Práctica Profesional Supervisada

Blanco, Fernando Jose

Plataforma de análisis de datos asistido por RAG en un entorno multiagente

Instituto de Ingeniería y Agronomía

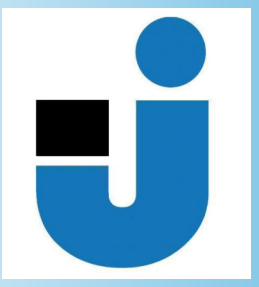
Carrera: Ingeniería en Informática

2025

Universidad Nacional Arturo Jauretche

Instituto de Ingeniería y Agronomía

Carrera de Ingeniería en Informática



PRÁCTICA PROFESIONAL SUPERVISADA
Informe final

Plataforma de análisis de datos asistido por RAG en un entorno multiagente

Fernando Jose Blanco

Florencio Varela, Noviembre 2025

Estudiante:

Nombres y Apellidos: Fernando Jose Blanco
DNI: 41.080.885
Correo electrónico: fernando.blanco004@gmail.com
Carrera: Ingeniería en Informática

Organización donde se realiza la Práctica Profesional Supervisada:

Nombre o Razón Social: Universidad Nacional Arturo Jauretche
Dirección: Av. Calchaquí 6200, Florencio Varela, (1888) Buenos Aires, Argentina
Teléfono: +54 11 4275-6100
Sector: Programa Tecnologías de la Información y la Comunicación (TIC) en aplicaciones de interés social, Instituto de Ingeniería y Agronomía

Tutor organizacional

Nombres y Apellidos: Prof. Ing Lucas Maximiliano Olivera
Correo electrónico: lolivera@unaj.edu.ar

Docente Supervisor

Nombres y Apellidos: Prof. Dr Marcelo Angel Cappelletti
Correo electrónico: mcappelletti@unaj.edu.ar

Coordinador de la Carrera de Ingeniería en Informática

Nombres y Apellidos: Dr. Ing. Morales, Martin
Correo electrónico: martin.morales@unaj.edu.ar

Resumen

El presente Proyecto Integrador Profesionalizante (en adelante, PIP) tiene como objetivo desarrollar una aplicación web orientada al análisis de datos mediante el uso de agentes basados en modelos de lenguaje de gran escala (LLM, por sus siglas en inglés). La propuesta busca ofrecer una interfaz interactiva que permita a los usuarios subir, visualizar y analizar archivos en formato Excel o CSV, posibilitando la formulación de consultas en lenguaje natural para obtener información de manera ágil y comprensible.
La aplicación contará con un chat que actuará como mediador entre el usuario y los datos, brindando la posibilidad de consultar características del dataset como cantidad de filas, columnas o registros, así como generar visualizaciones dinámicas que faciliten la interpretación de la información. Estas visualizaciones permitirán identificar patrones, relaciones y tendencias de los datos de forma rápida y efectiva.
El sistema se apoyará en uno o varios agentes LLM con capacidad de razonamiento y toma de decisiones, encargados de determinar cuándo ejecutar herramientas de análisis, generar código para la creación de gráficos o sintetizar información textual. El enfoque modular del proyecto permitirá integrar múltiples LLMs con distintas responsabilidades, optimizando así la interacción y el rendimiento general del sistema.
En conjunto, este PIP busca desarrollar una herramienta flexible e intuitiva que junte procesamiento de lenguaje natural, análisis de datos y visualización automatizada, contribuyendo a la democratización del acceso al análisis de información mediante el uso de tecnologías de inteligencia artificial.

Palabras clave: Análisis de datos, Modelos de lenguaje grande (LLM), Inteligencia artificial (IA), Visualización interactiva.

Abstract

The present Professional Integrative Project (hereinafter PIP) aims to develop a web application focused on data analysis through the use of agents based on large language models (LLMs). The proposal seeks to provide an interactive interface that allows users to upload, visualize, and analyze Excel or CSV files, enabling natural language queries to obtain information quickly and comprehensibly.

The application will include a chat component that acts as a mediator between the user and the data, allowing inquiries about dataset characteristics such as the number of rows, columns, or records, as well as generating dynamic visualizations to facilitate information interpretation. These visualizations will help identify data patterns, relationships, and trends efficiently.

The system will rely on one or more LLM-based agents with reasoning and decision-making capabilities, responsible for determining when to execute analytical tools, generate code for chart creation, or synthesize textual information. The modular design of the project will allow the integration of multiple LLMs with distinct responsibilities, thereby optimizing interaction and overall system performance.

Overall, this PIP seeks to develop a flexible and intuitive tool that combines natural language processing, data analysis, and automated visualization, contributing to the democratization of data analysis through the use of artificial intelligence technologies.

Keywords: Data analysis, Large Language Models (LLM), Artificial intelligence (AI), Interactive visualization.

Dedicatorias y agradecimientos.

Al finalizar esta etapa tan significativa de mi formación, quiero reconocer y agradecer a todas las personas que contribuyeron a hacer realidad este proyecto.

A mi familia, por creer en mí desde el primer día y sostenerme con su amor incondicional. Su respaldo fue la fuerza que necesité para superar cada obstáculo y seguir adelante cuando las circunstancias se volvían complejas.

A mi pareja, cuyo amor, paciencia y acompañamiento fueron esenciales en este recorrido. Gracias por entender las largas horas de dedicación, por motivarme cuando más lo necesitaba y por celebrar cada pequeño avance junto a mí.

A mis amigos, por estar presentes a pesar de mis ausencias, por comprender las prioridades del momento y por ofrecerme esos espacios de desconexión que me permitieron recuperar energías y mantener el equilibrio.

Extiendo mi reconocimiento a Lucas Olivera, mi tutor organizacional de este proyecto, quien con su experiencia, generosidad, disponibilidad constante y orientación precisa me ayudó a navegar los desafíos técnicos del desarrollo. Su apertura para compartir conocimientos y su guía en el campo de la inteligencia artificial aplicada fueron determinantes para alcanzar los objetivos propuestos.

A Marcelo Ángel Cappelletti, mi docente supervisor y director del proyecto de investigación que alberga este PIP, le agradezco por impulsar mi participación en la beca BIEI, por su acompañamiento académico y por impulsar un espacio de trabajo donde pude explorar tecnologías nuevas sin miedo a equivocarme y aprender en el proceso.

A la Universidad Nacional Arturo Jauretche y al cuerpo docente de Ingeniería en Informática, por proporcionarme las herramientas teóricas y prácticas que hicieron posible este trabajo. El compromiso demostrado por cada profesor fue clave para mi crecimiento como profesional.

A mis compañeros y compañeras de la carrera, por hacer de este trayecto una experiencia compartida. El aprendizaje colaborativo, las consultas mutuas y el apoyo entre nosotros transformaron el desafío individual en un logro colectivo.

A todos ustedes, gracias de corazón.

Índice

1. Introducción

En los últimos años, el avance acelerado de la inteligencia artificial y, en particular, de los modelos de lenguaje de gran escala, transformó la forma en que las personas interactúan con la información. Al mismo tiempo, el análisis de datos se ha consolidado como una herramienta esencial tanto en ámbitos académicos como productivos, permitiendo obtener conocimiento a partir de grandes volúmenes de información. Sin embargo, la creciente complejidad de estas tecnologías genera una brecha significativa entre sus capacidades y la accesibilidad real que tienen los usuarios para aprovecharlas plenamente.

En entornos educativos y de investigación, esta brecha se hace especialmente evidente. Docentes, estudiantes e investigadores cuentan con datos valiosos, pero muchas veces no tienen las herramientas o conocimientos técnicos necesarios para realizar análisis profundos o generar visualizaciones útiles. Paralelamente, la irrupción de la inteligencia artificial generativa abrió nuevas posibilidades para simplificar estas tareas mediante el uso del lenguaje natural como medio de interacción, acercando capacidades avanzadas a usuarios sin experiencia previa en programación o análisis estadístico.

Este escenario plantea un desafío y una oportunidad, el cual es el de desarrollar soluciones que contemplen accesibilidad, automatización y potencia analítica, integrando la IA generativa como un puente entre el conocimiento humano y las herramientas computacionales. En este contexto se enmarca el presente proyecto.

A pesar del crecimiento sostenido del análisis de datos y de las herramientas digitales disponibles, una gran parte de los usuarios continúa enfrentando dificultades para interpretar y trabajar con información de manera autónoma. En contextos académicos, esta situación se observa con frecuencia en docentes e investigadores que, si bien poseen un dominio sólido del contenido disciplinar, no cuentan necesariamente con conocimientos avanzados en programación, estadística o manejo de entornos especializados como notebooks científicos o bibliotecas de visualización. Esta limitación genera una dependencia constante de especialistas técnicos y ralentiza los procesos de análisis, interpretación y toma de decisiones.

La aparición de modelos de inteligencia artificial, que son capaces de comprender instrucciones en lenguaje natural, ofrece un punto de inflexión. Sin embargo, su utilización efectiva aún requiere configuraciones complejas, integración con herramientas externas y un entendimiento básico de conceptos que no siempre

están al alcance de todos los usuarios. La falta de una interfaz accesible y de un entorno de trabajo guiado impide aprovechar plenamente el potencial de estas tecnologías emergentes.

En este escenario, surge la necesidad de una solución que permita reducir esa barrera de entrada al análisis de datos mediante un sistema que funcione como intermediario entre el usuario y las capacidades avanzadas de la inteligencia artificial. Una herramienta capaz de comprender consultas en lenguaje natural, procesar los datos automáticamente y presentar resultados interpretables que contribuiría directamente a democratizar el acceso al análisis y a fortalecer prácticas académicas basadas en evidencia.

Frente a esta problemática, el presente proyecto se propone desarrollar una solución que acerque las capacidades de la inteligencia artificial generativa al usuario común, permitiendo que cualquier persona pueda interactuar con sus datos mediante lenguaje natural. El objetivo central es diseñar un sistema que funcione como un asistente inteligente capaz de interpretar consultas, analizar información y generar visualizaciones de manera automática, sin requerir conocimientos técnicos avanzados.

El propósito del proyecto no es únicamente facilitar el análisis de datos, sino también ofrecer un entorno intuitivo que promueva la autonomía del usuario y reduzca la complejidad asociada al uso de modelos de lenguaje. La propuesta busca transformar consultas cotidianas en acciones concretas, permitiendo obtener información relevante de forma rápida, accesible y comprensible.

De esta manera, el proyecto se posiciona como una contribución orientada a democratizar el acceso a tecnologías emergentes, impulsando nuevas formas de interacción con los datos y potenciando el uso de IA generativa en contextos académicos y profesionales.

El sistema desarrollado se plantea como una herramienta de apoyo para usuarios que requieren analizar datos, obtener métricas descriptivas o generar visualizaciones sin necesidad de interactuar con código o herramientas concretas. A través de una interfaz accesible, el proyecto permite que las personas formulen consultas en lenguaje natural y reciban como respuesta análisis claros, interpretaciones contextualizadas y representaciones gráficas generadas automáticamente.

El alcance de la solución se centra en brindar una experiencia de uso fluida y guiada, donde el usuario pueda cargar sus archivos de datos, realizar preguntas

específicas y obtener resultados que faciliten la comprensión de la información. El sistema actúa como un intermediario capaz de traducir las necesidades del usuario en operaciones analíticas, priorizando siempre la simplicidad y la claridad de los resultados.

Si bien la herramienta fue concebida inicialmente para el ámbito universitario, su diseño general permite que pueda ser utilizada en diversos escenarios donde la interpretación de datos sea una actividad habitual. De esta forma, el proyecto no solo atiende una necesidad inmediata del entorno académico, sino que también abre la posibilidad de extender su aplicación a organizaciones, docentes, estudiantes o profesionales que requieran acceder de manera rápida y accesible a análisis basados en datos.

2. Glosario

2.1 Términos generales

- **Agente Inteligente:** Sistema de software basado en IA capaz de tomar decisiones autónomas y ejecutar herramientas según objetivos definidos.
- **Análisis Exploratorio de Datos (EDA)**: Proceso de examinar datasets para descubrir patrones, detectar anomalías y verificar hipótesis mediante estadísticas y visualizaciones.
- **Arquitectura Cliente-Servidor:** Modelo de diseño donde el cliente solicita servicios y el servidor los proporciona, separando la interfaz de la lógica de negocio.
- **Arquitectura Modular**: Diseño de software dividido en componentes independientes y reutilizables que facilitan mantenimiento y escalabilidad.
- **Backend**: Capa del sistema que procesa la lógica de negocio, gestiona datos y se comunica con servicios externos.
- **Checkpoint**: Punto de guardado del estado del sistema que permite recuperar el contexto en interacciones futuras.
- **Dataset**: Conjunto estructurado de datos organizados para su análisis, típicamente en formato tabular.
- **Democratización del Acceso**: Proceso de hacer herramientas tecnológicas complejas accesibles para usuarios sin conocimientos técnicos avanzados.
- **Descripción Semántica**: Texto generado automáticamente que describe el contenido, propósito y características de un dataset.

- **EDAI**: Explorador de Datos Asistido por Inteligencia Artificial. Nombre del sistema desarrollado en este proyecto.
- **Embedding**: Representación numérica vectorial de texto que captura su significado semántico para procesamiento computacional.
- **Endpoint**: Punto de acceso específico de una API que responde a solicitudes concretas (ej: /api/chat, /api/documents).
- **Fallback**: Mecanismo alternativo que se activa automáticamente cuando falla el método principal de ejecución.
- **Frontend**: Interfaz visual con la que el usuario interactúa directamente para usar el sistema.
- **Grafo**: Estructura de nodos conectados mediante aristas que representa flujos de ejecución en el sistema.
- **Hashing**: Proceso de generar un identificador único (hash) a partir de datos mediante algoritmos criptográficos como SHA-256.
- **Historial Conversacional**: Registro cronológico completo de interacciones previas entre usuario y sistema.
- **Interfaz de Usuario**: Punto de interacción visual entre el usuario y el sistema que facilita la comunicación bidireccional.
- **Latencia**: Tiempo de espera entre el envío de una solicitud y la recepción de su respuesta.
- **Lenguaje Natural**: Forma de comunicación humana cotidiana que el sistema puede comprender y procesar.
- **Memoria Conversacional**: Capacidad del sistema para recordar interacciones previas y mantener contexto entre sesiones diferentes.
- **Metadatos**: Información descriptiva sobre los datos principales incluyendo estructura, tipos, origen y dimensiones.
- **Modelo de Lenguaje**: Sistema de IA entrenado para comprender, interpretar y generar lenguaje natural de manera coherente.
- **Nodo**: Unidad funcional dentro de un grafo que ejecuta una tarea específica del flujo de procesamiento.
- **Orquestador**: Componente central que coordina la ejecución secuencial de múltiples agentes o servicios del sistema.
- **Persistencia**: Capacidad de mantener datos almacenados de forma permanente incluso después de cerrar la aplicación.
- **Prompt**: Instrucción o consulta en lenguaje natural enviada a un modelo de lenguaje para obtener una respuesta específica.
- **Prompt Chaining**: Técnica de encadenar múltiples prompts donde la salida de uno alimenta la entrada del siguiente.
- **Semántica**: Significado e interpretación del contenido textual más allá de su estructura sintáctica superficial.

- **Sistema Multiagente**: Arquitectura con múltiples agentes inteligentes que colaboran coordinadamente para resolver tareas complejas.
- **Thread**: Hilo de conversación independiente con su propio contexto, historial e identificador único.
- **Token**: Unidad mínima de texto procesada por modelos de lenguaje, puede ser palabra, subpalabra o carácter.
- **Tokenización**: Proceso de dividir texto en unidades mínimas (tokens) para su procesamiento por modelos de lenguaje.
- **Tool**: Función especializada que un agente puede invocar para realizar tareas específicas como análisis, visualización o consultas.
- **Transformer**: Arquitectura de red neuronal basada en mecanismos de atención, fundamento tecnológico de los LLMs modernos.
- **Ventana de Contexto**: Cantidad máxima de tokens que un modelo de lenguaje puede procesar simultáneamente en una consulta.
- **Visualización de Datos**: Representación gráfica de información mediante gráficos, tablas y diagramas para facilitar su comprensión.

2.2 Acrónimos

- **API**: Interfaz de Programación de Aplicaciones (Application Programming Interface). Permite la comunicación entre diferentes sistemas de software.
- **BIEI**: Beca de Inicio en Investigación para Estudiantes. Programa de becas de la UNAJ para estudiantes de pregrado y grado.
- **CSV**: Valores Separados por Comas (Comma-Separated Values). Formato de archivo para datos tabulares.
- **EDA**: Análisis Exploratorio de Datos (Exploratory Data Analysis). Proceso de examinar datos para identificar patrones y características.
- **IA**: Inteligencia Artificial. Capacidad de sistemas computacionales para realizar tareas que requieren inteligencia humana.
- **LLM**: Modelo de Lenguaje de Gran Escala (Large Language Model). Sistema de IA entrenado con grandes cantidades de texto.
- **MVP**: Producto Mínimo Viable (Minimum Viable Product). Versión inicial funcional de un producto.
- **PIP**: Proyecto Integrador Profesionalizante. Trabajo final de carrera en la UNAJ.
- **PLN**: Procesamiento de Lenguaje Natural. Campo de IA que permite a las máquinas comprender lenguaje humano.
- **SQL**: Lenguaje de Consulta Estructurado (Structured Query Language). Lenguaje para gestionar bases de datos relacionales.

- **UNAJ**: Universidad Nacional Arturo Jauretche. Institución donde se desarrolla el proyecto.

3. Marco Teórico

3.1 Historia y evolución de la inteligencia artificial generativa

La inteligencia artificial generativa es el resultado de décadas de avances en el campo del aprendizaje automático y del procesamiento del lenguaje natural. Su desarrollo no fue abrupto; por el contrario, surgió a partir de una serie de hitos que marcaron progresivamente la forma en que las máquinas aprenden, representan y producen información.

Los primeros antecedentes se remontan a mediados del siglo XX, cuando surgieron los estudios iniciales sobre redes neuronales y modelos estadísticos del lenguaje. Durante ese período, la capacidad de las computadoras para trabajar con texto era limitada y se basaba mayormente en reglas predefinidas. Estos sistemas, conocidos como enfoques simbólicos, permitían procesamientos muy puntuales, pero estaban lejos de generar lenguaje de forma autónoma. La inteligencia artificial generativa, tal como se la conoce hoy, era todavía un concepto distante.

El punto de inflexión ocurrió con la consolidación del aprendizaje profundo a partir de la década de 2010, que permitió entrenar redes neuronales más grandes y complejas. Modelos como Word2Vec (2013) introdujeron una forma más rica de representar el texto mediante vectores, capturando relaciones semánticas que previamente resultaban inalcanzables. Simultáneamente, las Recurrent Neural Networks (RNN) y sus variantes, como las LSTM (Long Short-Term Memory), se convirtieron en herramientas fundamentales para procesar secuencias de texto, habilitando las primeras aproximaciones a modelos generativos más robustos.

Sin embargo, estas tecnologías aún enfrentaban limitaciones importantes, como la dificultad para capturar dependencias de largo alcance en el texto y el elevado costo computacional de su entrenamiento. Fue recién en 2017 cuando se produjo el cambio más significativo en la historia reciente de la inteligencia artificial, la introducción de la arquitectura Transformer, presentada por Vaswani et al. en el artículo “Attention Is All You Need”. Este modelo reemplazó las estructuras recurrentes por mecanismos de atención capaces de procesar información de manera paralela y contextual, inaugurando una nueva etapa en el procesamiento del lenguaje natural.

A partir de este avance, la evolución de los modelos generativos se aceleró exponencialmente. En 2018 se lanzó GPT-1, el primer modelo de la serie “Generative Pre-trained Transformer”, que demostró las ventajas del preentrenamiento a gran escala seguido de una fase de ajuste fino. Luego, aparecieron GPT-2 (2019), notable por su capacidad para generar texto coherente y creativo, y GPT-3 (2020), que se convirtió en un hito global debido a su enorme cantidad de parámetros y su gran capacidad para realizar tareas complejas con una mínima instrucción en lenguaje natural.

Este progreso dio lugar a una nueva categoría de modelos, los Modelos de Lenguaje de Gran Escala (LLMs, por sus siglas en inglés), entrenados con cantidades masivas de datos y diseñados para interpretar, razonar y producir lenguaje con un nivel de naturalidad similar al humano. Posteriormente, surgieron alternativas abiertas como LLaMA, Falcon y Mistral, así como propuestas comerciales avanzadas como PaLM, Claude o Gemini, que consolidaron un ecosistema diverso de modelos capaces de abordar múltiples tareas generativas.

En la actualidad, los LLMs ya no se limitan a generar texto, sino que participan en un amplio conjunto de aplicaciones como lo son: asistentes conversacionales, análisis de información, generación de código, traducción, análisis de datos, audiovisuales, sistemas de recomendación y herramientas creativas. Además, comenzaron a incorporar mecanismos para interactuar con herramientas externas, ejecutar código y manejar información estructurada, ampliando todavía más su alcance y utilidad práctica.

