 # GENERACIÓN DEL PROMPT
226
227 # String de los datos maestros
228 columnas = ["DNI","Nombre","Apellido_1","Apellido_2","Email","Teléfono"]
229 text_datos_maestros = df_datos_maestros[columnas].to_csv(sep=";", index=False
    )
230 print(text_datos_maestros)
231
232 # String de los Datos transaccionales
233 columnas = ["DNI","Nombre","Apellido_1","Apellido_2","Email","Teléfono"]
234 text_datos_transaccionales = df_datos_transaccionales[columnas].to_csv(sep="
    ", index=False)
235 print(text_datos_transaccionales)
236
237 # Generación del prompt
238 # Parte A
239 prompt_parte_A = """Eres un experto en corrección de datos, especializado en
    la detección y estandarización de errores en datos transaccionales con
```

```
    información de personas. Además, eres experto en la alineación de datos
    transaccionales con datos maestros. Tengo dos conjuntos de datos en
    formato CSV con delimitador ";". El primero incluye datos maestros con el
    formato y atributos correctos de personas y sus datos personales. El
    segundo contiene datos transaccionales de personas que contienen errores y
     datos ausentes (NaN). Los errores pueden ser de diferentes tipos, como un
     mal formato. Tu tarea es corregir meticulosamente los datos
    transaccionales, basándote en los datos maestros, de manera que tengan el
    formato correcto y no haya datos ausentes (NaN). Aplica tu conocimiento
    para alinear y estandarizar los datos transaccionales con precisión. Asegú
    rate de que los valores que ya están estandarizados queden inalterados. No
     utilices herramientas de análisis ni Python, solo tu conocimiento.

Tu respuesta debe contener todos los datos transaccionales corregidos. El
    formato de la respuesta debe ser un CSV con cabecera, con delimitador ";"
    y con las columnas que se muestran a continuación:

DNI;Nombre;Apellido_1;Apellido_2;Email;Teléfono

MUY IMPORTANTE: Sólo incluye el CSV en tu respuesta. No incluyas espacios
    entre cada fila. No incluyas ninguna palabra adicional en tu respuesta. No
     incluyas explicaciones, informaciones, razonamientos o pasos seguidos.
MUY IMPORTANTE: Asegúrate que el número de filas de la respuesta es el
    correcto. No dupliques, ni te dejes valores.

Datos maestros:
"""

# Parte B
prompt_parte_B = """

Archivo CSV de datos transaccionales a corregir:
"""

# Parte C
prompt_parte_C = """

Ejemplo de respuesta:
DNI;Nombre;Apellido_1;Apellido_2;Email;Teléfono
91697831M;Miguel;Ortiz;Nieto;miguel.ortiz.nieto@yahoo.com;738427558
81213749J;Raquel;Álvarez;Blanco;raquel.alvarez.blanco@gmail.com;650538087
20102230T;Lorena;Peña;Soto;lorena.peña.soto@outlook.com;757957493


Archivo CSV de datos transaccionales corregido:"""
```

```
268
269 # Agrupamiento de strings
270 prompt = prompt_parte_A + text_datos_maestros +  prompt_parte_B +
        text_datos_transaccionales + prompt_parte_C
271 print(prompt)
272
273 # CONFIGURACIÓN DE APIS DE LLM
274
275 # OpenAI
276 OPENAI_API_KEY = XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
277 client = OpenAI(api_key=OPENAI_API_KEY)
278
279 # Google Vertex AI
280 vertexai.init(project=XXXXXXXXXXXXXXX, location=XXXXXXXXXXX)
281
282 # Hugging Face
283 HUGGING_FACE_API_KEY = XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
284
285 # INFERENCIA
286 use_GPT_4_Turbo = True     # Utilizar (True) o no (False) GPT-4 Turbo (OpenAI
        )
287 use_GPT_3_5_Turbo = True   # Utilizar (True) o no (False) GPT-3.5 Turbo (
        OpenAI)
288 use_Gemini_Pro = True      # Utilizar (True) o no (False) Gemini Pro (Google)
289 use_LLaMA_2_70B = True    # Utilizar (True) o no (False) LLaMA 2 70B Chat (
        Meta)
290
291 # Inferencia de GPT-4-Turbo (OpenAI)
292 if use_GPT_4_Turbo:
293   start_time = time.time()
294   response = client.chat.completions.create(
295     model="gpt-4-1106-preview",
296     messages=[
297       {"role": "user", "content": prompt}
298     ],
299     max_tokens=4000
300   )
301   end_time = time.time()
302   time_GPT_4_Turbo = end_time - start_time
303   text_datos_corregidos_GPT_4_Turbo = response.choices[0].message.content
304   print(text_datos_corregidos_GPT_4_Turbo)
305
306 # Inferencia de GPT-3.5 Turbo (OpenAI)
307 if use_GPT_3_5_Turbo:
308   start_time = time.time()
```

