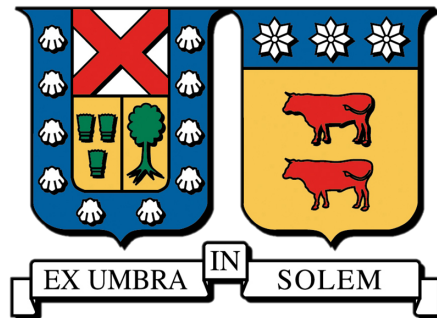


UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA
DEPARTAMENTO DE ELECTRÓNICA
VALPARAÍSO - CHILE



“Aplicación impulsada por modelos de lenguaje de gran tamaño para el diseño de consultas SQL mediante ingeniería de prompt.”

Vicente Pablo Alvear Valderrama

**MEMORIA DE TITULACIÓN PARA OPTAR AL TÍTULO DE INGENIERO
CIVIL TELEMÁTICO**

PROFESOR GUÍA:

Nicolas Jara C

PROFESOR CORREFERENTE:

Mauricio Araya L

Julio 2025



CONSTANCIA DE VALIDACIÓN Y CONFIDENCIALIDAD DE MONOGRAFÍA A REPOSITORIO ACADÉMICO

1.- IDENTIFICACIÓN DEL TRABAJO ACADÉMICO

Tipo de monografía (marcar una opción): ☒ Memoria o trabajo de título; ☐ Tesis de Postgrado;

Título del trabajo: Aplicación impulsada por modelos de lenguaje de gran tamaño para el diseño de consultas SQL mediante ingeniería de prompt.

Nombre del candidato(a): Vicente Pablo Alvear Valderrama

Carrera / Grado: Ingeniería Civil Telemática

Campus: Casa Central Valparaíso ; **Departamento:** Electrónica

2.- VALIDACIÓN DEL PROFESOR GUÍA/DIRECTOR DE TESIS

Yo, Nicolás Jara Carvallo, en mi calidad de profesor(a) guía/director(a) del trabajo académico mencionado anteriormente **DEJO CONSTANCIA** que:

- He revisado esta versión del documento y corresponde a la versión final aprobada del trabajo.
- El trabajo cumple con los requisitos académicos y de formato establecidos por la institución

3.- EVALUACIÓN DE CONFIDENCIALIDAD POR PROPIEDAD INDUSTRIAL

☒ El trabajo **NO contiene información que amerite confidencialidad** y puede ser publicado de inmediato en repositorio con acceso abierto.

☐ El trabajo **CONTIENE** información con potenciales implicancias de propiedad industrial o intelectual y requiere un periodo de confidencialidad (embargo) por:

☐ 6 meses; ☐ 12 meses; ☐ 2 años; ☐ 3 años; ☐ 5 años; ☐ 10 años

Fundamentación de la necesidad de confidencialidad (obligatorio si se solicita embargo):

4.- FIRMAS

Profesor(a) guía o director(a) de memoria o tesis:

Fecha: _____ ; Firma: _____

Estudiante o Candidato(a):

Fecha: 03/10/2025 ; Firma: _____

Este formulario debe ser insertado como página 2 de la memoria o tesis, completado y firmado por estudiante y profesor(a) antes de la entrega en portal PRISMA de Biblioteca USM.

Agradecimientos

Gracias a toda la gente que ha sido parte del proceso que ha sido sacar mi carrera, desde familia, pareja, amistades de la infancia y amistades que formé durante esta época. Gracias a ellos por el constante apoyo e influenciar el como soy hoy en día. Todos ustedes hicieron de esta experiencia, algo mucho mas llevadero.

Gracias también a los que ya no están terrenalmente, pero que imagino, de alguna manera, estuvieron viendo este dificultoso capítulo, y que imagino seguirán ahí para los que vienen.

Resumen

El siguiente informe se enfoca en el ideado e implementación de una plataforma digital segura cuyo objetivo principal es democratizar el acceso a los datos en una base de datos de un equipo de trabajo. Esto se logra con la ayuda de los modelos de lenguaje de gran tamaño, los cuales actúan como herramienta para evitar la necesidad de tener conocimiento sobre el confeccionado de consultas SQL. El proceso consta de una serie de modelos comunicados entre sí, que generan salidas dependiendo del mensaje que el usuario envíe como entrada.

Durante el desarrollo del proyecto, se mantuvo un enfoque constante en el usuario, priorizando que la plataforma fuera intuitiva y cómoda de usar, asegurando una experiencia accesible para personas sin conocimientos técnicos. Además, se contemplaron múltiples casos para manejar situaciones en las que las preguntas no estén necesariamente relacionadas con los datos, o cuando se intente hacer un mal uso de la plataforma, como la solicitud de modificaciones de información, preservando así la integridad y seguridad del sistema.