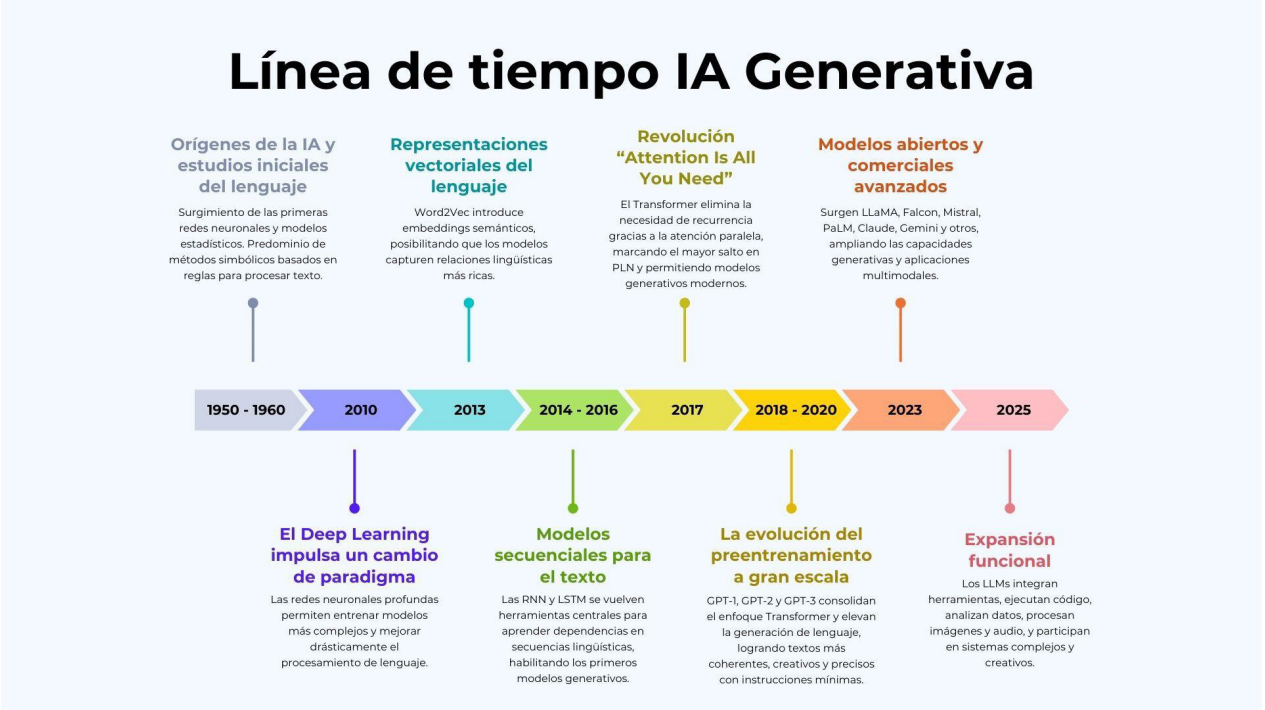


Figura 3.1. Línea de tiempo de la IA Generativa

Este recorrido histórico demuestra que la inteligencia artificial generativa no es un fenómeno aislado, sino la culminación de avances progresivos en representación del lenguaje, arquitectura de modelos y disponibilidad de datos y capacidad computacional. Comprender esta evolución es fundamental para contextualizar el proyecto desarrollado, ya que las capacidades actuales de los modelos de lenguaje son posibles gracias a esta trayectoria tecnológica que redefinió el modo en que las máquinas procesan y generan información.

3.2 Modelos de Lenguaje de Gran Escala (LLMs)

Los modelos de lenguaje de gran escala, conocidos como LLMs por sus siglas en inglés (Large Language Models), representan uno de los avances más significativos en el campo de la inteligencia artificial contemporánea. Estos modelos son sistemas entrenados con enormes cantidades de texto proveniente de libros, artículos, páginas web y otros recursos digitales, lo que les permite aprender patrones lingüísticos, estructuras sintácticas, relaciones semánticas y, en muchos casos, conocimientos generales sobre el mundo. Como resultado, pueden generar, interpretar y transformar lenguaje humano con un nivel de fluidez y coherencia que hasta hace pocos años parecía inalcanzable.

En esencia, un modelo de lenguaje es un sistema capaz de predecir la probabilidad de una palabra dentro de un contexto. Sin embargo, los LLMs modernos han trascendido esta definición tradicional gracias al empleo de arquitecturas cada vez más complejas, como los Transformers, y a la disponibilidad de recursos computacionales masivos. Esto les permite abordar tanto tareas simples de predicción de texto, como también actividades más avanzadas, como responder preguntas, generar resúmenes, elaborar código, analizar datos, traducir entre idiomas o realizar inferencias a partir de información incompleta.

La característica que distingue a los LLMs es su escala. Mientras que los modelos tradicionales contenían millones de parámetros, los LLMs actuales cuentan con miles de millones o incluso billones de parámetros. Esta escala les permite capturar relaciones de alta complejidad y adquirir habilidades emergentes que no estaban explícitamente programadas, sino que surgen de los patrones estadísticos presentes en los datos con los que fueron entrenados. Por este motivo, son modelos no supervisados en gran parte de su entrenamiento inicial, lo que significa que aprenden a partir de enormes corpus de texto sin necesidad de etiquetas predefinidas.

Además de su tamaño, otro aspecto clave de los LLMs es su versatilidad. A diferencia de modelos tradicionales entrenados para una tarea específica, los LLMs pueden adaptarse dinámicamente a una gran variedad de situaciones mediante simples instrucciones en lenguaje natural. Esta capacidad, conocida como “in-context learning”, permite que el modelo ajuste su comportamiento a partir de ejemplos proporcionados directamente por el usuario, sin necesidad de modificar sus parámetros internos.

En los últimos años, los LLMs también han incorporado mecanismos para trabajar más allá del texto. Algunos modelos pueden procesar imágenes, audio o señales multimodales, mientras que otros pueden utilizar herramientas externas, ejecutar fragmentos de código o interactuar con sistemas complejos. Estas capacidades ampliadas han dado lugar al concepto de “modelos fundacionales”, es decir, modelos que sirven como base para una amplia gama de aplicaciones especializadas.

Dentro del contexto del presente proyecto, los LLMs cumplen un rol fundamental, ya que son los encargados de interpretar las consultas en lenguaje natural realizadas por el usuario, razonar sobre ellas y transformar esa información en acciones concretas relacionadas con el análisis de datos. Su capacidad para comprender intenciones, contextualizar instrucciones y generar explicaciones comprensibles los convierte en una pieza esencial del sistema, permitiendo reducir la brecha entre el

usuario y las operaciones técnicas necesarias para obtener información significativa a partir de los datos cargados.

Los modelos de lenguaje de gran escala representan una herramienta poderosa y flexible que redefine la interacción entre seres humanos y sistemas computacionales. Su evolución continúa avanzando de manera acelerada, impulsada por mejoras en la arquitectura, técnicas de entrenamiento más eficientes y una creciente disponibilidad de datos. Comprender su funcionamiento y potencial resulta indispensable para interpretar las posibilidades y limitaciones del sistema desarrollado en este proyecto.

3.3 Arquitectura Transformer

La arquitectura Transformer constituye uno de los avances más influyentes en la historia reciente del aprendizaje profundo y es la base sobre la cual se desarrollan los modelos de lenguaje modernos. Introducida por Vaswani et al. En 2017 en el trabajo “Attention Is All You Need”, esta arquitectura marcó un cambio de paradigma al reemplazar completamente los mecanismos recurrentes tradicionales. Este enfoque permitió superar limitaciones históricas relacionadas con la eficiencia, la paralelización y la dificultad para capturar dependencias a larga distancia en secuencias textuales.

Antes del Transformer, modelos como las RNN y las LSTM procesaban el texto palabra por palabra, en un orden estrictamente secuencial. Este procesamiento lineal dificultaba comprender relaciones entre palabras distantes y hacía que los modelos fueran poco eficientes en el entrenamiento. El Transformer solucionó estos problemas mediante la introducción de un mecanismo central: la autoatención (self-attention).

3.3.1. Mecanismo de autoatención (self-attention)

La autoatención permite que el modelo analice simultáneamente todas las palabras de una secuencia y determine cuáles son más relevantes para interpretar el significado de cada token. En lugar de avanzar palabra por palabra, el modelo compara cada token con todos los demás, asignando pesos que indican la importancia contextual de cada relación.

Este enfoque posibilita comprender dependencias largas, ambigüedades y matices semánticos complejos que antes requerían estructuras mucho más costosas.

Además, habilita el procesamiento paralelo, reduciendo significativamente los tiempos de entrenamiento.

3.3.2. Embeddings y codificación posicional

Para poder trabajar con texto, el Transformer convierte cada palabra o token en un vector numérico mediante un embedding, el cual captura propiedades sintácticas y semánticas aprendidas durante el entrenamiento.

A diferencia de las redes recurrentes, el Transformer no procesa los datos siguiendo un orden natural, por lo que necesita un mecanismo que incorpore información sobre la posición de cada token. Esto se logra mediante la codificación posicional (positional encoding), que agrega a cada embedding una señal matemática que indica su ubicación dentro de la secuencia. Gracias a esto, el modelo puede preservar el orden de las palabras y comprender estructuras como sujeto-verbo-objeto o la relación entre palabras que dependen una de la otra.

3.3.3. Bloques del encoder y del decoder

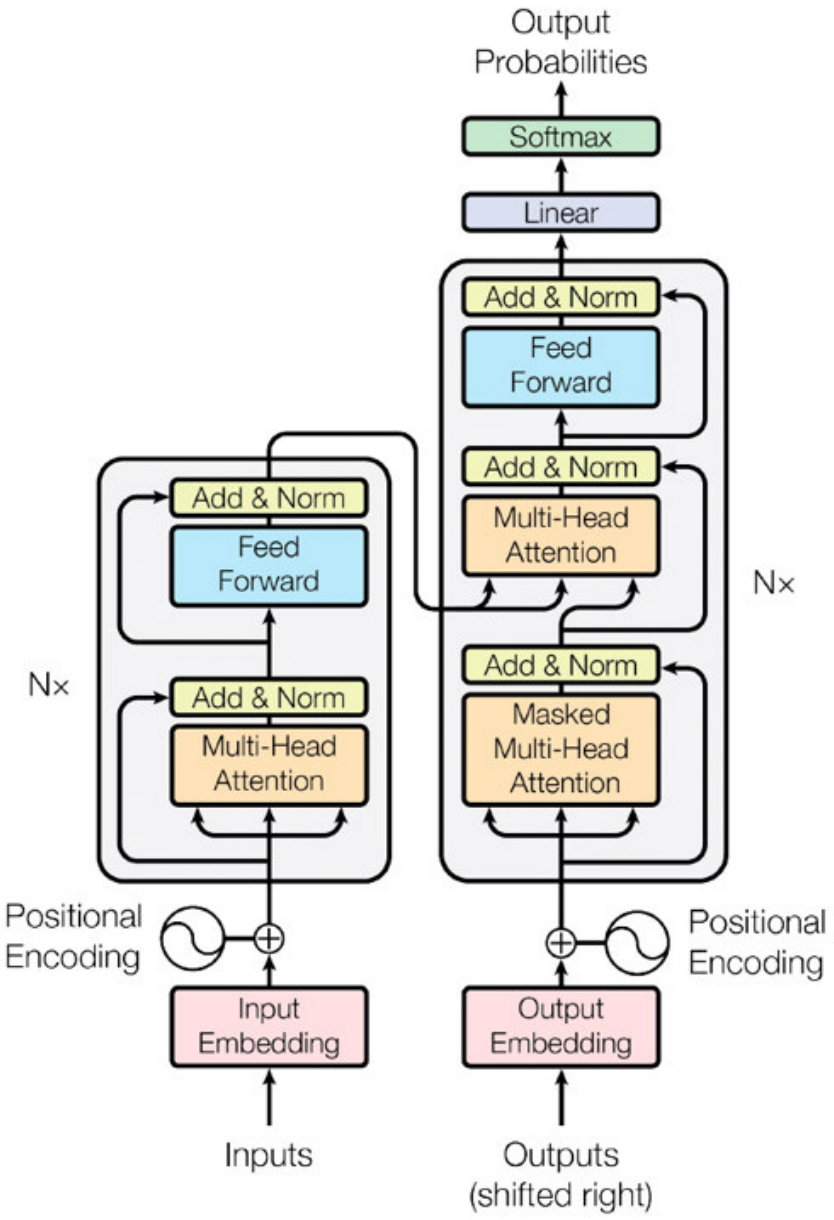


Figura 3.3. Arquitectura de transformer

El Transformer se organiza en dos componentes principales:

- **Encoder:** recibe la secuencia de entrada y genera representaciones contextuales enriquecidas para cada token.

- **Decoder:** utiliza esas representaciones para producir la secuencia de salida, token por token, como en tareas de generación o traducción.

Ambos están formados por múltiples capas que comparten tres elementos fundamentales:

1. **Multi-Head Attention:** varias atenciones funcionando en paralelo, permitiendo al modelo captar diferentes tipos de relaciones al mismo tiempo.
2. **Add & Norm:** conexiones residuales más normalización, que estabilizan el flujo de información y mejoran el entrenamiento.
3. **Feed-Forward Network:** una pequeña red neuronal que transforma cada token individualmente para refinar sus características internas.

En el decoder aparece además una variante llamada masked multi-head attention, que impide que el modelo “vea” posiciones futuras de la secuencia que aún no deben predecirse.

3.3.4. Proyección final y generación de la salida

Una vez procesada la información, el decoder produce una representación que debe convertirse en una palabra del vocabulario. Para esto se utiliza una capa lineal, encargada de mapear ese vector a un conjunto de valores (logits) para cada token posible.

Finalmente, la función softmax transforma esos logits en probabilidades, indicando cuál es la palabra más probable para continuar la secuencia. Este proceso se repite iterativamente hasta completar la generación de la salida.

La combinación de autoatención, codificación posicional y bloques altamente paralelizables convirtió a los Transformers en la arquitectura dominante para el procesamiento del lenguaje natural. Su capacidad para capturar relaciones complejas, junto con su eficiencia y escalabilidad, ha permitido el desarrollo de los modelos de lenguaje de gran escala (LLMs) utilizados hoy en día. En el contexto del presente proyecto, comprender este funcionamiento es esencial, ya que constituye la base técnica que permite a los modelos interpretar consultas y generar respuestas coherentes a partir de los datos cargados.

3.4 Tokens, embeddings y representación del lenguaje

Para que un modelo de lenguaje pueda procesar texto de manera efectiva, es necesario transformar las palabras en una representación que resulte comprensible para la computadora. Este proceso constituye uno de los pilares conceptuales del funcionamiento de los modelos modernos y se basa en tres elementos fundamentales: los tokens, los embeddings y las representaciones vectoriales del lenguaje. Comprender estos conceptos resulta esencial para interpretar cómo una inteligencia artificial puede analizar, generar o razonar sobre texto humano.

El primer paso en el procesamiento del lenguaje es la tokenización. Un token es una unidad mínima de texto que el modelo utiliza para trabajar. Dependiendo del modelo y del método de tokenización, un token puede ser una palabra completa, una sílaba, un fragmento de palabra o incluso un carácter individual. Esta segmentación no es arbitraria ya que busca equilibrar eficiencia y flexibilidad, permitiendo que el sistema maneje vocabularios extensos y palabras desconocidas. Por ejemplo, términos poco frecuentes o nombres propios pueden dividirse en varios tokens, lo que evita que el modelo “desconozca” completamente una palabra nueva.

Una vez tokenizado el texto, cada token se convierte en un vector numérico mediante un embedding. Un embedding es una representación matemática que captura el significado de una palabra en función de su relación con otras palabras dentro del lenguaje. Esto significa que palabras que suelen aparecer en contextos similares tendrán vectores cercanos entre sí en el espacio multidimensional donde el modelo organiza las representaciones. De esta manera, el sistema no solo identifica palabras de forma aislada, sino que también comprende similitudes semánticas, aspectos sintácticos y relaciones conceptuales. En la Figura 3.4 se observa el esquema del proceso de embeddings, donde imágenes, documentos y audio se transforman en vectores numéricos que luego se representan como puntos relacionados dentro de un espacio tridimensional.

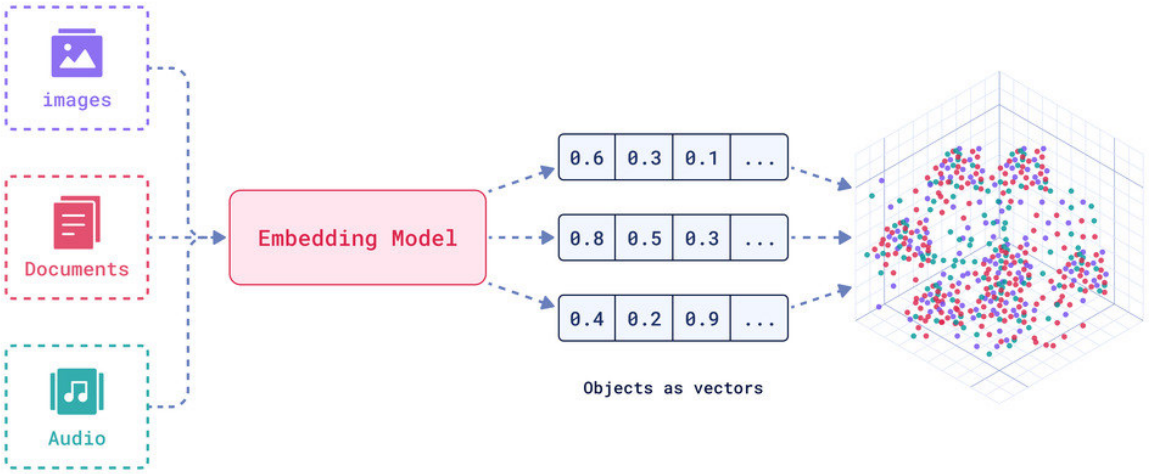


Figura 3.4. Esquema del proceso de embeddings

Este enfoque vectorial fue uno de los grandes avances de la inteligencia artificial moderna, ya que permitió a los modelos manejar el lenguaje con una profundidad mucho mayor que los métodos tradicionales basados en reglas o listas de palabras. Al codificar el significado en un espacio continuo, los embeddings hacen posible que el modelo reconozca que “maestro” y “profesor” son conceptos cercanos, o que “árbol” se relaciona más con “bosque” que con “computadora”. Estas relaciones emergen de los patrones estadísticos que el modelo aprende durante su entrenamiento sobre grandes volúmenes de texto.

A medida que el modelo procesa los tokens, estas representaciones iniciales se enriquecen a través de múltiples capas del Transformer, donde interactúan con los mecanismos de atención. En cada capa, los vectores se ajustan teniendo en cuenta el contexto global de la oración o del documento completo. Este proceso de refinamiento progresivo permite al modelo comprender el significado individual de las palabras, como también su función dentro de la oración, su relación con elementos distantes y la intención general del texto.

El resultado final es una representación contextualizada del lenguaje, una forma de codificar la información que combina el significado semántico, el uso sintáctico y el contexto específico en el que las palabras aparecen. Esta representación es la que permite que los modelos de lenguaje realicen tareas avanzadas como responder preguntas, generar explicaciones, realizar inferencias o convertir instrucciones en acciones, tal como ocurre en el presente proyecto cuando el sistema interpreta una

consulta en lenguaje natural y determina qué tipo de análisis o visualización debe generar.

3.5 Entrenamiento y funcionamiento interno de un LLM

El funcionamiento de un modelo de lenguaje de gran escala (LLM) es el resultado de un proceso de aprendizaje complejo que combina grandes volúmenes de datos, arquitecturas avanzadas como los Transformers y técnicas de optimización desarrolladas en la última década. Comprender cómo se entrenan y operan estos modelos permite dimensionar por qué son capaces de generar texto coherente, interpretar instrucciones en lenguaje natural y resolver tareas que antes estaban limitadas a especialistas humanos.

El entrenamiento de un LLM se realiza generalmente en dos etapas:

1. Preentrenamiento (pre-training).
2. Ajuste fino (fine-tuning).

En la primera etapa, el modelo se expone a cantidades masivas de texto, usualmente provenientes de fuentes diversas como libros digitalizados, artículos, repositorios académicos y páginas web. Durante este proceso, el objetivo principal es aprender a predecir la palabra siguiente dentro de una secuencia. Aunque esta tarea puede parecer simple, al repetirse millones o miles de millones de veces, el modelo adquiere una comprensión profunda de patrones gramaticales, asociaciones semánticas y estructuras discursivas presentes en el lenguaje humano.

Este aprendizaje no se basa en conocimiento explícitamente programado, sino en la detección estadística de regularidades. El modelo ajusta sus parámetros para minimizar el error en la predicción del siguiente token. Cuantos más parámetros posea el modelo, mayor será su capacidad para representar relaciones complejas, aunque esto también implica mayores requerimientos computacionales y de memoria. De esta forma, modelos como GPT, LLaMA o Gemini alcanzaron escalas que superan los miles de millones de parámetros, lo que les permite capturar asociaciones sutiles y desarrollar habilidades emergentes que no se encontraban en modelos más pequeños.

La segunda etapa, el ajuste fino, consiste en especializar el modelo para tareas específicas. En este punto, se entrena con conjuntos de datos más pequeños y cuidadosamente curados, orientados a mejorar su rendimiento en actividades como

seguir instrucciones, responder preguntas, generar código o mantener diálogos más naturales. En muchos casos, esta etapa incluye técnicas como Reinforcement Learning from Human Feedback (RLHF), donde evaluadores humanos califican o comparan respuestas del modelo para guiar su comportamiento hacia resultados más útiles, seguros y alineados con expectativas humanas.

Durante la inferencia, es decir, cuando el modelo responde a una consulta del usuario, no se modifica ningún parámetro. En su lugar, el LLM utiliza las representaciones aprendidas para generar una respuesta token por token, prediciendo en cada paso cuál debería ser la siguiente unidad lingüística según el contexto previo. Aunque el proceso interno involucra operaciones matemáticas altamente complejas, desde la perspectiva del usuario se percibe como un diálogo fluido donde el modelo interpreta instrucciones, extrae información relevante y produce contenido coherente.

El funcionamiento interno del modelo también se apoya en la capacidad del Transformer para considerar todo el contexto simultáneamente mediante el mecanismo de atención. Esto le permite identificar qué partes de la instrucción del usuario son más relevantes para responder correctamente. En la práctica, esta habilidad es esencial para casos como el del presente proyecto, donde el LLM debe analizar preguntas relacionadas con datos cargados, distinguir si se requiere un análisis numérico o la generación de un gráfico, y decidir cómo estructurar la respuesta para que sea comprensible.