```
  response = client.chat.completions.create(
    model="gpt-3.5-turbo-0125",
    messages=[
      {"role": "user", "content": prompt}
    ],
    max_tokens=4000
  )
  end_time = time.time()
  time_GPT_3_5_Turbo = end_time - start_time
  text_datos_corregidos_GPT_3_5_Turbo = response.choices[0].message.content
  print(text_datos_corregidos_GPT_3_5_Turbo)

# Inferencia de Gemini Pro (Google)
if use_Gemini_Pro:
    generation_config = {
        "max_output_tokens": 4000
    }
    safety_settings ={
        HarmCategory.HARM_CATEGORY_UNSPECIFIED: HarmBlockThreshold.BLOCK_NONE
    ,
        HarmCategory.HARM_CATEGORY_HATE_SPEECH: HarmBlockThreshold.BLOCK_NONE
    ,
        HarmCategory.HARM_CATEGORY_DANGEROUS_CONTENT: HarmBlockThreshold.
    BLOCK_NONE,
        HarmCategory.HARM_CATEGORY_HARASSMENT: HarmBlockThreshold.BLOCK_NONE,
        HarmCategory.HARM_CATEGORY_SEXUALLY_EXPLICIT: HarmBlockThreshold.
    BLOCK_NONE,
    }
    model = GenerativeModel("gemini-pro", generation_config=generation_config
    , safety_settings=safety_settings)
    chat = model.start_chat()
    start_time = time.time()
    response = chat.send_message(prompt,safety_settings=safety_settings)
    end_time = time.time()
    time_Gemini_Pro = end_time - start_time
    text_datos_corregidos_Gemini_Pro = response.text
    print(text_datos_corregidos_Gemini_Pro)

# Inferencia de LLaMA 2 70B Chat (Meta)
if use_LLaMA_2_70B:
    client_llama_2 = InferenceClient(model="meta-llama/Llama-2-70b-chat-hf",
    token=HUGGING_FACE_API_KEY)
    start_time = time.time()
    response = client_llama_2.text_generation(prompt, max_new_tokens=4096)
    end_time = time.time()
```