El informe refleja el proceso de desarrollo completo, abarcando una investigación inicial, la definición de objetivos y requisitos funcionales, la especificación de los módulos e interfaces que componen el sistema y, finalmente, el desarrollo junto a los resultados obtenidos.

Keywords: Modelos de lenguaje de gran tamaño (LLM), Claude, Prompt Engineering, text-to-SQL

Glosario

LLM: Un sistema computacional diseñado para procesar y generar texto basado en datos de entrenamiento.

Token: Es una unidad básica de un texto, que puede ser una palabra, un número, un signo de puntuación o incluso una parte de una palabra.

Tokenización: El proceso de dividir un texto en unidades más pequeñas, como palabras o subpalabras.

Embedding models: Representaciones numéricas de palabras o conceptos que capturan sus significados en un espacio vectorial.

RAG: Un sistema de recuperación-aumentación-generación diseñado para responder preguntas combinando información recuperada de una base de conocimiento con modelos generativos.

Prompt Engineering: El proceso de diseñar instrucciones claras y específicas para guiar el comportamiento de un modelo de lenguaje en una tarea particular.

Backend: La parte del sistema encargada de procesar datos, gestionar la lógica del negocio y comunicarse con las bases de datos y servicios.

Frontend: La interfaz gráfica de usuario de una aplicación o sistema, diseñada para permitir la interacción directa con el usuario.

API: Una interfaz de programación de aplicaciones que permite la comunicación entre diferentes sistemas o componentes de software.

ORM: Un sistema que permite mapear objetos de una aplicación a tablas de bases de datos relacionales, simplificando las operaciones de manipulación de datos.

Fine Tuning: El proceso de ajustar un modelo preentrenado en un conjunto de datos específico para mejorar su rendimiento en tareas concretas.

Base de Datos Relacional: Un tipo de base de datos que organiza la información en tablas interrelacionadas mediante estructuras de datos como SQL.

Similitud de cosenos: Una métrica para medir la similitud entre dos vectores en un espacio vectorial, utilizada para comparar embeddings.

Acrónimos

LLM: Large Language Model

RAG: Retrieval Augmented Generation

NL: Lenguaje Natural

SQL: Structured query language

API: Application Programming Interface

ORM: Object Relational Mapping

Índice de figuras

5.1. Diagrama de contexto	30
5.2. Diagrama de arquitectura	31
6.1. Modelo de navegación	38
6.2. Inicio de sesión	39
6.3. Menú principal	39
6.4. Registros de conversaciones	39
6.5. Métricas de satisfacción	39
6.6. Consultar datos	39
6.7. Inicio de sesión	40
6.8. Menú principal - chatbot	40
6.9. Registros de conversaciones	41
6.10. Métricas de satisfacción	41
7.1. Vista principal con barra lateral y sección de administrador	45
7.2. Ejemplo de estructura de prompt	47
7.3. Prompt del identificador general	56
7.4. Prueba unitaria caso conversación	57
7.5. Prueba unitaria caso consulta SQL	57
7.6. Prueba unitaria caso opción	57
7.7. Prueba unitaria caso gráfico	57
7.8. Prompt experto SQL	58

7.9. Documentos RAG	59
7.10. Prueba unitaria caso consulta SQL	60
7.11. Prueba unitaria caso consulta SQL no relacionada la base de datos	60
7.12. Prompt del identificador NL/SQL	60
7.13. Prueba unitaria caso SQL	61
7.14. Prueba unitaria caso NL	61
7.15. Prompt del corrector SQL	62
7.16. Prueba unitaria para caso de nuevo SQL	62
7.17. Prueba unitaria para caso donde no hay información en la base de datos	62
7.18. Prompt del traductor data a NL	63
7.19. Prueba unitaria para caso menor a 500 tokens	64
7.20. Prueba unitaria para caso mayor a 500 tokens	64
7.21. Prueba unitaria para cuando no se encuentra información en la base de datos	64
7.22. Prompt del recolector	66
7.23. Prueba unitaria solicitud de gráfico con información	67
7.24. Gráfico generado a partir de información recolectada	67
7.25. Solicitud de nuevo gráfico	67
7.26. Nuevo gráfico generado a partir de la ultima solicitud y la información previa	68
7.27. Prueba unitaria solicitud de gráfico sin información completa	68
7.28. Salida del recolector con parámetro de falta de datos asignado como True, indicando la falta de información para generar el gráfico	69
7.29. Gráfico generado a partir de la nueva información recolectada	69
7.30. Prueba unitaria para Experto SQL-Gráfico donde pese a petición de gráfico, se genera una consulta SQL	70
7.31. Función generador grafico completa	71
7.32. Diagrama de flujo prototipo	72
7.33. Mensaje recibido en la plataforma al iniciar una nueva conversación	78