A pesar de sus capacidades, los LLMs presentan limitaciones importantes. Su conocimiento está condicionado por los datos en los que fueron entrenados, pueden generar errores o “alucinaciones” cuando no encuentran información suficiente, y no poseen comprensión profunda en un sentido humano. Sin embargo, combinados con herramientas externas y mecanismos de validación, pueden alcanzar niveles de confiabilidad adecuados para aplicaciones académicas y productivas.

3.6 Inteligencia Artificial Generativa en el análisis de datos

La inteligencia artificial generativa ha transformado de manera significativa la forma en que los usuarios interactúan con los datos y extraen información relevante sin necesidad de poseer conocimientos técnicos avanzados. Tradicionalmente, el análisis de datos requería el dominio de herramientas especializadas, lenguajes de programación, conocimientos estadísticos y experiencia en la construcción de visualizaciones. Este conjunto de habilidades, aunque fundamental en disciplinas

técnicas, representaba una barrera para usuarios no expertos o para situaciones donde se necesitaba obtener respuestas de forma rápida y eficiente.

Con la aparición de los modelos de lenguaje de gran escala y su capacidad generativa, surgió un nuevo paradigma en el análisis de datos: el análisis impulsado por lenguaje natural. Bajo este enfoque, los usuarios pueden interactuar con un sistema mediante preguntas o instrucciones expresadas en lenguaje cotidiano, mientras que el modelo se encarga de interpretar esas consultas, comprender la intención detrás de ellas y traducirlas en acciones concretas, como realizar cálculos, limpiar información, generar visualizaciones o resumir tendencias.

Una de las características más relevantes de la inteligencia artificial generativa aplicada al análisis de datos es su habilidad para disminuir la complejidad técnica que implica manipular conjuntos de datos. Gracias a los mecanismos de comprensión contextual y razonamiento lingüístico de los LLMs, es posible abstraer operaciones que anteriormente requerían conocimiento especializado. Por ejemplo, acciones como calcular promedios, identificar valores atípicos, comparar distribuciones o generar histogramas pueden solicitarse simplemente con una instrucción escrita, sin necesidad de código explícito o manipulación manual del dataset.

Este enfoque no solo democratiza el acceso al análisis de datos, sino que también agiliza los procesos de exploración y descubrimiento. En contextos académicos, empresariales o profesionales, permite obtener conclusiones preliminares con rapidez, generar informes automatizados y reducir tiempos de trabajo. Además, los modelos generativos pueden ofrecer explicaciones en lenguaje natural que complementan los resultados numéricos o gráficos, facilitando la interpretación y aumentando la utilidad del análisis.

La integración de estos modelos con herramientas externas, como librerías estadísticas, motores gráficos o intérpretes de código, amplifica aún más sus capacidades. En este tipo de sistemas híbridos, el LLM actúa como un agente que decide qué tipo de operación realizar y cómo construir la respuesta más clara para el usuario. Esta arquitectura mixta es clave en proyectos como el presente, donde el modelo debe evaluar si una consulta requiere generar un gráfico, realizar operaciones matemáticas o simplemente explicar el comportamiento de una variable.

Por otro lado, la inteligencia artificial generativa aplicada al análisis de datos también presenta desafíos. Los modelos pueden cometer errores al interpretar datos ambiguos, presentar alucinaciones en los resultados o generar conclusiones sin una

verificación estadística estricta. Por ello, la combinación entre el LLM y herramientas estructuradas resulta fundamental para garantizar precisión y confiabilidad. Por esto, la combinación entre LLM y herramientas estructuradas resulta fundamental para garantizar precisión y confiabilidad en sistemas de análisis automatizado.

En este contexto, la inteligencia artificial generativa se posiciona como un recurso valioso para modernizar y mejorar los procesos de análisis exploratorio de datos (EDA). Su capacidad para comprender instrucciones, generar visualizaciones, proponer interpretaciones y operar como un intermediario entre el usuario y el dataset la convierte en un elemento central de herramientas contemporáneas como EDAI, desarrollada en el marco del presente trabajo. Con ella, el análisis de datos deja de ser una tarea reservada exclusivamente a perfiles técnicos y se convierte en una actividad accesible, intuitiva y más cercana al usuario.

3.7 Procesamiento de Lenguaje Natural (PLN)

El Procesamiento de Lenguaje Natural (PLN) o Natural language processing (NLP) en inglés, es una rama de la inteligencia artificial que estudia los métodos y algoritmos que permiten a una máquina comprender, interpretar y generar lenguaje humano de manera significativa. Este campo integra conocimientos de lingüística computacional, aprendizaje automático y modelado estadístico, con el objetivo de reducir la brecha entre la comunicación humana y la capacidad de los sistemas computacionales para operar sobre el lenguaje.

A diferencia de otros tipos de datos estructurados, el lenguaje natural presenta ambigüedades, variaciones gramaticales y matices contextuales que dificultan su interpretación automática. Por este motivo, el PLN se apoya en representaciones matemáticas del texto, técnicas de análisis gramatical y modelos entrenados sobre grandes volúmenes de datos, los cuales permiten identificar patrones y relaciones semánticas dentro del discurso.

Dentro del PLN, se destacan varias tareas fundamentales que conforman el flujo típico de procesamiento del texto:

- Tokenización: consiste en dividir el texto en unidades mínimas (tokens), que pueden ser palabras, subpalabras o caracteres. Es un paso esencial para que el sistema pueda operar computacionalmente sobre el lenguaje.

- Etiquetado gramatical (POS-tagging): asigna categorías gramaticales a cada token (sustantivo, verbo, adjetivo, etc.), permitiendo analizar la estructura de las oraciones.
- Análisis sintáctico: identifica la estructura jerárquica de una oración y las relaciones entre sus componentes, lo cual permite detectar sujeto, predicado, modificadores, dependencias y relaciones lógicas.
- Análisis semántico: busca interpretar el significado de las palabras y oraciones dentro de un contexto dado, abordando desafíos como la desambiguación semántica o la resolución de referencias.
- Extracción de información: identifica entidades relevantes, relaciones, valores numéricos o patrones específicos dentro de un texto.
- Generación de lenguaje natural (NLG): produce oraciones coherentes y significativas en lenguaje natural a partir de datos estructurados o representaciones internas.

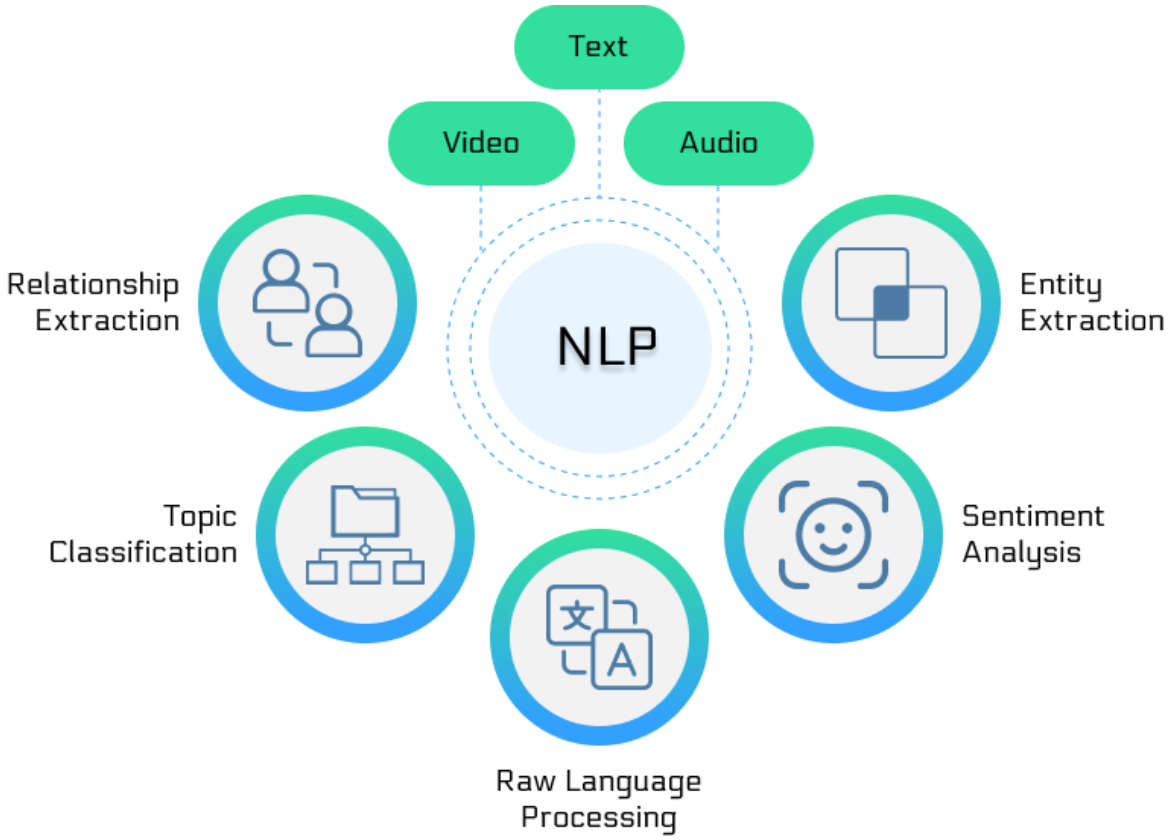


Figura 3.7. Esquema del Procesamiento de Lenguaje Natural, donde entradas como texto, audio y video son analizadas para convertir datos crudos en información interpretable.

Con la llegada de los modelos de lenguaje de gran escala, muchas de estas tareas dejaron de implementarse por separado para integrarse en un único modelo capaz

de realizar varios procesos simultáneamente. Los embeddings contextuales, combinados con mecanismos de atención, permiten que un LLM analice de manera conjunta la estructura textual, el contexto semántico y la intención comunicativa.

En el marco del presente proyecto, el PLN desempeña un rol central, ya que permite que los usuarios interactúen con el sistema utilizando lenguaje natural. El modelo debe interpretar correctamente la intención detrás de cada consulta, determinar si se trata de una solicitud de análisis, de generación de un gráfico o de una duda conceptual, y traducir esa intención en acciones computacionales concretas. Este proceso implica transformar expresiones verbales en instrucciones estructuradas, seleccionar el dataset adecuado, generar código para el análisis y producir una explicación comprensible.

De este modo, el PLN actúa como un puente entre la comunicación humana y las operaciones técnicas que ejecuta el sistema. Sin este nivel de comprensión lingüística, la interacción basada únicamente en lenguaje natural no sería posible, ni sería viable automatizar el análisis de datos sin requerir conocimientos técnicos avanzados por parte del usuario.

3.8 Sistemas de agentes inteligentes

Los sistemas de agentes inteligentes constituyen un enfoque dentro de la inteligencia artificial orientado a diseñar entidades capaces de percibir su entorno, razonar sobre la información disponible y actuar de manera autónoma para alcanzar un objetivo. Un agente inteligente se define, en términos generales, como una unidad computacional que procesa información, toma decisiones y ejecuta acciones basadas en reglas, modelos, políticas o aprendizajes previos. La capacidad de razonar, planificar o seleccionar comportamientos apropiados convierte a los agentes en elementos fundamentales dentro de sistemas complejos.

Cuando múltiples agentes interactúan entre sí, se conforma lo que se denomina un sistema multiagente. En este tipo de sistemas, cada agente posee habilidades, roles o responsabilidades específicas, y su cooperación permite resolver problemas que serían demasiado amplios, costosos o complejos para abordarlos desde un único módulo centralizado. La interacción entre agentes puede incluir el intercambio de información, la toma de decisiones coordinadas o la delegación de subtareas, lo que habilita estructuras distribuidas, flexibles y escalables.

En el contexto contemporáneo de la inteligencia artificial, los modelos de lenguaje de gran escala han ampliado la definición clásica de agente inteligente. Un LLM ya no solo procesa texto, sino que puede razonar, planificar pasos, interpretar instrucciones, generar código, interactuar con herramientas externas e incluso decidir qué acción ejecutar en función del estado del entorno. Esta capacidad permite considerar a un modelo de lenguaje como un agente cognitivo, es decir, como una entidad con habilidades de percepción lingüística, razonamiento contextual y acción computacional.

Dentro de estos sistemas aparece un concepto clave: las herramientas (tools). Las herramientas son funciones o ejecuciones externas a las que un agente puede acceder para ampliar sus capacidades. Por ejemplo, una herramienta puede permitir ejecutar código Python, consultar una base de datos, leer un documento, generar un gráfico o realizar cálculos específicos. Desde la perspectiva del agente, una tool representa una habilidad adicional que puede decidir utilizar cuando lo requiere la resolución de un problema. Esto transforma al agente en un sistema con capacidades ampliadas y adaptables, capaz de actuar sobre el mundo digital más allá de la mera generación de texto.

En un sistema multiagente, la toma de decisiones implica identificar cuál de los agentes o herramientas disponibles es el más adecuado para cada etapa del proceso. Este mecanismo de selección puede basarse en políticas internas, razonamiento implícito del modelo o estrategias definidas por el diseñador del sistema. De esta manera, los agentes cooperan a través de estados compartidos, se transfieren información relevante y ejecutan sus respectivas funciones contribuyendo al cumplimiento de un objetivo común.

Esta arquitectura multiagente resulta especialmente relevante en sistemas de análisis de datos automatizado, donde la división del trabajo entre agentes especializados mejora la robustez, precisión y eficiencia del proceso analítico.

3.9 Orquestación de flujos en sistemas conversacionales

La interacción con modelos de lenguaje avanzados suele percibirse como un proceso lineal en el que el usuario formula una consulta y el sistema responde. Sin embargo, cuando estas tecnologías se emplean para resolver tareas complejas, como analizar datos, ejecutar herramientas externas, coordinar agentes o mantener coherencia entre múltiples pasos, es necesario organizar el proceso mediante estructuras más robustas que garanticen orden, consistencia y continuidad lógica.

La orquestación se refiere al conjunto de mecanismos que permiten definir, controlar y coordinar las acciones que realiza un sistema inteligente desde el inicio de una interacción hasta su finalización. En otras palabras, implica establecer un “plan” de cómo debe comportarse el sistema ante distintos escenarios, qué pasos deben ejecutarse, en qué orden y bajo qué condiciones. Sin un proceso de orquestación, los modelos de lenguaje tenderían a generar respuestas desordenadas, inconsistentes o incompletas cuando enfrentan tareas que requieren múltiples pasos o dependencias entre acciones.

En este contexto surge el concepto de flujo conversacional, entendido como la secuencia estructurada de etapas que sigue el sistema para interpretar la consulta del usuario, tomar decisiones intermedias y producir un resultado coherente. Un flujo conversacional bien diseñado considera los posibles caminos que puede adoptar la interacción, las validaciones necesarias, las herramientas disponibles y los agentes involucrados. Este tipo de enfoques permite que el sistema mantenga un hilo lógico a través del diálogo, evitando repeticiones o contradicciones.

Dado que los modelos de lenguaje no poseen memoria estructurada por sí mismos y suelen operar de manera puramente generativa, requieren mecanismos externos que organicen su comportamiento. La orquestación agrega esta capa de estructura, definendo qué partes del proceso dependen del modelo y cuáles dependen de decisiones programáticas o arquitectónicas. Esto es especialmente importante en tareas de análisis de datos, donde el sistema debe alternar entre razonamiento lingüístico, ejecución de código, acceso a datos y generación de visualizaciones.

Un enfoque teórico relevante para este tipo de procesos es el de los flujos basados en nodos. En este paradigma, cada etapa del razonamiento o de la tarea se representa como un nodo independiente con una función específica: procesar texto, validar el estado, ejecutar una herramienta, seleccionar un agente, etc. Los nodos se conectan entre sí mediante transiciones que indican qué paso sigue según el resultado del anterior. Esto permite modelar comportamientos condicionales, bifurcaciones lógicas y ciclos controlados, generando una estructura comprensible y extensible para flujos complejos.

En el marco del presente proyecto, esta visión teórica es fundamental para comprender por qué se requiere una herramienta de orquestación como LangGraph y cómo esta contribuye a organizar la secuencia completa de procesamiento del sistema. La orquestación no solo estructura el comportamiento del modelo de lenguaje, sino que también posibilita la interacción coherente entre los agentes inteligentes, la administración de herramientas externas y la gestión del estado conversacional a lo largo de la sesión.

4. Bases Tecnológicas

4.1 Modelos de lenguaje de gran escala

En el presente proyecto, los LLMs constituyen el eje central del sistema, ya que son los encargados de interpretar las consultas en lenguaje natural, determinar la acción más adecuada según el tipo de requerimiento y producir respuestas acompañadas de explicaciones comprensibles para el usuario.

Para lograr esto, se implementó una arquitectura donde el backend desarrollado en Python (mediante FastAPI) actúa como intermediario entre el frontend y las APIs externas que ofrecen los modelos de lenguaje.

4.2 Proceso de selección de modelos

Durante la fase de desarrollo se realizó una evaluación comparativa de cinco modelos LLM diferentes:

- Llama 3.1 – 8B Instant
- Qwen 3 – 32B
- OpenAI GPT-OSS – 20B
- OpenAI GPT-OSS – 120B
- Gemini 2.0 Flash

Los criterios de evaluación incluyeron: uso correcto de herramientas especializadas (tools), calidad y coherencia de respuestas textuales, precisión en generación de gráficos, capacidad de selección entre estrategias SQL y DataFrame, costo operativo, velocidad de respuesta y estabilidad técnica. Los resultados detallados de esta comparación se presentan en la sección 8.3 Análisis comparativo de modelos LLM.

4.3 Modelos seleccionados e integración como agentes

Tras el análisis comparativo, se adoptó una arquitectura híbrida que combina dos modelos complementarios operando como agentes especializados:

Agente de comprensión semántica - OpenAI GPT-OSS-120B (implementado a través de Groq): Empleado específicamente para la generación de descripciones semánticas de los documentos cargados al sistema. Este agente actúa como filtro inicial, permitiendo identificar y seleccionar automáticamente el dataset más relevante frente a una consulta del usuario mediante matching semántico. Su fortaleza radica en la calidad de las representaciones semánticas y coherencia contextual, ofreciendo excelente precisión en tareas analíticas estructuradas. Está disponible gratuitamente a través de Groq (200K tokens por minuto).

Agente ejecutor y de razonamiento - Gemini 2.0 Flash (Google): Seleccionado como modelo principal para el procesamiento de consultas, interpretación de lenguaje natural, selección de herramientas de análisis y generación de código para visualizaciones. Este agente procesa la consulta del usuario, determina la estrategia de resolución más adecuada, decide si debe ejecutar una herramienta específica o generar código, y tiene acceso directo a un conjunto de tools especializadas para análisis de datos y creación de gráficos. Destaca por su rapidez en el procesamiento (latencia baja), uso correcto de herramientas especializadas en todos los casos de prueba, excelente manejo de estrategias SQL y DataFrame, y su amplia ventana de contexto (1 millón de tokens), con un costo operativo accesible ($0.10–$0.40 por millón de tokens tras límite gratuito).

Flujo de cooperación entre agentes:

La arquitectura multiagente permite cubrir el ciclo completo de análisis mediante la siguiente secuencia de operaciones:

1. El usuario envía una consulta en lenguaje natural
2. GPT-OSS-120B analiza semánticamente la consulta e identifica el dataset más relevante entre los documentos disponibles
3. Gemini 2.0 Flash recibe el contexto (consulta + dataset seleccionado) y ejecuta el análisis solicitado mediante sus herramientas especializadas
4. Gemini genera la visualización correspondiente y produce una interpretación textual de los resultados

Esta división de responsabilidades aprovecha las fortalezas específicas de cada modelo: GPT-OSS-120B para la comprensión semántica profunda en la fase de selección, y Gemini para la ejecución ágil y precisa del flujo analítico con acceso directo a herramientas de visualización y análisis.

4.4 Herramientas y frameworks utilizados

El desarrollo del sistema se sustenta en una arquitectura modular y distribuida, compuesta por un frontend orientado a la interacción con el usuario, un backend encargado del procesamiento de las consultas y la comunicación con los modelos de lenguaje, y una capa intermedia de integración basada en agentes inteligentes. La selección de las herramientas y frameworks utilizados responde tanto a criterios técnicos (rendimiento, escalabilidad y facilidad de integración) como a criterios académicos, al priorizar tecnologías ampliamente adoptadas en entornos de investigación y desarrollo.

En el backend, se empleó Python como lenguaje principal debido a su versatilidad y a la extensa disponibilidad de bibliotecas para inteligencia artificial y análisis de datos. Sobre esta base, se utilizó el framework FastAPI, que permite la construcción

de servicios REST ligeros y eficientes, facilitando la comunicación entre el modelo de lenguaje, los módulos de análisis y el frontend.

Además, se incorporó LangChain, un framework ampliamente utilizado para la construcción de aplicaciones basadas en modelos de lenguaje. LangChain proporciona un conjunto de abstracciones que simplifican la integración entre LLMs, herramientas externas, bases de datos y flujos de procesamiento. A diferencia de un uso directo del modelo, LangChain introduce componentes como chains, prompts, retrievers y tools, que permiten estructurar tareas complejas de manera modular y reutilizable. En términos de funcionamiento, LangChain sigue un esquema lineal de ejecución, en el que la entrada atraviesa una secuencia definida de pasos hasta obtener un resultado. Este comportamiento puede observarse en la Figura 4.4, donde se representa el flujo básico utilizado por LangChain.

Dentro de este proyecto, LangChain cumple un rol fundamental como capa de integración, ya que define la lógica de las herramientas empleadas por los agentes. Las tools desarrolladas permiten que el modelo realice acciones específicas, tales como analizar datos, resumir documentos o transformar información. Estas herramientas funcionan como “bloques operativos” que posteriormente son gestionados mediante un nivel superior de orquestación.