```
348     time_LLaMA_2_70B = end_time - start_time
349     text_datos_corregidos_LLaMA_2_70B = response
350     print(text_datos_corregidos_LLaMA_2_70B)
351 
352 # GUARDADO DE RESULTADOS EN DATAFRAME
353 
354 # Resultados - GPT-4-Turbo (OpenAI)
355 if use_GPT_4_Turbo:
356     df_datos_corregidos_GPT_4_Turbo = pd.read_csv(StringIO(
    text_datos_corregidos_GPT_4_Turbo), delimiter=";",dtype=str)
357     df_datos_corregidos_GPT_4_Turbo["concat"] =
    df_datos_corregidos_GPT_4_Turbo["DNI"] + " " +
    df_datos_corregidos_GPT_4_Turbo["Nombre"] + " " +
    df_datos_corregidos_GPT_4_Turbo["Apellido_1"] + " " +
    df_datos_corregidos_GPT_4_Turbo["Apellido_2"] + " " +
    df_datos_corregidos_GPT_4_Turbo["Email"] + " " +
    df_datos_corregidos_GPT_4_Turbo["Teléfono"]
358 
359 # Resultados - GPT-3.5 Turbo (OpenAI)
360 if use_GPT_3_5_Turbo:
361     df_datos_corregidos_GPT_3_5_Turbo = pd.read_csv(StringIO(
    text_datos_corregidos_GPT_3_5_Turbo), delimiter=";",dtype=str)
362     df_datos_corregidos_GPT_3_5_Turbo["concat"] =
    df_datos_corregidos_GPT_3_5_Turbo["DNI"] + " " +
    df_datos_corregidos_GPT_3_5_Turbo["Nombre"] + " " +
    df_datos_corregidos_GPT_3_5_Turbo["Apellido_1"] + " " +
    df_datos_corregidos_GPT_3_5_Turbo["Apellido_2"] + " " +
    df_datos_corregidos_GPT_3_5_Turbo["Email"] + " " +
    df_datos_corregidos_GPT_3_5_Turbo["Teléfono"]
363 
364 # Resultados - Gemini Pro (Google)
365 if use_Gemini_Pro:
366     df_datos_corregidos_Gemini_Pro = pd.read_csv(StringIO(
    text_datos_corregidos_Gemini_Pro), delimiter=";",dtype=str)
367     df_datos_corregidos_Gemini_Pro["concat"] = df_datos_corregidos_Gemini_Pro
    ["DNI"] + " " + df_datos_corregidos_Gemini_Pro["Nombre"] + " " +
    df_datos_corregidos_Gemini_Pro["Apellido_1"] + " " +
    df_datos_corregidos_Gemini_Pro["Apellido_2"] + " " +
    df_datos_corregidos_Gemini_Pro["Email"] + " " +
    df_datos_corregidos_Gemini_Pro["Teléfono"]
368 
369 # Resultados - LLaMA 2 70B Chat (Meta)
370 if use_LLaMA_2_70B:
371     df_datos_corregidos_LLaMA_2_70B = pd.read_csv(StringIO(
    text_datos_corregidos_LLaMA_2_70B), delimiter=";",dtype=str)
```

```
372     df_datos_corregidos_LLaMA_2_70B["concat"] =
    df_datos_corregidos_LLaMA_2_70B["DNI"] + " " +
    df_datos_corregidos_LLaMA_2_70B["Nombre"] + " " +
    df_datos_corregidos_LLaMA_2_70B["Apellido_1"] + " " +
    df_datos_corregidos_LLaMA_2_70B["Apellido_2"] + " " +
    df_datos_corregidos_LLaMA_2_70B["Email"] + " " +
    df_datos_corregidos_LLaMA_2_70B["Teléfono"]
373
374 # Índices de las filas que contenían error y las que no
375 index_error_1 = df_datos_transaccionales[df_datos_transaccionales["error"
    ]==1].index
376 index_error_0 = df_datos_transaccionales[df_datos_transaccionales["error"
    ]==0].index
377
378 # EVALUACIÓN
379
380 # Evaluación de GPT-4-Turbo (OpenAI)
381 if use_GPT_4_Turbo:
382     calidad_posterior_GPT_4_Turbo = accuracy_score(
    df_datos_transaccionales_correctos["concat"],
    df_datos_corregidos_GPT_4_Turbo["concat"])*100
383     exactitud_datos_erroneos_GPT_4_Turbo = accuracy_score(
    df_datos_transaccionales_correctos.loc[index_error_1]["concat"],
    df_datos_corregidos_GPT_4_Turbo.loc[index_error_1]["concat"])*100
384     exactitud_datos_correctos_GPT_4_Turbo = accuracy_score(
    df_datos_transaccionales_correctos.loc[index_error_0]["concat"],
    df_datos_corregidos_GPT_4_Turbo.loc[index_error_0]["concat"])*100
385
386 # Evaluación de GPT-3.5 Turbo (OpenAI)
387 if use_GPT_3_5_Turbo:
388     calidad_posterior_GPT_3_5_Turbo = accuracy_score(
    df_datos_transaccionales_correctos["concat"],
    df_datos_corregidos_GPT_3_5_Turbo["concat"])*100
389     exactitud_datos_erroneos_GPT_3_5_Turbo = accuracy_score(
    df_datos_transaccionales_correctos.loc[index_error_1]["concat"],
    df_datos_corregidos_GPT_3_5_Turbo.loc[index_error_1]["concat"])*100
390     exactitud_datos_correctos_GPT_3_5_Turbo = accuracy_score(
    df_datos_transaccionales_correctos.loc[index_error_0]["concat"],
    df_datos_corregidos_GPT_3_5_Turbo.loc[index_error_0]["concat"])*100
391
392 # Evaluación de Gemini Pro (Google)
393 if use_Gemini_Pro:
394     calidad_posterior_Gemini_Pro = accuracy_score(
    df_datos_transaccionales_correctos["concat"],
    df_datos_corregidos_Gemini_Pro["concat"])*100
```

```
395     exactitud_datos_erroneos_Gemini_Pro = accuracy_score(
    df_datos_transaccionales_correctos.loc[index_error_1]["concat"],
    df_datos_corregidos_Gemini_Pro.loc[index_error_1]["concat"])*100
396     exactitud_datos_correctos_Gemini_Pro = accuracy_score(
    df_datos_transaccionales_correctos.loc[index_error_0]["concat"],
    df_datos_corregidos_Gemini_Pro.loc[index_error_0]["concat"])*100
397
398 # Evaluación de LLaMA 2 70B Chat (Meta)
399 if use_LLaMA_2_70B:
400     calidad_posterior_LLaMA_2_70B = accuracy_score(
    df_datos_transaccionales_correctos["concat"],
    df_datos_corregidos_LLaMA_2_70B["concat"])*100
401     exactitud_datos_erroneos_LLaMA_2_70B = accuracy_score(
    df_datos_transaccionales_correctos.loc[index_error_1]["concat"],
    df_datos_corregidos_LLaMA_2_70B.loc[index_error_1]["concat"])*100
402     exactitud_datos_correctos_LLaMA_2_70B = accuracy_score(
    df_datos_transaccionales_correctos.loc[index_error_0]["concat"],
    df_datos_corregidos_LLaMA_2_70B.loc[index_error_0]["concat"])*100
403
404 # IMPRESIÓN DE RESULTADOS
405
406 print("Calidad inicial de los datos transaccionales: 0.00%\n")
407
408 print("Resultados de los modelos: ")
409
410 # GPT-4-Turbo (OpenAI)
411 if use_GPT_4_Turbo:
412     print("GPT-4 Turbo:\t\tCalidad posterior:",calidad_posterior_GPT_4_Turbo,
413           "%    Incremento calidad:",calidad_posterior_GPT_4_Turbo,
414           "%    Exactitud (accuracy) en datos erroneos:",
    exactitud_datos_erroneos_GPT_4_Turbo,
415           "%    Exactitud (accuracy) en datos correctos:",
    exactitud_datos_correctos_GPT_4_Turbo,
416           "%    Tiempo:",time_GPT_4_Turbo,"s")
417 # GPT-3.5 Turbo (OpenAI)
418 if use_GPT_3_5_Turbo:
419     print("GPT-3.5 Turbo:\t\tCalidad posterior:",
    calidad_posterior_GPT_3_5_Turbo,
420           "%    Incremento calidad:",calidad_posterior_GPT_3_5_Turbo,
421           "%    Exactitud (accuracy) en datos erroneos:",
    exactitud_datos_erroneos_GPT_3_5_Turbo,
422           "%    Exactitud (accuracy) en datos correctos:",
    exactitud_datos_correctos_GPT_3_5_Turbo,
423           "%    Tiempo:",time_GPT_3_5_Turbo,"s")
424 # Gemini Pro (Google)
```

```
425 if use_Gemini_Pro:
426     print("Gemini Pro:\tCalidad posterior:",calidad_posterior_Gemini_Pro,
427           "%    Incremento calidad:",calidad_posterior_Gemini_Pro,
428           "%    Exactitud (accuracy) en datos erroneos:",
    exactitud_datos_erroneos_Gemini_Pro,
429           "%    Exactitud (accuracy) en datos correctos:",
    exactitud_datos_correctos_Gemini_Pro,
430           "%    Tiempo:",time_Gemini_Pro,"s")
431
432 # LLaMA 2 70B Chat (Meta)
433 if use_LLaMA_2_70B:
434     print("LLaMA_2_70B_Chat:\tCalidad posterior:",
    calidad_posterior_LLaMA_2_70B,
435           "%    Incremento calidad:",calidad_posterior_LLaMA_2_70B,
436           "%    Exactitud (accuracy) en datos erroneos:",
    exactitud_datos_erroneos_LLaMA_2_70B,
437           "%    Exactitud (accuracy) en datos correctos:",
    exactitud_datos_correctos_LLaMA_2_70B,
438           "%    Tiempo:",time_LLaMA_2_70B,"s")
```