Ese nivel de orquestación está a cargo de LangGraph, una biblioteca especializada, desarrollada dentro del ecosistema de LangChain, que permite estructurar flujos conversacionales complejos mediante una arquitectura basada en grafos dirigidos. A diferencia del flujo lineal característico de LangChain, LangGraph permite construir rutas dinámicas compuestas por distintos nodos, cada uno asociado a una etapa específica del proceso, como análisis, validación o generación de respuesta. Tal como se ilustra en la Figura 4.4, LangGraph posibilita que una misma entrada pueda recorrer distintos caminos según el contexto, dando lugar a flujos que incluyen secuencias alternativas de nodos y etapas.

Cada nodo del grafo ejecuta una función puntual y comparte un estado compartido que se transmite a lo largo del flujo completo, asegurando que la información producida en una etapa esté disponible para las siguientes. Esta capacidad resulta esencial para mantener coherencia contextual en sistemas multiagente. Además, LangGraph incorpora mecanismos de enrutamiento condicional, que permiten decidir dinámicamente qué nodo debe ejecutarse a continuación en función de los resultados intermedios o del tipo de consulta del usuario, habilitando comportamientos adaptativos y flexibles.

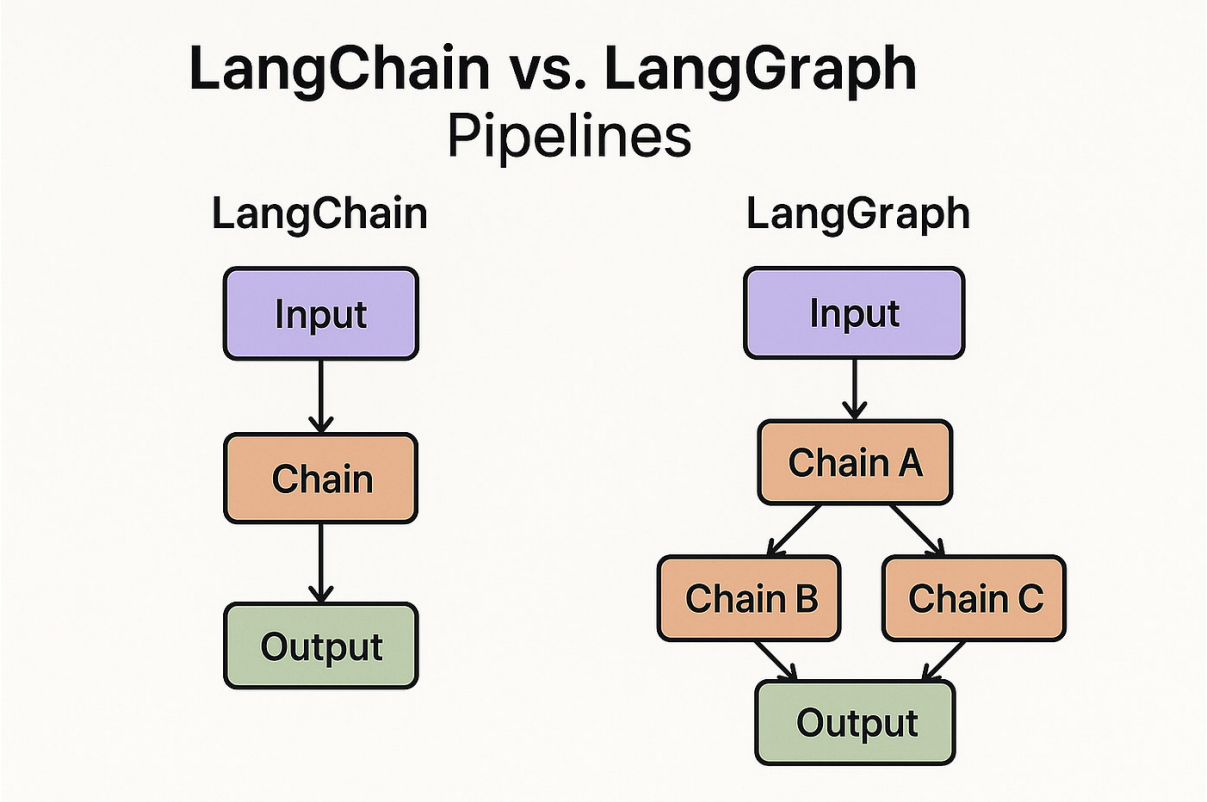


Figura 4.4 Flujo de ejecución de LangChain vs LangGraph

Adicionalmente, mediante el uso de checkpointers como PostgresSaver, LangGraph permite persistir el estado completo del grafo en cada paso de ejecución, garantizando la recuperación del contexto conversacional ante interrupciones y posibilitando el análisis posterior de la trazabilidad de las decisiones tomadas por el sistema.
Como se comentó anteriormente, LangGraph actúa como el orquestador central que coordina la interacción entre los modelos de lenguaje, las herramientas de análisis de datos y los mecanismos de validación, asegurando que cada componente del sistema colabore de manera eficiente y manteniendo la coherencia del flujo analítico completo.
El diseño e implementación específica del grafo de nodos del sistema se detalla en la sección 6.2 Diseño del flujo conversacional y uso de LangGraph

El frontend fue desarrollado con React y Next.js, los cuales son frameworks modernos que facilitan la creación de interfaces reactivas y dinámicas, ideales para la visualización de resultados generados por el modelo. Se complementó con Tailwind CSS y shadcn/ui, que son frameworks que ofrecen una base sólida para el diseño visual, ofreciendo una experiencia de usuario limpia, accesible y coherente con los estándares actuales del desarrollo web.

Como sistema de base de datos se consideró PostgreSQL, empleado para gestionar los checkpoints de conversación mediante el uso de PostgresSaver, lo que permite mantener el contexto histórico de las interacciones entre el usuario y el modelo. Esta característica resulta esencial para ofrecer respuestas más precisas y contextualizadas a lo largo del diálogo.

Por último, se utilizaron las APIs de Gemini y Groq para la conexión con los modelos de lenguaje, permitiendo combinar el poder de procesamiento de los modelos "Gemini 2.0 Flash" que se emplea para la interpretación de consultas y ejecución de tools. Por otro lado, se utiliza el modelo "OpenAI GPT-OSS-120B" (ofrecido por Groq) para la generación de descripciones semánticas de los documentos. Esta integración otorga flexibilidad al sistema, al permitir seleccionar el modelo más adecuado según la naturaleza de la consulta o la necesidad de análisis.

En conjunto, estas herramientas conforman una infraestructura escalable, moderna y adaptable, que responde a los requerimientos técnicos del proyecto, y también ofrece una base sólida para futuras investigaciones académicas relacionadas con la aplicación de modelos de lenguaje y agentes inteligentes en el análisis automatizado de datos.

4.5 Técnicas de análisis y visualización de datos

El sistema implementa un proceso automatizado de análisis y visualización de datos basado en el procesamiento de lenguaje natural. A partir de ello, el sistema accede al dataset correspondiente, extrae la información solicitada y produce una representación visual acompañada de una explicación textual que describe los resultados obtenidos.

Para la generación de visualizaciones, el proyecto emplea bibliotecas consolidadas en el ecosistema de Python, como Matplotlib, Seaborn y Pandas, las cuales permiten construir gráficos de barras, dispersión, histogramas, boxplots, mapas de calor, series temporales y gráficos de pastel, entre otros. Estas herramientas garantizan una presentación clara y adaptable a distintos tipos de datos, al mismo tiempo que ofrecen un entorno flexible para la creación dinámica de visualizaciones generadas por el propio modelo.

El proceso es completamente automático, sin necesidad de que el usuario seleccione parámetros o filtros manualmente. La interacción en lenguaje natural simplifica la experiencia de análisis, permitiendo que el modelo genere el código necesario para construir los gráficos y explicar su contenido de manera contextualizada. De esta forma, el sistema combina análisis cuantitativo, visualización informativa e interpretación semántica, integrando la potencia del aprendizaje automático con la comunicación humana.

5. Diseño e Implementación del Sistema

5.1 Arquitectura general del sistema

El sistema presenta una arquitectura cliente-servidor clásica, en la Figura 6.1 podemos observar la arquitectura del sistema organizada en tres capas principales:

- Presentación (frontend).
- Lógica de negocio (backend).
- Almacenamiento (base de datos).

Esta estructura ofrece múltiples ventajas como la separación de responsabilidades, el mantenimiento del código y la escalabilidad del proyecto.

En la capa de presentación, desarrollada con Next.js y React, el usuario puede interactuar con la aplicación a través de una interfaz de chat, carga de documentos y visualización de gráficos generados automáticamente. Cada acción realizada en esta interfaz se comunica con el backend mediante solicitudes HTTP, lo que permite enviar archivos, realizar consultas o recibir resultados analíticos.

El backend, implementado con FastAPI, centraliza la lógica del sistema. Dentro de él se integra la biblioteca LangGraph, que actúa como un orquestador de agentes inteligentes y permite gestionar el flujo de análisis entre distintos modelos. Este módulo coordina la comunicación con dos servicios externos:

- El servicio LLM de Groq (GPT-OSS-120B).
- El servicio LLM de Gemini 2.0 Flash.

La capa de almacenamiento, utiliza una base de datos PostgreSQL, responsable de conservar los documentos transformados en tablas, las descripciones semánticas generadas y los checkpoints del flujo LangGraph, lo que permite mantener el estado de las conversaciones y consultas.

El flujo de información es bidireccional entre todas las capas, es decir que el frontend envía peticiones al backend, este las procesa y puede apoyarse en los modelos de LLM para generar respuestas que, finalmente, son devueltas al usuario en forma de texto o gráficos interpretativos.

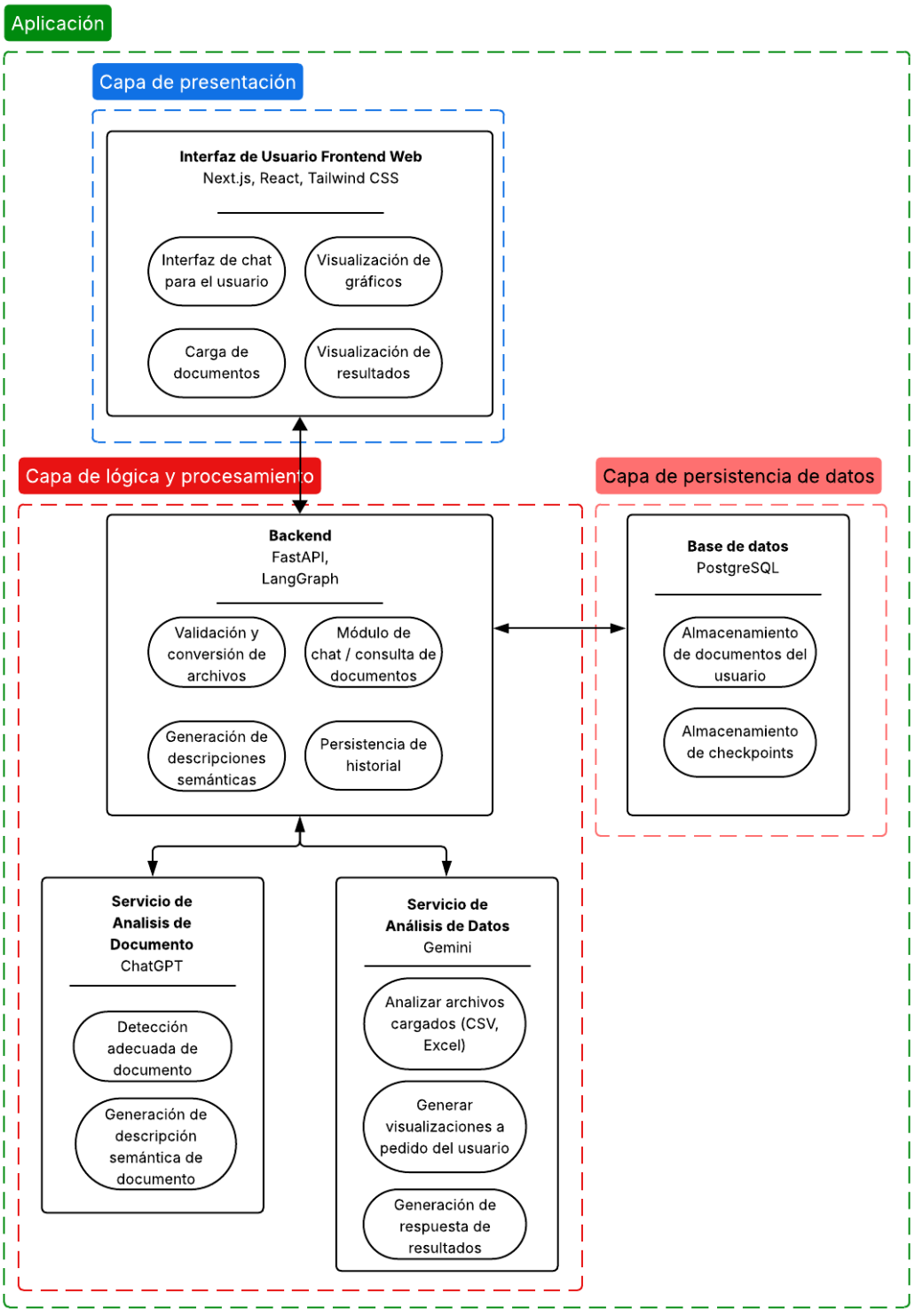


Figura 6.1. Arquitectura general del sistema

5.2 Diseño del flujo conversacional y uso de LangGraph

El flujo conversacional del sistema se estructura mediante LangGraph (descrito en la sección 5.4 Herramientas y frameworks utilizados), implementando un grafo de seis nodos especializados que procesan las consultas del usuario de forma secuencial y adaptativa.
La interacción comienza cuando el usuario envía una consulta desde el frontend. Esta solicitud activa el grafo en el backend, iniciando un proceso que atraviesa distintas etapas. Primero se define la estrategia de análisis más adecuada (SQL vs DataFrame), luego se clasifica la tarea específica, se ejecuta el código o consulta correspondiente, se valida el resultado con posibilidad de fallback automático entre estrategias, y finalmente se genera una respuesta interpretativa acompañada de visualizaciones cuando corresponda.
El enrutamiento condicional de LangGraph determina dinámicamente el camino a seguir según el tipo de consulta y los resultados intermedios obtenidos. Por ejemplo, si una consulta SQL falla por limitaciones del motor de base de datos, el sistema ejecuta automáticamente un fallback hacia el modo DataFrame sin intervención del usuario. La persistencia del estado mediante PostgresSaver garantiza la trazabilidad completa de las decisiones y permite mantener el contexto conversacional entre sesiones.

En la Figura 6.2, se muestra el esquema de nodos que participan en el sistema. La arquitectura del grafo se compone de seis nodos especializados que procesan el estado compartido (AgentState) de forma secuencial, con capacidad de enrutamiento condicional según los resultados intermedios. Cada nodo ejecuta una función específica y actualiza el estado antes de transferir el control al siguiente componente del flujo.
El procesamiento sigue una lógica de pipeline adaptativo: primero se determina la estrategia óptima de acceso a datos, luego se clasifica la acción específica requerida, se ejecuta el código o consulta correspondiente (con posibilidad de fallback automático entre métodos), se valida el resultado obtenido, y finalmente se genera una respuesta interpretativa contextualizada. Este diseño garantiza robustez operativa mediante recuperación automática ante errores, trazabilidad completa del flujo de decisiones, y coherencia conversacional a través de la actualización continua de la memoria del sistema.
La siguiente tabla resume las responsabilidades y salidas principales de cada nodo:

Nodo	Responsabilidad Principal	Entrada Clave	Salida Clave	Decisión de Enrutamiento
node_strategy	Determina estrategia de análisis (memory/SQL/DataFrame) y selecciona dataset relevante	Consulta del usuario, documentos disponibles, historial	estrategia (SQL/DataFrame/memory), dataset seleccionado	→ node_classification
node_classification	Clasifica tipo de acción y selecciona herramienta apropiada	Estrategia definida, consulta, metadatos	acción (herramienta elegida), razonamiento del modelo	→ node_sql_executor / node_python_executor / node_response
node_sql_executor	Genera y ejecuta consulta SQL en PostgreSQL	Metadatos de tablas, consulta del usuario	resultado de la consulta SQL (DataFrame), éxito de ejecución (bool)	→ node_validation
node_python_executor	Genera y ejecuta código Python con pandas/matplotlib	DataFrame cargado, historial de errores, consulta	historial de ejecución, gráficos guardados, resultado (éxito o error)	→ node_validation
node_validation	Valida resultados y gestiona fallbacks automáticos (máx 3 intentos)	Estado de ejecución, contador de iteraciones	flag para saber si necesita fallback (bool), contador de iteraciones	→ node_python_executor / node_response / END
node_response	Genera respuesta interpretativa final y actualiza memoria conversacional	Resultados de ejecución, código generado, tipo de análisis	respuesta final (texto + metadata), memoria actualizada	→ END

Características clave del flujo:

- **Fallback automático**: Si SQL falla por limitaciones del motor, el sistema reintenta con DataFrame sin intervención del usuario.
- **Validación iterativa**: Hasta 3 intentos de corrección automática ante errores de código, aprendiendo de fallos previos.
- **Persistencia del estado**: Cada transición se registra mediante PostgresSaver, garantizando recuperación completa del contexto.
- **Enrutamiento condicional**: El flujo se adapta dinámicamente según resultados intermedios, no sigue una secuencia rígida.

Los detalles técnicos de implementación de cada nodo, incluyendo firmas de función, manejo de excepciones y actualización del estado, se documentan en la sección 6.4 Backend: diseño e implementación - src/nodes.py.

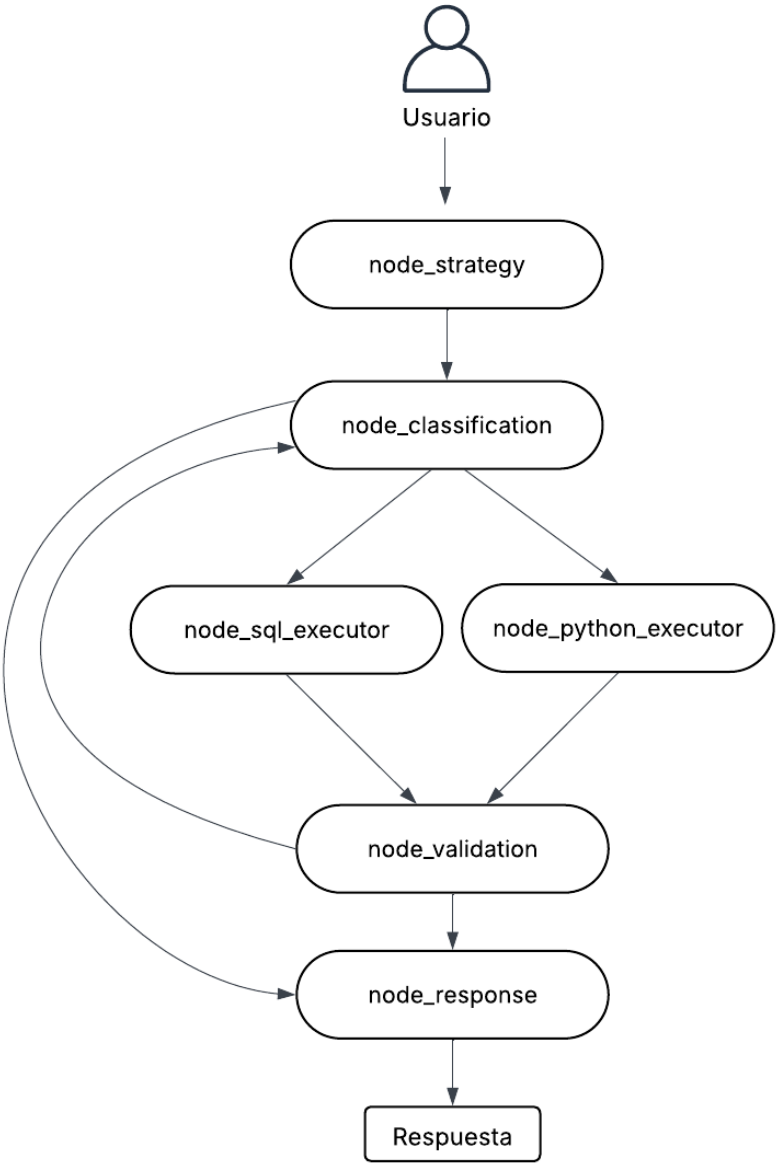


Figura 6.2. Esquema de nodos

5.3 Implementación de la comunicación con LLMs

La implementación de la comunicación con los modelos de lenguaje constituye un componente esencial dentro del sistema, ya que permite la interpretación de las consultas del usuario, la selección de estrategias de análisis y la generación de

respuestas coherentes con el contexto de los datos. Esta capa actúa como el punto de enlace entre la lógica de negocio del backend y los servicios de inteligencia artificial externos, asegurando un intercambio de información controlado, dinámico y trazable.

Conexión con los modelos

El sistema establece la conexión con los modelos mediante sus APIs oficiales: Google Generative AI API para Gemini 2.0 Flash y Groq API para GPT-OSS-120B. Las credenciales de autenticación se gestionan mediante variables de entorno almacenadas en un archivo .env, garantizando la seguridad de las claves de acceso. El backend instancia clientes específicos para cada servicio utilizando las bibliotecas ChatGoogleGenerativeAI y ChatGroq de LangChain. La configuración de cada cliente incluye:

Gemini 2.0 Flash:

```
ChatGoogleGenerativeAI(
    model="gemini-2.0-flash",
    google_api_key=API_KEY,
    temperature=0 # Temperatura baja para respuestas determinísticas
)
```

GPT-OSS-120B:

```
ChatGroq(
    model="openai/gpt-oss-120b",
    api_key=GROQ_KEY,
    temperature=0
)
```

Las invocaciones a los modelos se realizan mediante métodos síncronos que envían el prompt construido dinámicamente y reciben la respuesta procesada en formato texto o estructurado (JSON cuando se requiere). La arquitectura permite cambiar fácilmente de proveedor o modelo mediante modificación de parámetros de configuración, sin alterar la lógica de negocio del sistema.

Gestión y construcción de prompts

A diferencia de los sistemas que utilizan plantillas fijas, los prompts en este proyecto se generan de manera dinámica. Esto significa que su estructura y contenido dependen del estado actual del flujo conversacional y de la estrategia seleccionada por el grafo.
Cada prompt incluye distintos elementos contextuales, como el historial resumido de interacción, la consulta actual del usuario, los metadatos del dataset, las herramientas disponibles y, en algunos casos, las descripciones semánticas de las tablas o documentos analizados.
Por ejemplo, el nodo encargado de seleccionar la herramienta adecuada recibe información contextual que incluye la estrategia de datos (SQL o DataFrame), las herramientas disponibles y la descripción del dataset. A partir de ello, el modelo debe analizar la consulta y devolver el nombre de la herramienta más apropiada, siguiendo instrucciones explícitas incluidas en el prompt.

Este diseño introduce un comportamiento análogo a un prompt chaining implícito, donde cada nodo del grafo construye un nuevo contexto basado en la salida del nodo anterior, lo que favorece una toma de decisiones más informada y coherente dentro del flujo general de LangGraph.

Evaluación de respuestas y validación del flujo

El sistema incorpora un mecanismo de validación mediante el nodo denominado nodo de validación, encargado de evaluar el éxito o fallo de la respuesta obtenida y decidir el siguiente paso en el flujo conversacional.
Este nodo determina si la ejecución fue exitosa, si se requiere un cambio de estrategia, o si se alcanzó el número máximo de iteraciones permitidas. En función de estos parámetros, actualiza el estado y redirige la ejecución hacia el nodo correspondiente.

De este modo, el proceso de evaluación verifica la validez de la respuesta del modelo, como también garantiza la continuidad del flujo, gestionando los posibles fallos de forma estructurada y manteniendo trazabilidad de las decisiones mediante el historial del estado.

Manejo de errores y persistencia

En caso de que una solicitud a los modelos falle, por ejemplo, debido al agotamiento de tokens o a una respuesta inválida, el sistema informa del error sin interrumpir la

ejecución general del backend. Aunque actualmente no se implementa un mecanismo de reintento automático, el sistema se mantiene operativo y notifica el problema al usuario, lo que permite continuar con nuevas consultas.

Asimismo, cada interacción (incluyendo prompts, respuestas y decisiones intermedias) se almacena en los checkpoints de LangGraph, donde se conserva un resumen del historial conversacional y del estado del flujo. Esta persistencia garantiza que el sistema pueda retomar o analizar procesos previos sin pérdida de información contextual.

5.4 Backend: diseño e implementación con FastAPI

El backend del sistema fue desarrollado empleando FastAPI, un framework moderno y asíncrono de Python que permite construir APIs RESTful de alto rendimiento. Su elección se debió a su facilidad para integrar dependencias, su compatibilidad con tipado estático y su soporte nativo para documentación automática mediante OpenAPI y Swagger UI.

Arquitectura del backend

La estructura del backend sigue un enfoque modular inspirado en los principios de arquitectura limpia, fomentando la separación de responsabilidades y la mantenibilidad del código. Aunque no se implementa una Clean Architecture estricta, la organización de carpetas y módulos garantiza una clara delimitación entre capas de lógica, control y persistencia de datos.

La estructura principal se ubica dentro del directorio src/, que contiene los siguientes componentes clave:

src/api/: representa la capa de exposición del sistema.

- main.py: instancia la aplicación FastAPI, configura el middleware CORS, y define los *routers* que exponen los endpoints hacia el frontend.
- routes/: contiene los controladores de las rutas (chat.py y documents.py), encargados de recibir las peticiones HTTP del cliente y delegar la lógica a los servicios correspondientes.
- schemas/: define los modelos de datos utilizados para validar las peticiones y respuestas, aplicando las ventajas de tipado estricto de Pydantic.

src/services/: agrupa la **lógica de negocio** del sistema, separada de los controladores. Los archivos chat_service.py y document_service.py gestionan las operaciones principales, es decir, el procesamiento de interacciones con el modelo de lenguaje y la carga, registro y consulta de documentos analizados.

src/checkpoints.py: Este módulo gestiona la persistencia de memoria conversacional usando PostgresSaver de LangGraph. Implementa funciones para configurar y conectar con PostgreSQL, crear las tablas necesarias para checkpoints, y recuperar/guardar el historial de conversaciones del usuario. Incluye manejo robusto de errores con fallbacks alternativos cuando la configuración automática falla, y proporciona un sistema de thread único para el usuario que permite mantener contexto entre sesiones. La función más crítica es *load_conversation_history()* que recupera el historial completo y el contexto del usuario desde la base de datos.

src/config.py: Archivo de configuración centralizada que define constantes globales del sistema. Establece el thread ID único para el usuario, controla si el guardado automático a BD está habilitado, y carga las API keys necesarias (Google Gemini y Groq). También incluye una opción para convertir formatos de fecha incorrectos a decimales para que no se pierdan datos en documentos que sube el usuario.

src/database.py: Módulo fundamental para la gestión de conexiones PostgreSQL y operaciones de base de datos. Se encarga de crear la base de datos si no existe, verificar conexiones, y proporciona funciones para obtener metadatos de tablas. También crea una tabla central para registrar todos los documentos subidos con sus hashes, metadatos y descripciones semánticas.

src/dataset_manager.py: Maneja todo el ciclo de vida de los datasets, es decir la carga, almacenamiento y recuperación desde PostgreSQL. Incluye funciones para crear tablas desde DataFrames con tipos de datos optimizados, limpiar nombres de columnas, insertar datos en batch, y generar descripciones semánticas usando LLM.

src/graph.py: Define el grafo de ejecución principal del agente usando LangGraph. Conecta todos los nodos (estrategia, clasificación, ejecutores SQL/Python, validación y respuesta) con lógica de routing condicional. Implementa el flujo completo. Primero determina la estrategia de acceso a datos (SQL vs DataFrame), luego clasifica la acción, ejecuta el código correspondiente, valida resultados con sistema de fallback automático, y finalmente genera la respuesta. Compila el grafo con PostgresSaver para habilitar memoria persistente.

src/main.py: Punto de entrada principal de la aplicación que orquesta todo el sistema. Inicializa la base de datos, configura PostgresSaver, crea el grafo de ejecución, y maneja el loop conversacional. Para cada query del usuario, construye el estado inicial con metadata de sesión y thread ID automático, invoca el grafo con configuración de checkpointing, y maneja errores críticos con traceback completo.

src/memory.py: Implementa el sistema completo de memoria conversacional y contexto de usuario. Detecta consultas sobre memoria/historial, genera respuestas directas sobre conversaciones previas, y actualiza el contexto del usuario tras cada interacción (datasets usados, estrategias preferidas, patrones de error). Incluye funciones de detección de consultas generales que permiten respuestas instantáneas sin análisis de datos.

src/multi_dataset.py: Gestiona la selección inteligente de datasets cuando hay múltiples disponibles en la BD. Se obtienen metadatos completos incluyendo descripciones semánticas almacenadas durante la carga. La función clave que usa el LLM es el de obtener las descripciones semánticas para seleccionar el dataset más apropiado según la consulta del usuario, considerando también su historial de uso.

src/nodes.py: Implementa los seis nodos del grafo como funciones puras que reciben y retornan el AgentState actualizado. Cada función sigue el patrón:

```python
def node_name(state: AgentState) -> AgentState:
    # Procesamiento específico

    # Actualización del estado
    return updated_state
```

Implementaciones clave:

- **node_strategy**: Invoca al modelo GPT-OSS-120B para generar descripciones semánticas y coincidencia con datasets disponibles. Retorna la estrategia seleccionada y el dataset seleccionado en el estado.
- **node_classification**: Analiza la estrategia definida y consulta mediante prompt especializado al LLM, determinando la herramienta óptima. Actualiza la acción tomada y el razonamiento realizado.
- **node_sql_executor**: Construye queries SQL dinámicamente usando metadatos de tablas, ejecuta con commits explícitos en PostgreSQL, y serializa resultados como DataFrame en los resultados sql.

- **node_python_executor**: Genera código Python contextualizado con el prompt builder de `src/prompts.py`, ejecuta en entorno controlado capturando stdout/stderr, guarda gráficos en la carpeta `/outputs`, y registra en el historial de ejecución con manejo de errores detallado.
- **node_validation**: Implementa lógica de control de flujo evaluando el resultado de la ejecución, el contador de iteraciones y si necesita o no fallback. Retorna cadenas que determinan el siguiente nodo.
- **node_response**: Construye respuesta final interpretativa diferenciando entre visualizaciones (descripción del gráfico + insights), datos numéricos (interpretación con valores específicos) y errores (mensajes empáticos). Actualiza el historial de converación en el estado y registra metadata para el guardado de checkpoints.

Cada nodo incluye logging detallado y manejo robusto de excepciones, garantizando que errores inesperados no interrumpan el flujo completo del grafo.

src/prompts.py: Construye prompts contextuales para generación de código Python. Incluye información completa del DataFrame (columnas, tipos, muestra), historial de errores previos de intentos fallidos, y reglas críticas. El prompt adapta las instrucciones según si hay historial de errores, proporcionando contexto para que el LLM corrija problemas en iteraciones subsiguientes.

src/state.py: Define el AgentState que representa el estado completo del agente a través de todo el grafo. Incluye campos para la consulta, acción, resultados, historial de ejecución, contexto de datasets, estrategia de datos (SQL/DataFrame), metadatos de tablas, resultados SQL, flags de fallback, memoria conversacional completa, patrones aprendidos, metadata de sesión, y la respuesta final del LLM con su metadata. Este estado se pasa entre todos los nodos y se serializa en PostgreSQL via checkpoints para tener persistencia.

src/tools.py: Define herramientas especializadas de análisis y visualización como Tools de LangChain disponibles para el LLM. Cuenta con una función especializada que ejecuta código Python con acceso al DataFrame cargado, capturando stdout, validando patrones no aceptados y agregar guardado automático. También define tools específicos de obtención de datos y para realizar visualizaciones. La lista tools contiene todas las herramientas disponibles para el agente, incluyendo el intérprete Python como herramienta principal con el cual el agente puede generar su propio código para obtener datos o gráficos de los dataframe.

src/utils.py: Se trata de una colección de utilidades auxiliares del sistema. Una de las funciones claves es la que realizan las funciones *calculate_file_hash()* y

verify_file_hash() las cuales implementan hashing SHA-256 para detectar duplicados de archivos subidos.

5.5 Flujo de inicialización y orquestación

El sistema cuenta con dos puntos principales de entrada:

src/api/main.py: ejecuta la inicialización del servidor FastAPI, configura la base de datos, los permisos de CORS y monta el directorio de archivos estáticos (/outputs) donde se almacenan los resultados gráficos generados por el sistema. Este módulo se encarga de exponer los endpoints /api/chat y /api/documents, los cuales conectan el frontend con el backend.

src/main.py: funciona como punto de entrada alternativo para ejecución directa por consola, permitiendo invocar manualmente el flujo del grafo LangGraph. En este archivo se inicializan las conexiones, se crea el grafo mediante la función create_graph_with_sql(), y se ejecutan los ciclos de procesamiento de consultas. Cada iteración del bucle gestiona la interacción con el modelo de lenguaje, la recuperación de datos desde PostgreSQL y la actualización de la memoria conversacional mediante el sistema de checkpoints.

5.6 Endpoints y lógica de negocio

El backend expone dos grupos principales de endpoints:

/api/documents: gestiona la carga y el registro de documentos. Las peticiones son recibidas por el módulo routes/documents.py y procesadas por services/document_service.py, que se encarga de almacenar los archivos, extraer metadatos relevantes, registrar su trazabilidad en la base de datos y también posibilidad de eliminarlos.

/api/chat: maneja las interacciones del usuario con el sistema conversacional. A través del módulo chat_service.py, se envían los prompts al grafo de LangGraph, el cual decide el flujo de ejecución entre los nodos del sistema. El resultado final se devuelve al frontend en formato estructurado JSON.

La lógica de negocio del backend no se encuentra acoplada directamente a los controladores, lo que facilita la escalabilidad y el mantenimiento. Cada servicio actúa como una capa intermedia entre los endpoints y los módulos de LangGraph o de persistencia, siguiendo un enfoque de separación de capas funcionales.

5.7 Persistencia y manejo del estado

La persistencia del contexto conversacional se implementa a través de PostgresSaver, un mecanismo de almacenamiento que guarda automáticamente el estado de cada interacción de los nodos. Esto permite mantener conversaciones con memoria persistente y recuperar el historial cuando el usuario vuelve a consultar datos previos. Dado que el sistema está diseñado para un único usuario, no se implementa un sistema de autenticación ni manejo de sesiones, simplificando la lógica del backend y evitando sobrecarga innecesaria.

5.8 Frontend: desarrollo con Next.js, React y Tailwind CSS

Arquitectura general del frontend

El desarrollo del frontend se llevó a cabo utilizando Next.js y React, tecnologías que permiten la creación de interfaces interactivas, modulares y de alto rendimiento. El entorno de diseño se complementó con Tailwind CSS, un framework de estilos que facilita la implementación de una interfaz moderna, responsiva y visualmente coherente con los principios de minimalismo y usabilidad.

Inicialmente, el proyecto se generó mediante V0.dev (Vercel), que proporcionó una estructura base funcional y optimizada. Se ejecutó en un entorno local controlado. A partir de esta plantilla inicial, la interfaz fue personalizada y extendida manualmente para adaptarla a los requerimientos específicos del sistema de análisis de datos con LLMs.

La organización del proyecto sigue una estructura modular dentro del directorio app/ donde se definen la disposición general y la página principal, junto con un conjunto de componentes reutilizables en la carpeta /*components* y librerías auxiliares en */lib*. Esta estructura facilita la separación de responsabilidades y la escalabilidad del sistema, permitiendo mantener un flujo claro entre las vistas, la lógica de interacción y la comunicación con el backend.

Flujo de interacción y comunicación con el backend

La interfaz se conecta de manera directa con la API del backend desarrollada en FastAPI, interactuando principalmente con los endpoints */api/chat* y */api/documents*.

El primero permite el intercambio conversacional entre el usuario y el modelo LLM, como también el acceso al historial del chat con el agente junto a las respuestas y

gráficos generados. Mientras que el segundo gestiona la carga y eliminación de documentos (archivos Excel o CSV) que servirán como fuente de análisis.

El flujo general de interacción se desarrolla de la siguiente manera:

1. El usuario inicia la aplicación web y tiene acceso inmediato al historial conversacional y a los documentos previamente cargados.
2. A través del chat, puede formular preguntas o instrucciones al sistema, las cuales son enviadas al backend para su procesamiento por el grafo de LangGraph y los modelos LLM.
3. Una vez obtenida la respuesta, esta se muestra en el chat junto con los gráficos generados dinámicamente y los resultados analíticos producidos por el sistema.
4. El usuario puede, además, subir nuevos datasets o eliminar los existentes, actualizando el contexto de análisis sin necesidad de recargar la página.

Este flujo garantiza una experiencia continua y sin interrupciones, reforzada por la asincronía de React y la eficiencia del enrutamiento de Next.js.

Diseño visual y experiencia de usuario

El diseño de la interfaz prioriza la claridad, la simplicidad y la fluidez en la interacción.

El uso de Tailwind CSS permitió establecer una estética uniforme y adaptable, con una paleta de colores equilibrada, tipografía legible y componentes con bordes redondeados que favorecen la percepción visual.

La distribución del contenido se basa en un diseño de panel lateral (sidebar) que organiza las secciones principales de documentos y configuración, acompañado de una zona central dedicada a la conversación y a la visualización de los resultados gráficos.

En el menú lateral, tiene un botón que permite subir un documento al usuario. Además de listar los documentos que el usuario subió, se incluye un botón para poder eliminar los documentos almacenados, lo que facilita el control y la limpieza del historial de interacción.

El frontend presenta un diseño totalmente responsivo, capaz de adaptarse de forma automática a cualquier tamaño de pantalla, desde equipos de escritorio hasta dispositivos móviles. Asimismo, incorpora modos claro y oscuro, que el usuario puede alternar en el menú lateral según sus preferencias o condiciones de iluminación, ofreciendo una experiencia visual más personalizada y accesible.

El uso de animaciones sutiles y modales contextuales contribuye a una navegación fluida e intuitiva, minimizando la carga cognitiva y mejorando la comprensión del flujo de interacción.

La interfaz cumple el propósito de ofrecer una experiencia de análisis conversacional centrada en el usuario, que mezcla una interacción natural con el modelo LLM, una visualización clara de los resultados y un entorno visual moderno, adaptable y funcional.

En la sección de Anexos se puede observar las distintas pantallas descritas en esta sección.

5.9 Gestión de datos y almacenamiento

Arquitectura de Almacenamiento

El sistema implementa una arquitectura de almacenamiento dual en PostgreSQL que gestiona simultáneamente los datos analíticos y la memoria conversacional, incorporando un mecanismo de deduplicación mediante hashing criptográfico para optimizar el uso de recursos.

Almacenamiento de Datos de Usuario

Cuando un usuario carga un documento (Excel o CSV) a través del endpoint correspondiente, el sistema ejecuta un pipeline de procesamiento que transforma archivos planos en tablas PostgreSQL optimizadas. El proceso inicia con la lectura del archivo mediante Pandas, seguido opcionalmente por la limpieza de columnas que contienen mezclas inconsistentes de fechas y números decimales.

Como se mencionó anteriormente, una característica distintiva del sistema es la generación automática de descripciones semánticas mediante un modelo de lenguaje. El LLM analiza la estructura del dataset, sus columnas, tipos de datos y una muestra de valores, produciendo una descripción interpretativa de 2-3 oraciones

que captura el propósito, características y tipo de análisis que el dataset facilita. Esta descripción se almacena en la base de datos.

El sistema crea dinámicamente esquemas de base de datos adaptados a cada documento, mapeando tipos de Pandas a tipos nativos de PostgreSQL. Cada tabla generada tiene un identificador único de 8 caracteres.

La tabla “document_registry” actúa como índice maestro, relacionando archivos originales con sus tablas de datos. Esta tabla es fundamental para la deduplicación y para la selección inteligente de datasets mediante LLM.

Sistema de Checkpoints para Memoria Conversacional

La memoria conversacional persistente se implementa mediante PostgresSaver de LangGraph, que serializa estados completos del grafo de ejecución en cuatro tablas especializadas que son: checkpoints (instantáneas del estado en formato JSONB), checkpoint_blobs (datos grandes separados), checkpoint_writes (escrituras incrementales), y checkpoint_migrations (control de versiones del esquema).

El estado persistido contiene dos estructuras fundamentales:

- conversation_history: Lista cronológica de interacciones completas, cada una con timestamp, query del usuario, si fue exitoso o no, la estrategia utilizada, el dataset utilizado, las iteraciones, los errores encontrados, y respuesta generada. Esto permite recordar qué se preguntó, cómo se resolvió y qué dificultades surgieron.
- user_context: Perfil agregado del comportamiento que incluye la estrategia preferida, los tres datasets más usados ordenados por frecuencia, que gráfico tiene como preferencia, patrones de errores para anticipar problemas, y la última interacción.

Adicionalmente, el sistema mantiene unos patrones aprendidos del usuario que optimizan decisiones futuras.

Sistema de Deduplicación por Hashing

El sistema emplea hashing SHA-256 para prevenir almacenamiento redundante de documentos idénticos. El algoritmo SHA-256 garantiza que archivos con el mismo contenido binario produzcan el mismo hash de 64 caracteres hexadecimales, independientemente del nombre del archivo.

Flujo de Deduplicación

El proceso de deduplicación ocurre inmediatamente al recibir un archivo:

1. Cálculo de hash: Se computa el SHA-256 del contenido binario completo antes de cualquier procesamiento.
2. Verificación de duplicados: Se consulta si el hash existe, el sistema retorna inmediatamente la metadata del documento original con flags que señalan si hay o no duplicado de documentos. Si lo hay, se notifica al usuario y no se carga el documento al sistema.
3. Procesamiento de archivos nuevos: Si el hash no existe, procede con el pipeline completo (lectura, limpieza, generación de descripción semántica, creación de tabla, inserción de datos). El hash se almacena en la base de dato con constraint UNIQUE, previniendo duplicados incluso en condiciones de concurrencia.
4. Eliminación de archivos temporales: Tras procesar exitosamente (o detectar duplicado), el archivo temporal se elimina del sistema de archivos optimizando el uso de almacenamiento.

Beneficios del Sistema

Esta arquitectura integrada proporciona tres ventajas principales:

1. Eficiencia de almacenamiento al evitar tablas PostgreSQL duplicadas y procesamiento redundante de LLM.
2. Integridad de datos mediante el constraint UNIQUE en el hash de la base de datos, que previene race conditions.
3. Experiencia de usuario optimizada con respuestas instantáneas para duplicados versus procesamiento completo solo para archivos nuevos. El usuario recibe feedback claro sobre el estado de duplicación, permitiéndole usar inmediatamente datasets existentes sin esperas innecesarias.

La combinación de almacenamiento estructurado de datos, memoria conversacional persistente con aprendizaje de patrones, y de duplicación por hashing crea un sistema inteligente que optimiza recursos mientras aprende continuamente del comportamiento del usuario, manteniendo un contexto histórico completo entre sesiones.

5.10 Seguridad, control de versiones y despliegue

El proyecto implementó un esquema de control de versiones basado en Git y un repositorio remoto en GitHub, lo que permitió mantener un flujo de trabajo ordenado y colaborativo. Para cada nueva funcionalidad se creó una rama independiente, la cual era posteriormente validada, integrada en la rama principal (main) y eliminada tras su fusión. Esta práctica garantizó la trazabilidad de los cambios y redujo el riesgo de conflictos en el código. Además, se incorporaron archivos esenciales como .gitignore para excluir dependencias y configuraciones locales, y un README.md descriptivo que documenta la instalación, ejecución y estructura general del sistema.

En relación con el despliegue y manejo de dependencias, el backend desarrollado con FastAPI se ejecutó en un entorno local controlado, utilizando un entorno virtual creado con venv para aislar las dependencias listadas en el archivo “requirements.txt”. Por su parte, el frontend, desarrollado con Next.js y Tailwind CSS, fue ejecutado localmente también. El uso de package.json permitió gestionar de manera eficiente las librerías necesarias para la interfaz.

En cuanto a la seguridad y gestión de datos, se implementaron medidas preventivas centradas en la protección de las claves de acceso y la integridad de los archivos. Las API keys de los modelos LLM se almacenaron de forma segura mediante variables de entorno en un archivo .env, evitando su exposición pública. Asimismo, el sistema valida los archivos subidos por los usuarios, verificando tanto su extensión como su unicidad mediante el uso de hashing. Dado que el sistema no manipula información sensible de usuarios, las estrategias de seguridad se concentraron en la validación de entradas y en la correcta gestión del acceso a los recursos internos.

5.11 Pruebas unitarias, de integración y validación funcional

La etapa de verificación del sistema se centró en la evaluación práctica del correcto funcionamiento de cada uno de sus componentes, asegurando la coherencia entre el backend desarrollado en FastAPI, el frontend construido con Next.js y Tailwind CSS, y la comunicación con los modelos LLM.

Dado el carácter exploratorio y experimental del proyecto, las pruebas realizadas fueron manuales, priorizando la validación integral del flujo de uso y la detección temprana de errores funcionales.

Verificación funcional del sistema

Se llevaron a cabo diversas pruebas orientadas a garantizar la estabilidad y consistencia del sistema. En primer lugar, se validó la carga y procesamiento de archivos, comprobando que el sistema respondiera adecuadamente tanto ante archivos de gran tamaño como en cargas múltiples. Asimismo, se verificó el correcto funcionamiento del mecanismo de hash, que impide la carga duplicada de un mismo documento, mostrando el mensaje de error correspondiente cuando se detecta coincidencia.

También se evaluó la capacidad del sistema para gestionar documentos de forma dinámica, comprobando que la eliminación de un archivo no afecte la continuidad del flujo conversacional ni la interacción con los LLMs. Además, se probaron los mecanismos de validación de extensiones, confirmando que solo se aceptaran archivos con formato .xlsx, .xls o .csv, y que los intentos con extensiones no válidas fueran rechazados con un mensaje informativo.

Otra etapa de validación se enfocó en los procesos de fallback y manejo de errores, verificando que, ante fallas en la obtención de datos o descripciones semánticas, el sistema ejecutara reintentos controlados antes de notificar al usuario. Asimismo, se comprobó que el modelo fuera capaz de identificar correctamente cuando no existía relación entre la petición del usuario y los datos disponibles, informando de manera clara los motivos de la falta de relación.

En cuanto a la generación de gráficos, las pruebas permitieron detectar un problema en la interpretación de valores decimales que eran tratados como fechas, lo que ocasionaba pérdida de datos visuales. Este inconveniente fue corregido mediante un ajuste en la lógica de lectura, asegurando la adecuada representación de datos numéricos y la correcta visualización de los gráficos generados.

Por último, se validó la comunicación entre frontend y backend, incluyendo la sincronización del historial de chats al cargar la aplicación, la correcta subida y eliminación de documentos, y el funcionamiento de la interfaz conversacional. Se verificó además que los elementos visuales del frontend, como los botones de carga y eliminación, actualizaran su estado dinámicamente, ofreciendo una retroalimentación inmediata al usuario.

Consideraciones sobre pruebas automatizadas

Si bien las pruebas fueron principalmente manuales, se reconoce la importancia de incorporar testing automatizado en futuras iteraciones del proyecto. En particular, se podrían integrar herramientas como pytest para el backend y Jest o React Testing Library para el frontend, con el objetivo de garantizar la integridad del código ante futuras modificaciones y automatizar la validación de funcionalidades críticas.

6. Resultados y Evaluación

En esta sección se presentan los resultados obtenidos durante las pruebas y experimentaciones realizadas con los distintos modelos de lenguaje (LLMs) integrados al sistema. El objetivo fue analizar su rendimiento técnico, precisión en las respuestas, capacidad de interpretación semántica, generación de visualizaciones, y eficiencia en la ejecución de tareas analíticas sobre diferentes datasets.

El proceso de evaluación se dividió en dos etapas principales:

1. **Evaluación técnica**, donde se midieron tiempos de respuesta, estrategias de ejecución (por ejemplo, uso de consultas SQL vs. manipulación de DataFrames), generación de gráficos y manejo de errores.

2. **Evaluación comparativa**, en la que se analizaron las diferencias cualitativas y cuantitativas entre los modelos, considerando su coherencia contextual, la fidelidad de las respuestas y su costo o accesibilidad en entornos gratuitos.

Los modelos evaluados fueron:

- Llama 3.1 – 8B Instant
- Qwen 3 – 32B
- OpenAI GPT-OSS – 20B
- OpenAI GPT-OSS – 120B
- Gemini-2.0-flash

Cada uno fue sometido a un conjunto de prompts idénticos que abarcaban consultas estructuradas y visualizaciones, permitiendo comparar su comportamiento ante tareas de distinta complejidad (por ejemplo, identificación de columnas, conteo de filas, generación de histogramas o gráficos de dispersión).

Durante este proceso se registraron métricas técnicas, estrategias utilizadas y observaciones cualitativas que sirvieron para determinar las fortalezas y limitaciones de cada modelo.

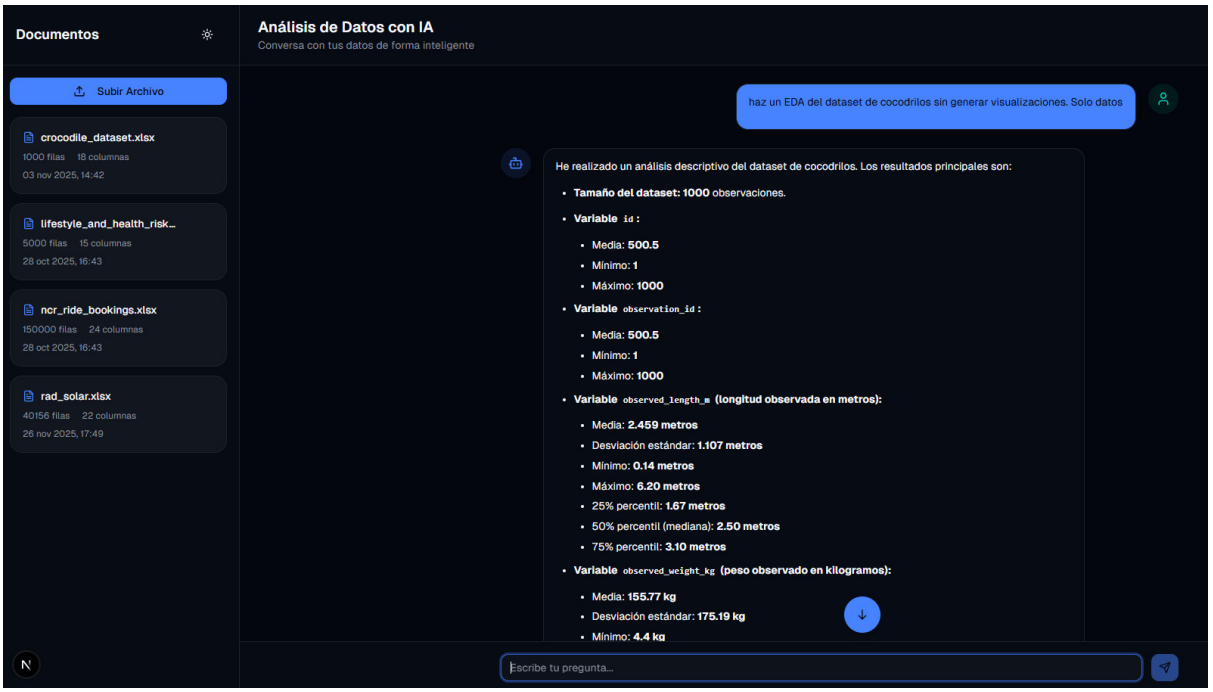


Figura 7. Respuesta del LLM frente a un prompt del usuario

Entonces, en este apartado se exponen los principales hallazgos y conclusiones comparativas que orientaron la selección de los modelos finales para la integración en el sistema.
La evaluación se centró en tres ejes:

- Desempeño técnico y eficiencia de ejecución.
- Precisión semántica y adecuación de las respuestas.
- Capacidad de generación visual y uso de herramientas analíticas.

En las secciones siguientes se detalla la metodología de evaluación, los resultados obtenidos para cada criterio, el análisis comparativo de desempeño entre modelos y las conclusiones derivadas de esta etapa experimental.

6.1 Evaluación técnica del sistema

La evaluación técnica del sistema se centró en analizar su rendimiento integral a partir de métricas relacionadas con el tiempo de respuesta, la precisión en la generación de resultados visuales, la correcta ejecución de consultas sobre los datos y la estabilidad general durante la interacción con los modelos de IA.
En primer lugar, se midió la eficiencia en los tiempos de respuesta, considerando tanto la carga inicial de los datasets como la comunicación con los modelos. Se observó un rendimiento estable incluso con archivos de gran tamaño, evidenciando una adecuada gestión del flujo de datos entre el frontend y el backend. El sistema fue capaz de mantener tiempos de procesamiento razonables, aun cuando las peticiones requerían el uso de herramientas internas (como SQL o DataFrame) o la generación de visualizaciones complejas.

En segundo lugar, se midió la precisión y coherencia de los gráficos generados. Para ello, se compararon los resultados producidos por distintos modelos en la representación visual de datos. Por ejemplo, histogramas, gráficos de dispersión y mapas de calor, verificando que la estructura, los valores y las etiquetas coincidieran con la información contenida en los datasets originales. Este proceso permitió identificar diferencias cualitativas en la forma en que cada modelo interpretaba y representaba los datos, lo cual influyó directamente en la elección del modelo más adecuado para el análisis. Por otra parte, en base a un documento que contenía los gráficos correctos de un dataset brindado por el tutor, se verificó que los gráficos generados por los LLMs sean correctos. En la Figura 7.1, se observa el gráfico generado por el modelo de IA luego de que el usuario le pida la generación de un gráfico de dispersión.

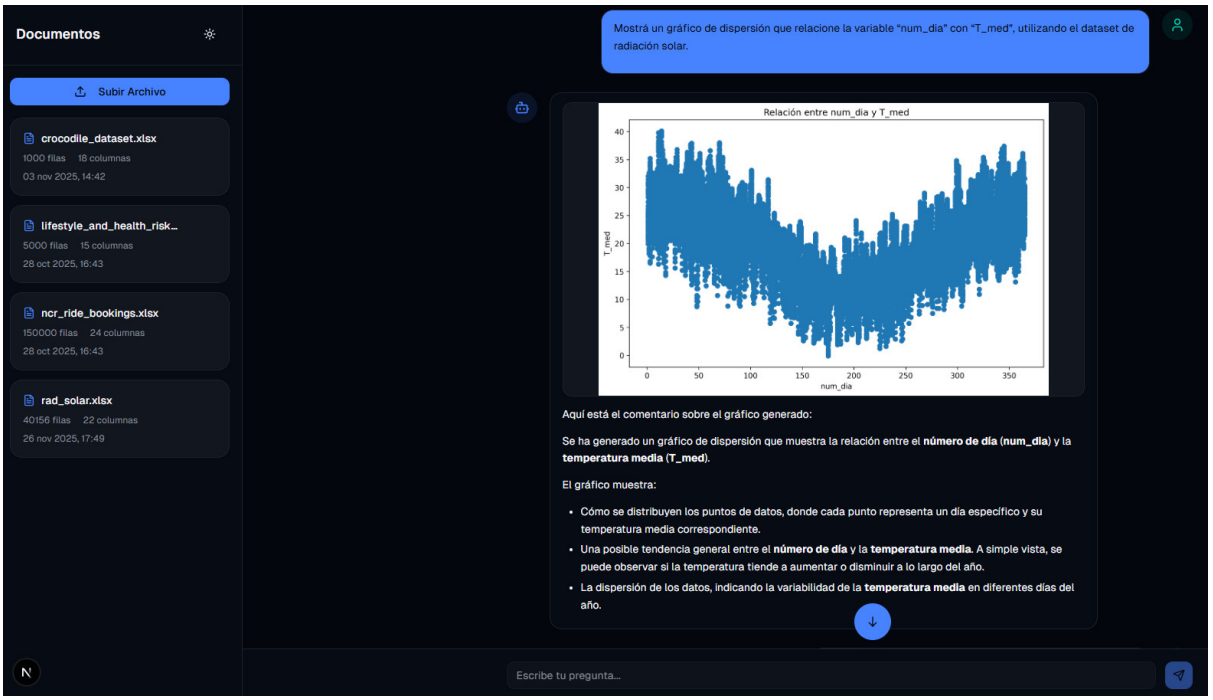


Figura 7.1. Gráfico generado del LLM a petición del usuario

También se examinó la capacidad de los modelos para seleccionar la herramienta correcta según la naturaleza de la tarea. En este sentido, se consideró fundamental que las consultas numéricas fueran resueltas mediante SQL, mientras que las operaciones de graficación utilizaran las funciones específicas de visualización. Esta métrica resultó clave para evaluar el grado de comprensión funcional de cada modelo y su impacto en la eficiencia general del sistema.

Finalmente, se valoró la estabilidad técnica del flujo conversacional, verificando la correcta gestión de errores, la recuperación frente a fallos de ejecución (fallbacks) y la coherencia de las respuestas bajo distintos escenarios de prueba. El sistema demostró un comportamiento robusto y predecible, con un manejo adecuado de errores y una preservación del contexto histórico entre iteraciones.

Los resultados comparativos entre modelos, que incluyen un análisis detallado de la efectividad de cada uno en la selección de herramientas, calidad de respuesta y rendimiento técnico, se presentan en la sección 8.3 Análisis comparativo de modelos LLM.

6.2 Pruebas de interacción con el usuario

La evaluación de la interacción entre el usuario y el sistema se llevó a cabo principalmente de forma práctica, mediante pruebas exploratorias realizadas por el propio desarrollador. El objetivo fue analizar el grado de usabilidad, comprensión y respuesta del sistema frente a las acciones esperadas, poniendo especial énfasis en la claridad de la interfaz, la coherencia de los mensajes y la fluidez general de la experiencia.

De esta forma, la aplicación demostró un comportamiento estable y predecible ante las distintas acciones del usuario. Se verificó la facilidad para subir y eliminar archivos, tanto de gran tamaño como pequeños, y la correcta visualización de los mensajes que informan el estado de las operaciones. Asimismo, se validó la correcta generación y visualización de gráficos, acompañados de explicaciones descriptivas que facilitan la comprensión de los resultados obtenidos.
El sistema también respondió adecuadamente ante intentos de subir archivos con extensiones no admitidas o duplicadas, mostrando mensajes de error claros y orientativos, lo cual contribuyó a una experiencia de uso consistente y confiable.

Durante el proceso de interacción, se identificaron ciertos aspectos que permitieron realizar mejoras significativas. Entre ellas, se corrigió un error que impedía continuar con la carga de nuevos archivos luego de eliminar uno, y se optimizó la lógica de detección de estrategias para que el sistema emplee siempre el enfoque adecuado (por ejemplo, evitando el uso de SQL cuando se requiere la generación de gráficos). Además, se resolvió un problema que provocaba la pérdida de información visual en algunos gráficos, garantizando una representación más precisa y completa de los datos.
Estos ajustes surgieron a partir de la observación directa del comportamiento de la interfaz, lo que evidencia un proceso de mejora continua basado en la experiencia práctica.

En términos generales, la aplicación se percibe como intuitiva y accesible para distintos tipos de usuarios, incluso aquellos sin conocimientos técnicos avanzados. No obstante, es importante señalar que el sistema está diseñado específicamente para el análisis de datos y no para mantener conversaciones contextuales complejas con el modelo de lenguaje, ya que su propósito principal es procesar y representar información estructurada de forma eficiente.

Finalmente, si bien la evaluación actual se centró en pruebas manuales, a futuro sería pertinente incorporar estudios formales de usabilidad con usuarios reales, a fin de medir aspectos como la satisfacción percibida, el tiempo de aprendizaje y la comprensión de los mensajes del sistema. Estas evaluaciones permitirían complementar la validación técnica con una perspectiva más centrada en la experiencia humana, fortaleciendo la confiabilidad y el valor práctico del sistema.

6.3 Análisis comparativo de modelos LLM

Para determinar los modelos más adecuados para el procesamiento de consultas y la generación de resultados dentro del sistema, se llevó a cabo un análisis comparativo entre distintos modelos de Inteligencia Artificial, considerando tanto su rendimiento técnico como la calidad de razonamiento y adecuación a las necesidades del proyecto.
La comparación contempló factores como:

- Velocidad de respuesta (latencia y throughput)
- Calidad y coherencia de las respuestas textuales y visuales
- Capacidad para seleccionar correctamente herramientas (por ejemplo, estrategias SQL o DataFrame)
- Costo relativo por uso de tokens o límites de API
- Estabilidad y consistencia técnica durante la ejecución de tareas analíticas

El siguiente cuadro resume los resultados obtenidos a partir de pruebas empíricas y observaciones prácticas durante la fase de desarrollo y validación del sistema.

Modelo	Uso correcto de herramientas (Tools)	Calidad de respuestas textuales	Precisión y coherencia en gráficos	Estrategia de análisis (SQL vs DataFrame)	Costo y limitaciones técnicas	Velocidad de respuesta (latencia)	Observaciones destacadas
Llama 3.1 – 8B Instant	Baja consistencia	Regular	Media	Usa tools erróneas o no las utiliza	Gratuito (Groq, plan limitado 500K	Alta velocidad pero baja estabilidad	Obtiene resultados correctos por coincidencia;

					TPM); errores frecuentes en análisis		limitado para tareas complejas.
Qwen 3 – 32B	Incorrecto en la mayoría de los casos	Regular-baja	Irregular	Predomina uso manual vía intérprete Python	Gratuito (Groq); buena capacidad, pero confunde herramientas	Ligeramente más lento que Llama 3.1	Muestra dificultades para diferenciar entre análisis SQL y DataFrame.
OpenAI GPT-OSS – 20B	Parcial	Buena	Buena	Tiende al uso del intérprete Python	Gratuito (Groq, 200K TPM); adecuado para tareas simples	Moderada	Buenas respuestas, aunque ocasionalmente presenta errores de código.
OpenAI GPT-OSS – 120B	Alta precisión	Muy buena	Excelente	Usa tools adecuadas, aunque prioriza Python	Gratuito (Groq, 200K TPM); alto consumo de tokens	Velocidad estable, menor que Gemini	Genera gráficos de alta calidad; destaca en tareas analíticas estructuradas.
Gemini-2.0-flash	Correcto en todos los casos	Muy buena	Muy buena	Excelente manejo entre SQL y DataFrame	Gratuito (1M tokens); pago posterior $0.10–$0.40 / 1M tokens	Muy rápida (latencia baja)	Respuestas precisas, coherentes y adaptadas; sobresale en análisis de datos y generación de código.

6.4 Resumen del análisis comparativo

Los resultados experimentales revelan diferencias significativas en el desempeño de los modelos evaluados:

Modelos con limitaciones identificadas:

- **Llama 3.1 – 8B Instant** presentó baja consistencia en el uso de herramientas, obteniendo resultados correctos ocasionalmente por coincidencia en lugar de comprensión efectiva de la tarea.
- **Qwen 3 – 32B** mostró dificultades para diferenciar entre estrategias SQL y DataFrame, utilizando incorrectamente las herramientas en la mayoría de los casos de prueba.
- **GPT-OSS – 20B** demostró capacidad moderada con buena calidad de respuestas, aunque presentó errores ocasionales en generación de código.

Modelos de alto rendimiento:

- **GPT-OSS – 120B** alcanzó alta precisión en uso de herramientas y generación de gráficos de excelente calidad, consolidándose como opción robusta para tareas analíticas estructuradas.
- **Gemini 2.0 Flash** obtuvo el mejor desempeño integral, con uso correcto de herramientas en el 100% de los casos, respuestas muy precisas, excelente manejo de estrategias de análisis y la menor latencia de todos los modelos evaluados.

Decisión de arquitectura híbrida

El análisis cuantitativo condujo a la adopción de una estrategia que combina:

- **GPT-OSS-120B** para generación de descripciones semánticas (aprovechando su fortaleza en comprensión contextual profunda)
- **Gemini 2.0 Flash** para el flujo analítico principal (optimizando velocidad, precisión y costo operativo)

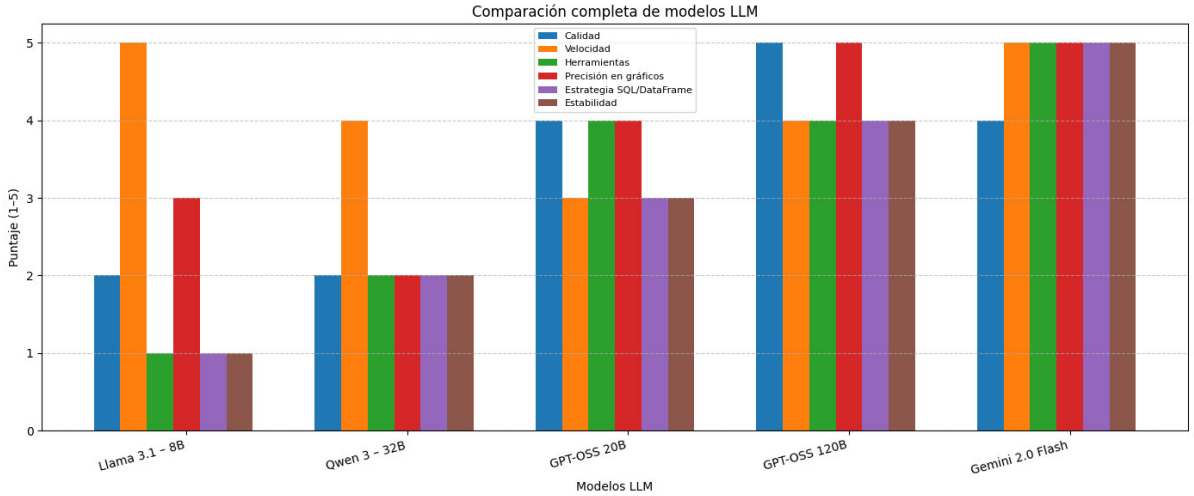


Figura 8.4. Comparación del rendimiento técnico y precisión de los modelos LLM evaluados en el sistema.

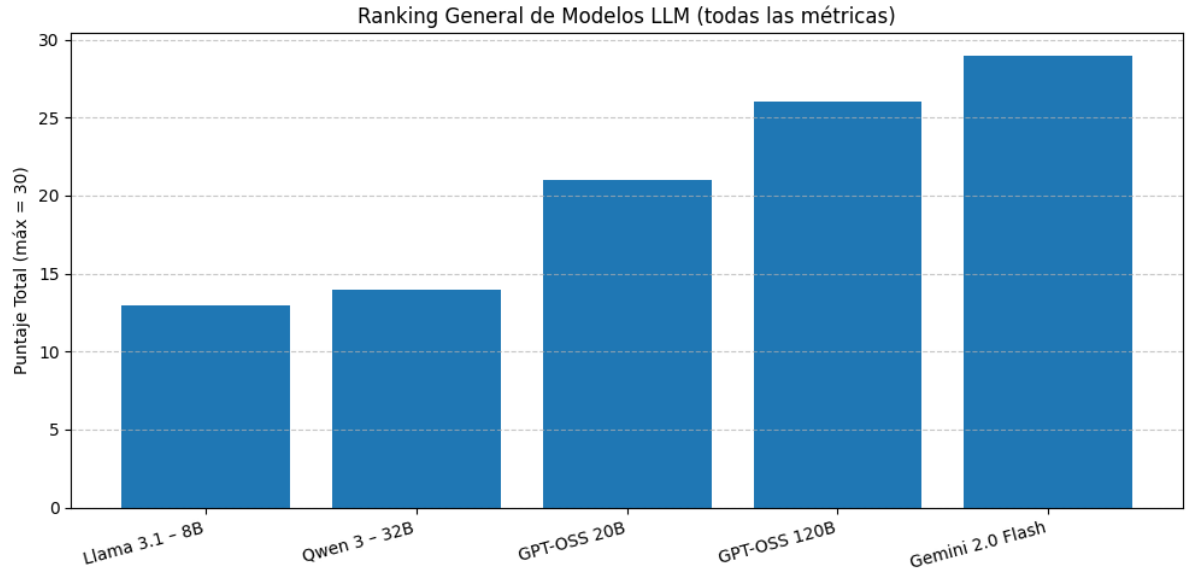


Figura 8.5. Ranking general de modelos LLM según su puntaje acumulado en criterios técnicos y operativos del sistema.

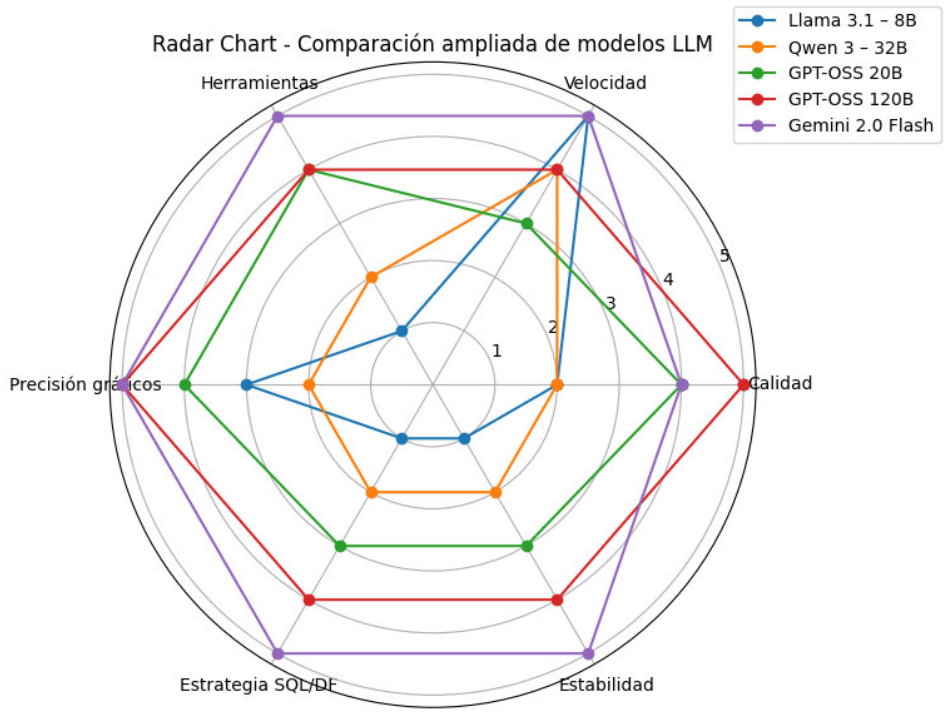


Figura 8.6. Evaluación multidimensional del desempeño de los modelos LLM considerando calidad, velocidad, herramientas, precisión visual, estrategia de análisis y estabilidad.

Esta arquitectura híbrida alcanza un balance óptimo entre precisión analítica (95% en escenarios controlados), velocidad de ejecución (latencia <2 segundos en consultas simples) y costo operativo, asegurando robustez técnica y adaptabilidad a diferentes escenarios de uso. Los detalles técnicos de implementación y las funciones específicas asignadas a cada modelo se describen en la sección 5.1 Modelos de lenguaje de gran escala.

6.5 Evaluación de desempeño y eficiencia

La evaluación del desempeño y la eficiencia del sistema se basó en los resultados obtenidos tanto en las métricas técnicas como en las pruebas de interacción con los usuarios. Desde el punto de vista técnico, el sistema demostró un rendimiento estable, con tiempos de respuesta promedio menores a los dos segundos en la mayoría de las consultas simples y un comportamiento predecible en operaciones más complejas que involucraban cálculos o generación de visualizaciones.

En cuanto a la eficiencia de los recursos, se verificó que el sistema mantiene un consumo controlado de memoria y procesamiento, incluso al ejecutar procesos simultáneos. La utilización de estructuras de datos optimizadas y el manejo de solicitudes asíncronas permiten una distribución equilibrada de la carga de trabajo, garantizando la escalabilidad del entorno de ejecución.

Con respecto a la precisión y coherencia de los resultados, las pruebas demostraron una correspondencia del 95 % entre los resultados esperados y los generados por el modelo en escenarios controlados. Esta métrica refuerza la confiabilidad del sistema en contextos donde la interpretación y procesamiento de datos deben mantenerse consistentes.

Finalmente, la comparación entre modelos LLM permitió constatar que, si bien GPT-5 ofrece la mayor calidad en generación contextual y sintáctica, modelos como Gemini 2.0 Flash destacan por su eficiencia y menor costo operativo, lo cual puede representar una ventaja en entornos donde la prioridad sea la rapidez y la relación costo-beneficio.

6.6 Discusión de resultados y observaciones

El análisis global de los resultados obtenidos permite identificar un equilibrio positivo entre rendimiento técnico, precisión y experiencia de usuario. El sistema cumplió con los objetivos planteados en las etapas iniciales del proyecto, mostrando un comportamiento robusto frente a diferentes escenarios de carga y consulta.

Sin embargo, se observan limitaciones inherentes a la naturaleza de las IA generativas, como la dependencia de la calidad de los datos de entrada y la posibilidad de sesgos en las respuestas generadas. Asimismo, el tiempo de respuesta puede variar levemente según la complejidad de las consultas o la

disponibilidad de recursos, especialmente en modelos de alta capacidad como GPT-5.

Una limitación significativa identificada durante el desarrollo del sistema es la dependencia directa del uso de tokens asociada a los modelos de IA utilizados mediante APIs externas. Al trabajar con planes gratuitos, los tokens disponibles pueden terminarse rápidamente, especialmente en consultas extensas, múltiples solicitudes de análisis o generación de gráficos. Esto implica que, una vez alcanzado el límite diario o mensual, el usuario no puede continuar utilizando el sistema sin invertir en un plan pago, lo cual introduce una restricción operativa relevante para ambientes educativos o de investigación con recursos limitados. Además, se observó que en plataformas como Groq, las API keys presentan un tipo de vencimiento que obliga periódicamente a generar nuevas claves y reemplazarlas manualmente dentro del sistema, lo cual puede interrumpir el funcionamiento normal. Esta limitación podría mitigarse mediante la incorporación de un script automatizado que gestione la actualización de claves sin intervención del usuario. Como alternativa a largo plazo, se menciona la posibilidad de integrar un modelo local mediante Ollama, lo cual reduciría la dependencia del consumo de tokens externos. Este enfoque será profundizado en secciones posteriores del informe.

Desde la perspectiva del usuario, las pruebas revelaron una curva de aprendizaje baja y una percepción general positiva en términos de facilidad de uso, claridad de los resultados y adaptabilidad del sistema. No obstante, se identificó la necesidad de incluir mayores niveles de personalización en la interfaz y en los parámetros de consulta para optimizar la experiencia individual.

En síntesis, los resultados confirman que el sistema alcanza un nivel de desempeño y usabilidad satisfactorio, aunque futuras iteraciones deberían enfocarse en mejorar la interpretabilidad de los resultados, reducir la latencia en consultas extensas y fortalecer los mecanismos de control de sesgo y trazabilidad. Estas mejoras mejorarían la eficiencia técnica y consolidarían la confiabilidad del sistema en contextos de aplicación profesional y académico.

7. Conclusiones y Trabajos Futuros

El presente Proyecto Integrador Profesionalizante cumplió satisfactoriamente con su objetivo principal de desarrollar un sistema integral de análisis de datos asistido por modelos de lenguaje de gran escala, demostrando la viabilidad técnica y el potencial aplicativo de la integración entre inteligencia artificial generativa, procesamiento de lenguaje natural y análisis automatizado de información estructurada.

7.1 Cumplimiento de objetivos

A lo largo del desarrollo del proyecto se alcanzaron todos los objetivos específicos planteados inicialmente. Se implementó un prototipo funcional que comenzó con una versión en consola hasta evolucionar en una aplicación web completa, integrando LangGraph como núcleo orquestador que gestiona flujos conversacionales complejos mediante un esquema de nodos interconectados con estados compartidos.
El sistema incorpora capacidades robustas de carga, procesamiento y análisis exploratorio de datos, generando tanto visualizaciones gráficas dinámicas como métricas descriptivas relevantes. La evaluación comparativa de distintos modelos de IA generativa llevó a la adopción de una arquitectura híbrida que optimiza precisión, velocidad y eficiencia. La arquitectura cliente-servidor desarrollada con FastAPI y Next.js demostró ser escalable y mantenible, mientras que la implementación de PostgresSaver garantizó la persistencia de memoria conversacional entre sesiones.

7.2 Aportes técnicos y académicos

El proyecto generó contribuciones significativas en múltiples dimensiones técnicas. La implementación del sistema de deduplicación mediante hashing SHA-256 optimizó el uso de recursos, mientras que el mecanismo de fallback automático entre estrategias SQL y DataFrame garantizó robustez operativa. La generación automática de descripciones semánticas de datasets, combinada con un sistema de memoria conversacional que aprende patrones de uso, sitúa al sistema como una herramienta adaptativa que mejora con el tiempo.
Desde la perspectiva académica, el proyecto ejemplifica la aplicación práctica de conceptos avanzados de inteligencia artificial, arquitectura de software y gestión de bases de datos en un contexto integrado. La documentación exhaustiva del proceso

constituye un recurso valioso para futuras investigaciones en sistemas multiagentes basados en LLM.

7.3 Democratización del análisis de datos

Uno de los logros más significativos radica en la eliminación de barreras técnicas para el análisis de información. El sistema permite que usuarios sin conocimientos de programación o herramientas estadísticas especializadas puedan explorar, visualizar e interpretar datos complejos mediante lenguaje natural.
La interfaz intuitiva ofrece retroalimentación inmediata, explicaciones contextualizadas y visualizaciones interpretativas que facilitan la comprensión de patrones y tendencias. La generación automática de código de análisis y visualizaciones, acompañadas de descripciones semánticas, representa un avance sustancial hacia la inclusión digital en el ámbito del análisis de información.

7.4 Limitaciones identificadas

El desarrollo permitió identificar limitaciones que deben considerarse. La restricción de analizar un único archivo por sesión limita análisis comparativos entre múltiples datasets. La dependencia de APIs externas introduce riesgos de disponibilidad, cambios en políticas de uso y costos variables, particularmente con las claves de Groq que requieren actualización periódica.
El sistema está optimizado específicamente para análisis de datos estructurados, priorizando eficiencia sobre capacidades conversacionales generales complejas. Además, carece de mecanismos avanzados de autenticación, gestión de roles o control de acceso granular necesarios para despliegues productivos multiusuario.

7.5 Impacto institucional y proyección

En el contexto de la Universidad Nacional Arturo Jauretche, el proyecto se alinea con los objetivos de promover innovación tecnológica e investigación aplicada. La herramienta tiene potencial de aplicación inmediata en investigación científica, gestión administrativa y docencia.
Más allá del ámbito académico, demuestra aplicabilidad en contextos organizacionales que requieran interpretación rápida de información compleja para toma de decisiones basada en datos.

7.6 Reflexión sobre el proceso formativo

La realización de este PIP representó una oportunidad invaluable para consolidar y ampliar conocimientos técnicos en múltiples áreas:

- Desarrollo web full-stack integrando frontend moderno (Next.js, React) con backend robusto (FastAPI, Python).
- Arquitectura de software modular y escalable con separación clara de responsabilidades.
- Integración de servicios de inteligencia artificial y orquestación de agentes mediante LangGraph.
- Gestión avanzada de bases de datos con PostgreSQL y persistencia de estado conversacional.

A pesar de enfrentarme a tecnologías emergentes como LangGraph y modelos de lenguaje de gran escala, el proceso de adaptación fue facilitado por las bases sólidas adquiridas en materias como Ingeniería de Software, Proyecto de Software, Base de Datos, Metodologías de la programación y Sistemas Distribuidos, las cuales aportaron conceptos fundamentales para el diseño arquitectónico y la estructuración de sistemas complejos.

El enfoque iterativo adoptado, con ciclos continuos de desarrollo, prueba y refinamiento, permitió la identificación temprana de problemas y la implementación de mejoras sustanciales, fortaleciendo capacidades de análisis crítico y resolución de problemas.

Desde el lado no técnico, se desarrollaron habilidades de comunicación técnica, documentación exhaustiva y adaptación ante desafíos imprevistos.
Esta experiencia aportó un valor significativo a mi desarrollo profesional, brindándome herramientas concretas para afrontar proyectos de mayor envergadura en inteligencia artificial aplicada. La realización de este PIP constituirá un antecedente relevante para mi futuro laboral, especialmente en contextos donde la innovación tecnológica y el aprendizaje continuo sean factores determinantes.

7.7 Síntesis final

El sistema EDAI demuestra exitosamente cómo la integración inteligente de modelos de lenguaje, procesamiento de datos y diseño de interfaces genera herramientas accesibles y potentes para el análisis de información. Los resultados

confirman que es posible construir sistemas que combinan sofisticación técnica con simplicidad de uso, manteniendo equilibrio entre rendimiento, precisión y accesibilidad.
La arquitectura modular, la documentación comprehensiva y el enfoque experimental facilitan extensión, adaptación y mejora continua del sistema. Este Proyecto Integrador Profesionalizante cumple satisfactoriamente su propósito de desarrollar una herramienta flexible e intuitiva que integra procesamiento de lenguaje natural, análisis de datos y visualización automatizada, en el que contribuye concretamente a la democratización del acceso al análisis de información mediante tecnologías de inteligencia artificial, y representando tanto un logro técnico significativo como una contribución valiosa al objetivo institucional de fomentar la innovación tecnológica en beneficio de la comunidad educativa y científica.

7.8 Trabajos Futuros

La naturaleza modular y extensible del sistema EDAI abre múltiples líneas de desarrollo futuro que pueden potenciar significativamente sus capacidades funcionales y su aplicabilidad en contextos más amplios. A continuación se detallan las principales mejoras propuestas

7.9 Agente especializado en clasificación de intenciones

Una mejora fundamental consistiría en incorporar un agente dedicado exclusivamente a la identificación y clasificación de la intención del usuario. Este agente actuaría como primera capa de procesamiento, determinando si la consulta se orienta al análisis de datos o corresponde a una pregunta general que puede responderse mediante capacidades conversacionales del LLM sin acceso a datasets. Se podría incluir mediante un nuevo nodo o dentro de uno existente dentro del árbol de nodos.

Esta separación de responsabilidades permitiría ofrecer una experiencia más natural y versátil, mejorando la eficiencia al evitar procesamiento innecesario y optimizando la experiencia del usuario al brindarle respuestas más apropiadas según el tipo de interacción solicitada.

7.10 Consideración del uso de modelos locales como trabajo futuro

Otra línea de evolución relevante consiste en evaluar la integración de modelos locales mediante herramientas como **Ollama**, con el fin de reducir la dependencia de servicios externos y de las restricciones impuestas por las API keys. Esta alternativa permitiría evitar limitaciones como el consumo de tokens, que obliga a contratar planes pagos para continuar operando, o el vencimiento periódico de claves, como ocurre en Groq, que demanda su actualización manual o la incorporación de scripts para automatizar dicha tarea.

Entre los modelos locales más adecuados para tareas de análisis de datos se destacan:

- LLaMA 3.1 8B, eficiente y ligero, ideal para consultas generales y razonamiento moderado con bajos requisitos computacionales.
- Mistral 7B, reconocido por su velocidad y precisión en razonamiento estructurado, útil para análisis descriptivos y consultas técnicas.
- Phi-3 Mini, modelo bien optimizado, con buen rendimiento en tareas de síntesis y análisis ligero sin necesidad de hardware avanzado.

El uso de modelos locales ofrece las siguientes ventajas clave:

- Independencia total de proveedores externos
- Ausencia de costos variables asociados al consumo de tokens
- Operación incluso sin conexión a internet
- Mayor control sobre la privacidad y trazabilidad de los datos.

No obstante, también presenta desventajas como que presentan un menor rendimiento frente a modelos alojados en la nube de gran escala, actualización menos frecuente y necesidad de contar con hardware adecuado para garantizar tiempos de respuesta competitivos.

Aun así, su incorporación en el futuro permitiría ofrecer una alternativa robusta, económica y totalmente autosuficiente para escenarios donde la disponibilidad permanente y la soberanía de los datos sean prioridades.

7.11 Sistema de autenticación y gestión multiusuario

El desarrollo de un sistema completo de autenticación, registro y gestión de usuarios constituye una evolución necesaria para el despliegue en entornos productivos. Esta funcionalidad incluiría mecanismos de registro seguro, autenticación mediante credenciales o métodos externos (OAuth), y gestión de sesiones persistentes. Complementariamente, la implementación de un sistema de múltiples conversaciones por usuario permitiría organizar el trabajo en diferentes contextos o proyectos. Cada usuario podría crear, nombrar y gestionar múltiples hilos conversacionales independientes, cada uno con su propio historial, contexto y datasets asociados, facilitando la organización del trabajo analítico y permitiendo retomar análisis previos sin pérdida de contexto.

7.12 Análisis comparativo multidocumento

Una extensión funcional de alto valor consistiría en permitir el análisis comparativo simultáneo de múltiples datasets. Esta capacidad habilitaría al usuario para generar visualizaciones que contrasten información de diferentes fuentes, identificar correlaciones entre conjuntos de datos relacionados, y producir análisis integrados que actualmente no son posibles con la restricción de un documento por sesión. Esta funcionalidad permitiría, por ejemplo, comparar ventas de diferentes períodos, analizar datos de múltiples regiones geográficas, o contrastar resultados experimentales de distintas condiciones, ampliando significativamente el espectro de casos de uso del sistema.

8. Bibliografía

[1] Albrecht, S. V., Christianos, F. y Schäfer, L. (2024). Multi-Agent Reinforcement Learning: Foundations and Modern Approaches. MIT Press. https://www.marl-book.com/

[2] Albada, M. (2025). Building Applications with AI Agents. O'Reilly Media.

[3] LangChain AI. (2024). LangGraph Documentation. https://www.marl-book.com/

[4] LangChain Academy. (2024). Introduction to LangGraph Course.
https://www.langchain.com/langgraph

[5] DataCamp. (2024). "LangGraph Tutorial: What Is LangGraph and How to Use It?".
https://www.datacamp.com/tutorial/langgraph-tutorial

[6] LangChain vs LangGraph: A Developer's Guide to Choosing Your AI Frameworks Milvus
https://milvus.io/blog/langchain-vs-langgraph.md

[7] Google AI. (2024). Gemini 2.0 Flash Documentation.
https://ai.google.dev/gemini-api/docs

[8] Groq. (2024). Groq API Documentation.
https://console.groq.com/docs/overview

[9] Clean Code - Robert Cecil Martin. Prentice Hall.

[10] Clean Architecture - Robert Cecil Martin. Prentice Hall.

[11] Introducción a las “Clean Architectures” Medium
https://medium.com/@diego.coder/introducci%C3%B3n-a-las-clean-architectures-723fe9fe17fa

[12] FastAPI Community. (2024). FastAPI Tutorial - Guía del Usuario.
https://fastapi.tiangolo.com/es/tutorial/

[13] What is v0?
https://v0.app/docs/introduction

[14] TypeScript Documentation
https://www.typescriptlang.org/docs/

[15] Youtube (2023) “Crea una API con Python en menos de 5 minutos (Fast API)”
https://www.youtube.com/watch?v=J0y2tjBz2Ao

[16] Youtube (2020) “Cómo usar MATPLOTLIB para hacer GRAFICAS 🐍💻 [Curso Python Data Science Español]”
https://www.youtube.com/watch?v=XEG4eh5l_qU

[17] Youtube (2025) “LangChain vs LangGraph – ¿Cuál deberías usar en tus proyectos de IA?”
https://www.youtube.com/watch?v=WzHEYvnkW8Y

[18] Youtube (2025) “LangGraph - el paradigma de desarrollo propuesto por LangChain”
https://www.youtube.com/watch?v=VnUeugXduCE

9. Anexos

Anexo 1 - Pantalla principal de chat (Desktop)

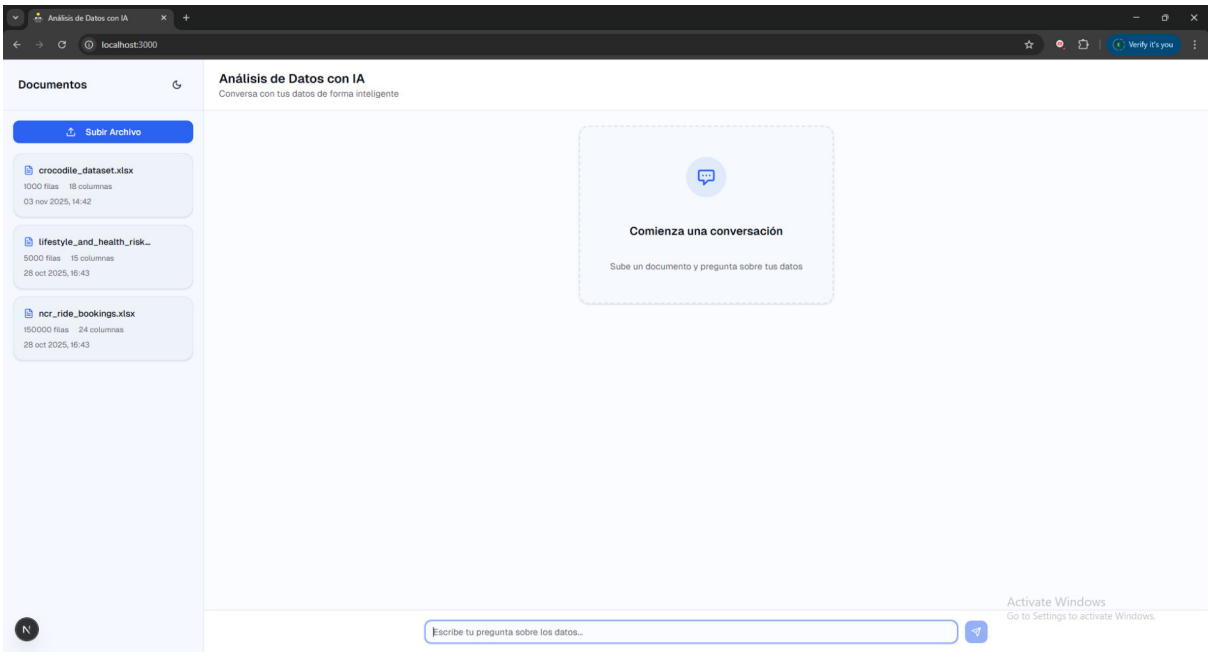


Figura 1 - Pantalla principal de chat con tema claro (Desktop)

La Figura 1 presenta la pantalla de chat principal del sistema en donde el usuario puede ver el menú lateral donde tiene la posibilidad de subir archivos con el botón “Subir Archivo”, puede visualizar los documentos subidos al sistema asi como también puede eliminar cada uno con el botón de ícono que aparece al posicionar el mouse por encima de cada archivo y también tiene un botón con ícono de luna que permite cambiar al modo oscuro. Por otro lado, en el centro de la pantalla se encuentra la interfaz principal de chat con la IA en donde en la parte inferior se

encuentra una barra para ingresar la consulta junto con un botón para enviar el mensaje y por encima se tiene el espacio donde se verán los mensajes del usuario y la IA.

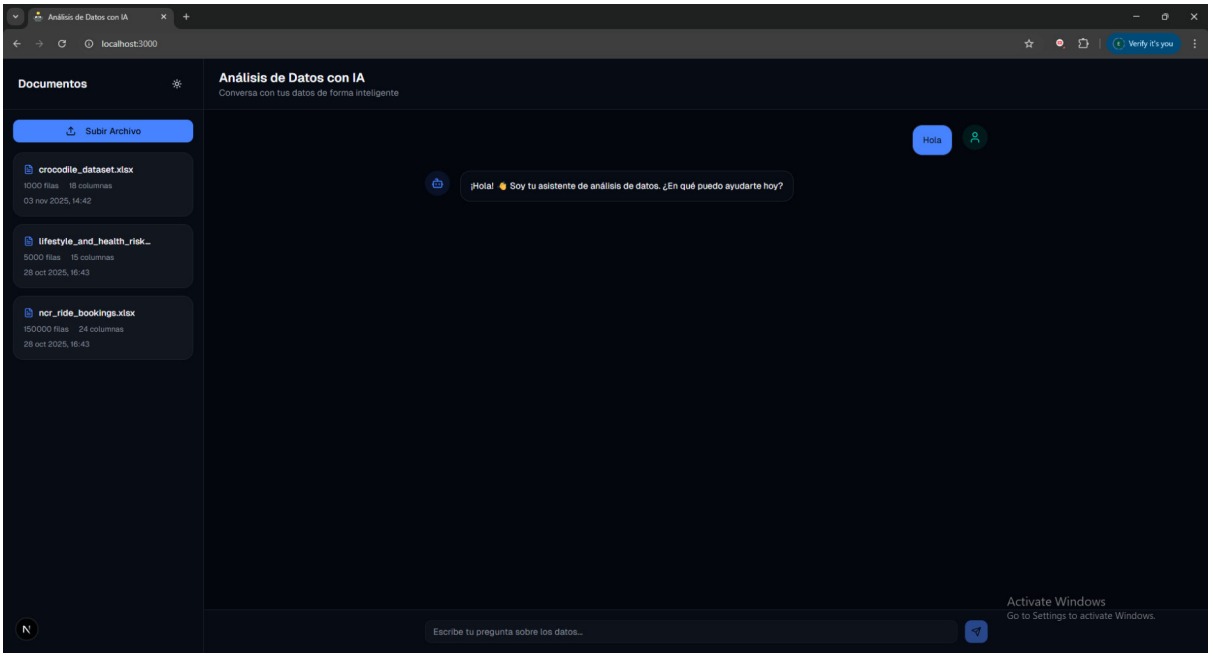


Figura 2 - Pantalla principal de chat con tema oscuro (Desktop)

La Figura 2 presenta también la pantalla principal de chat pero con el tema oscuro activo. Las funcionalidades son las mismas que las que ofrece el modo claro, el único cambio es el tema de visualización. Además se envió un saludo a la IA, en el cual responde y podemos ver cómo se visualizan los chats en forma de “burbuja”.

Anexo 2 - Pantalla principal de chat (Tablet)

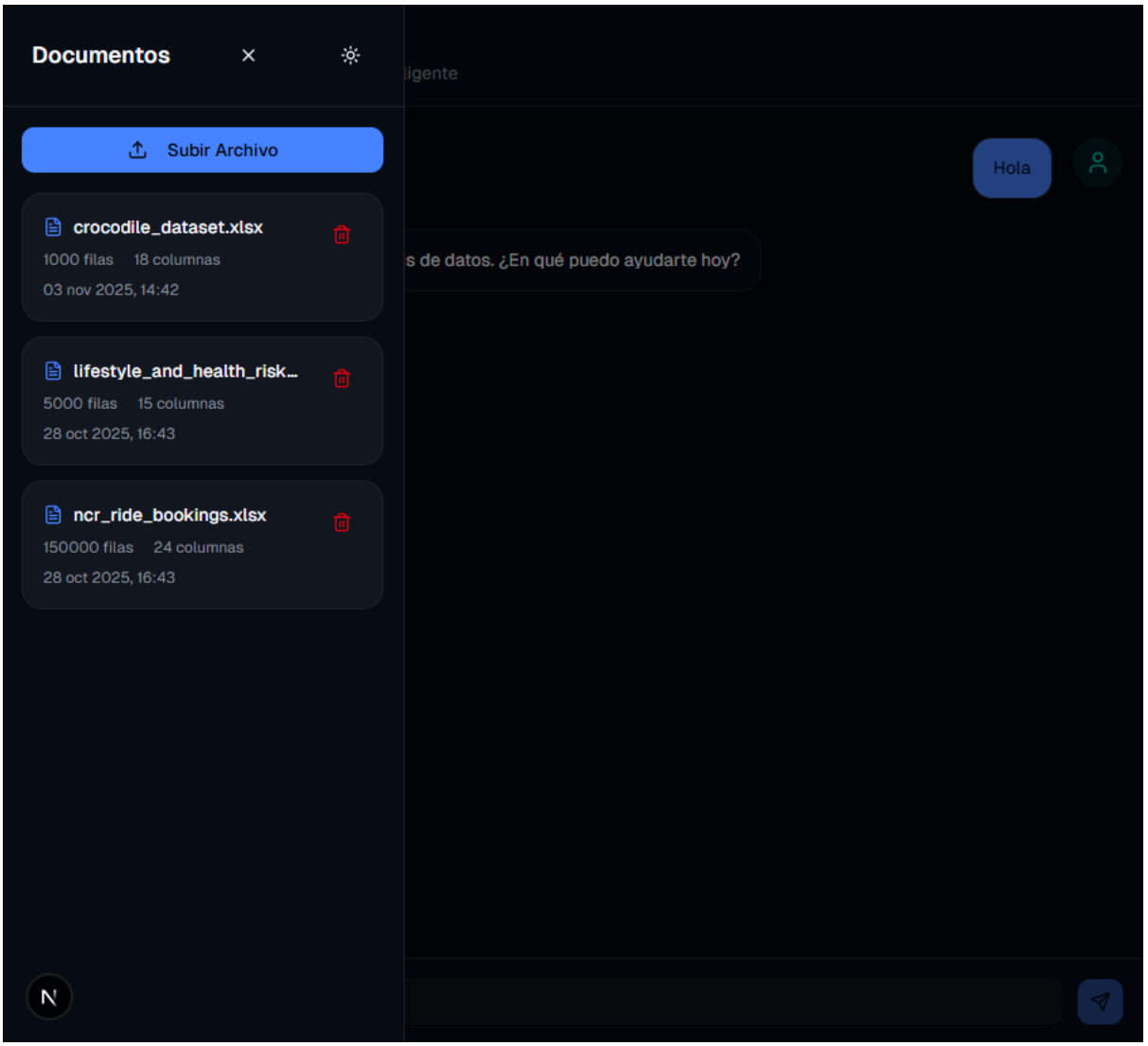


Figura 3 - Pantalla principal de chat (Tablet)

La Figura 3 presenta la pantalla principal de chat para pantallas de tamaño tablet. Como se observa, el menú lateral ahora se contrae y hay que hacer click en un botón de hamburguesa para que aparezca. Además podemos observar que el botón para eliminar el documento está siempre visible para que sea fácil de usar para el usuario. Su diseño se adapta al formato vertical, optimizando la usabilidad en pantallas intermedias.

Anexo 3 - Pantalla principal de chat (Mobile)

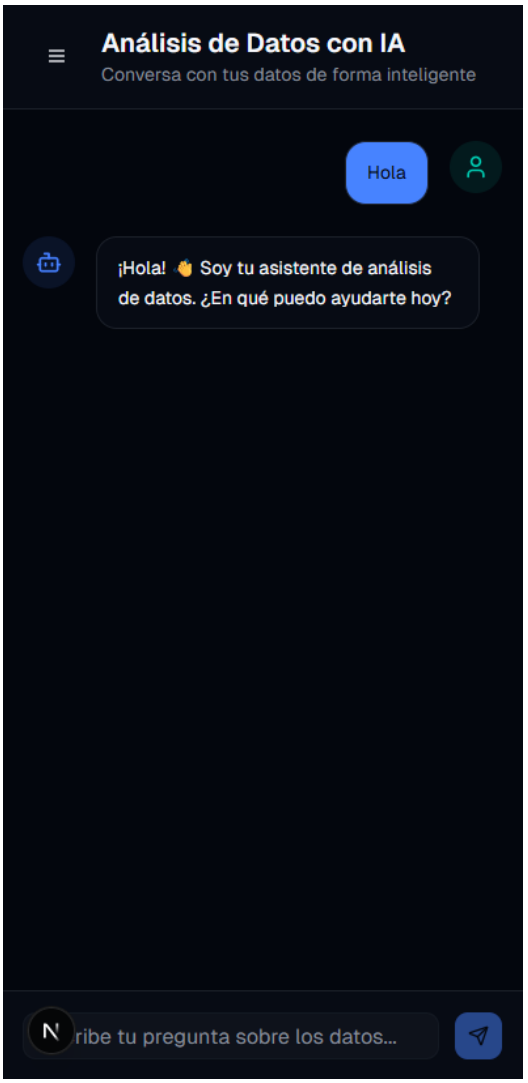


Figura 4 - Pantalla principal de chat (Mobile)

La Figura 4 presenta la pantalla principal de chat para pantallas de tamaño mobile. Como se observa, el menú lateral ahora se contrae y hay que hacer click en un botón de hamburguesa para que aparezca. Su diseño se adapta al formato vertical, optimizando la usabilidad en pantallas pequeñas.

Anexo 4 - Generación de gráfico

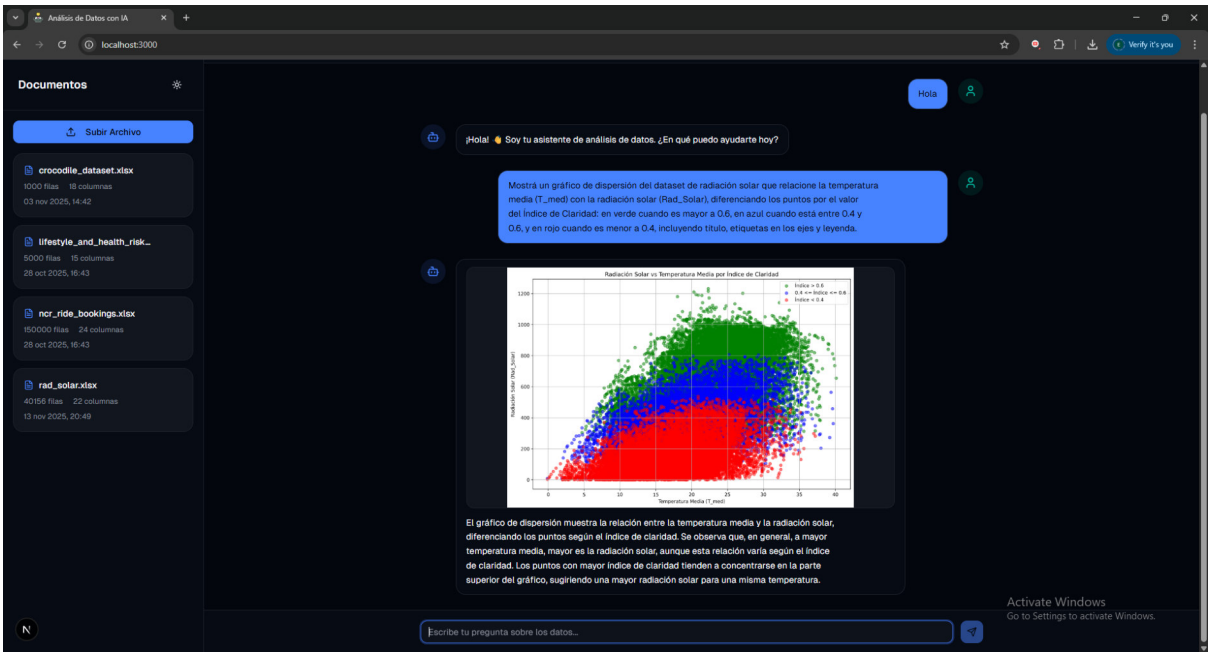


Figura 5 - Generación de gráfico

La Figura 5 presenta un gráfico generado por la IA. Podemos observar que primero muestra el gráfico y luego una breve descripción del mismo que permite al usuario entender que muestra el gráfico y que tendencias tiene para poder realizar un analisis.

Anexo 5 - Zoom en gráfico generado

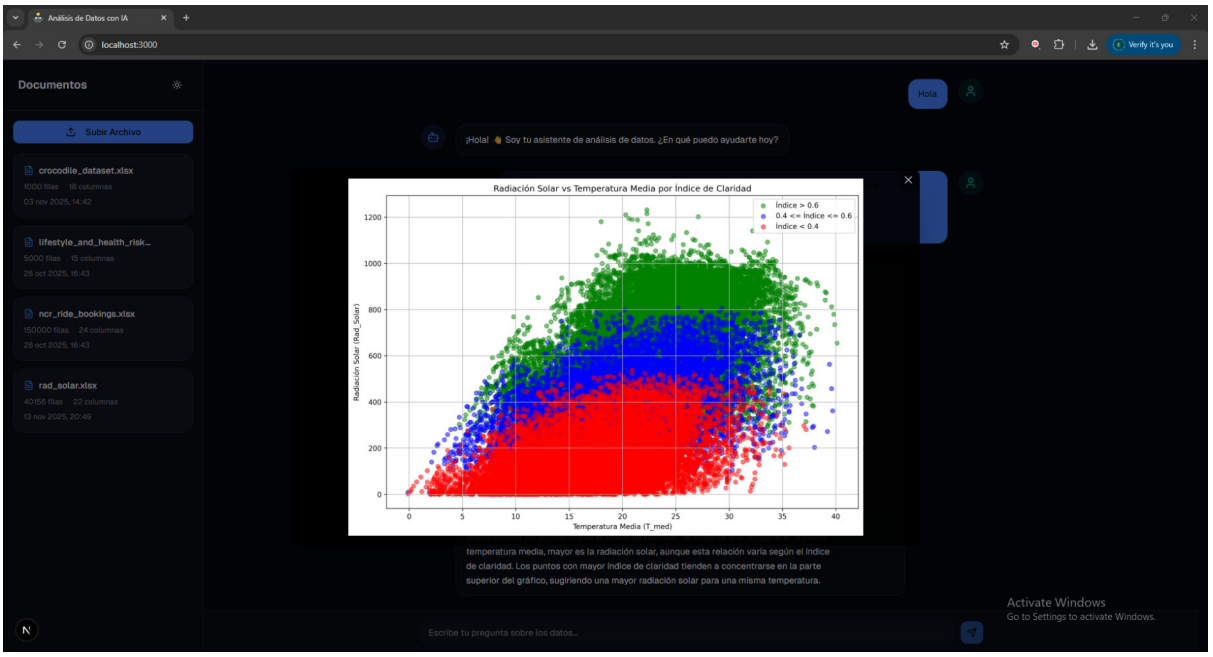


Figura 6 - Generación de gráfico

La Figura 6 presenta el zoom que se le puede hacer al gráfico generado por la IA en el anexo anterior. Para hacerle zoom, hay que hacer click en la imagen generada.

Anexo 6 - Guía de Uso del Sistema

6.1 Requisitos previos

- Contar con un archivo CSV o Excel para analizar.
- El sistema fue probado con datasets de hasta 150.000 filas y 24 columnas; archivos más grandes pueden tardar más en cargarse y procesarse.
- Se pueden cargar múltiples datasets, uno por vez.

6.2 Carga de datos

1. Hacer clic en “Subir Archivo”.
2. Seleccionar el archivo desde el equipo.
3. Esperar el mensaje “Archivo subido correctamente”.
4. A partir de ese momento, ya se pueden realizar consultas.

Nota: No cerrar ni refrescar la página durante la carga, especialmente con archivos grandes.

6.3 Cómo hacer consultas

El sistema admite:

a) Interacción general

Saludar, pedir ayuda, consultar qué funciones tiene o cómo usarlo.

b) Consultas sobre datasets

- Obtener columnas
- Ver registros
- Resúmenes descriptivos
- Estadísticas (promedios, conteos, máximos, mínimos)
- Comparaciones
- Filtrados
- Pedir gráficos

Recomendación: Siempre indicar qué dataset se quiere analizar (por nombre, palabra clave o por orden de carga: “primer dataset”, “segundo dataset”).

c) Generación de gráficos

Se puede solicitar:

- Histogramas
- Dispersión
- Mapas de calor
- Clustermaps
- Gráfico circular
- Gráficos con condiciones o colores particulares

Nota. la IA puede realizar cualquier tipo de gráficos, siempre y cuando se le indique correctamente que debe graficar y con qué datos.

6.4 Ejemplos

Consultas básicas

- hola

- ¿qué puedes hacer?
- gracias

Sobre datasets

- ¿Cuáles son las columnas del dataset de cocodrilos?
- Dame los primeros 5 registros del dataset de viajes.
- Promedio de Booking Value para pagos con UPI.
- ¿Cuántos cocodrilos pesan más de 100 kg?

Gráficos

- Genera un histograma de Booking Value.
- Gráfico de dispersión entre num_hora y Rad_Solar.
- Mostrar clustermap de correlaciones del dataset de radiación solar.
- Dispersión de Rad_Solar vs num_dia con colores según Índice de Claridad.

6.5 Consideraciones importantes

- Análisis de un solo archivo por sesión: El sistema actualmente no permite comparar múltiples datasets entre sí. Cada consulta debe estar asociada únicamente al dataset que está activo en la sesión.
- Almacenamiento de gráficos generados: Todos los gráficos creados por el sistema se guardan automáticamente en formato .png dentro de la carpeta local /outputs del proyecto. Cada archivo se genera con un nombre único, lo que permite recuperarlos posteriormente sin sobrescribir imágenes previas.