7.34. Clasificación del mensaje como SQL por el identificador general	79
7.35. Registro del mensaje del usuario en la base de datos	79
7.36. Consulta SQL generada por el modelo experto	79
7.37. Clasificación de la consulta como SQL por el identificador SQL/NL . .	80
7.38. Consulta SQL almacenada correctamente en la base de datos	80
7.39. Resultados obtenidos tras ejecutar la consulta	80
7.40. Respuesta en lenguaje natural generada por el modelo traductor	81
7.41. Respuesta almacenada como tipo response en la base de datos	81
7.42. Vista de la plataforma al iniciar el caso de uso 2	82
7.43. Clasificación del mensaje como graph	82
7.44. Registro del mensaje como tipo conversación	83
7.45. Archivo más reciente encontrado en la conversación	83
7.46. Validación del archivo por el modelo recolector mediante estructura	
JSON	84
7.47. Ejecución del generador gráfico y almacenamiento de salida	84
7.48. Mensaje al usuario con confirmación de gráfico generado	85
7.49. Opción de descarga del gráfico en formato PNG	85
7.50. Vista de la plataforma al recibir un mensaje en una conversación existente	86
7.51. Clasificación del mensaje como xlsx por el identificador general . . .	87
7.52. Registro de la opción seleccionada en la base de datos	87
7.53. Consulta previa recuperada desde la conversación	88
7.54. Confirmación de la clasificación final del mensaje como xlsx	88
7.55. Mensaje de confirmación de creación del archivo	89
7.56. Interfaz con opción de descarga del archivo generado	89
7.57. Registro del archivo generado como tipo file en la base de datos . . .	90
7.58. Camino para los distintos casos de uso reflejado en el diagrama de flujo	91

Índice de cuadros

3.1. Cuadro comparativo entre enfoques de Text-to-SQL en literatura re-	
ciente y esta propuesta	21
4.1. Eventos externos	25
4.2. Respuestas del sistema	26
4.3. Perfiles de usuario	27
4.4. Objetivos específicos estructurados	28
5.2. Módulos y definiciones	32
6.1. Eventos prototipo y definiciones	42
7.2. Comparativa modelos familia Claude	73
7.3. Costos por modelo	74
7.4. Tasa de falsos negativos por modelo.	75
7.5. Tasa de falsos positivos por modelo.	76
7.6. Tasa de tiempos de ejecución por modelo.	76
7.7. Costos cada 1000 tokens por modelo.	76
9.1. Riesgos y respectivas medidas de mitigación	98

Índice general

1. Introducción	7
1.1. Objetivos	8
1.1.1. Objetivo General	8
1.1.2. Objetivos Específicos	8
1.2. Estructura	9
2. Marco Teórico	11
3. Estado del Arte	13
4. Requisitos del sistema	23
4.1. Requisitos	23
4.1.1. Requisitos funcionales	23
4.1.2. Requisitos no funcionales	24
4.1.3. Requisitos de Interfaces	25
4.1.4. Requisitos de Ambiente	26
4.1.5. Perfiles de usuarios	27
4.1.6. Planificación del Proyecto	28
5. Diseño de arquitectura del sistema	29
5.1. Diagrama de contexto	29
5.2. Diagrama de arquitectura	30
5.3. Enumeración de módulos	31

5.4. Definición de módulos	33
5.4.1. Frontend	33
5.4.2. Backend	33
5.4.3. API	33
5.4.4. Modelos de lenguaje / Amazon Bedrock (LLMs)	34
5.4.5. Base de datos (BBDD)	35
6. Diseño de interfaces	37
6.1. Modelo de navegación	37
6.2. Prototipo de interfaces usuarias	38
6.2.1. Prototipo 1.1	38
6.2.2. Prototipo 1.2	39
7. Desarrollo y Resultados de la solución	43
7.1. Hitos principales del proyecto	43
7.1.1. Revisión de requerimientos	44
7.1.2. Revisión del diseño	45
7.1.3. Revisión de Modelos	46
7.1.4. Estructura de prompts	46
7.1.5. Implementación de memoria en los modelos	47
7.1.6. RAG	48
7.1.7. Revisión del prototipo	49
7.2. Desarrollo y Resultados de la solución enfocado a modelos	55
7.2.1. Flujo principal	55
7.2.2. Flujo de gráficos	65
7.2.3. Limitaciones actuales del flujo	71
7.2.4. Diagrama de flujo	71
7.2.5. Comparativa de modelos de lenguaje	72
7.2.6. Comparativa de modelos de embeddings	75
7.3. Casos de uso	78

7.3.1. Caso 1: Petición de consulta SQL a la plataforma con informa-	
ción que si se encuentra en la base de datos	78
7.3.2. Caso 2: Petición de gráfico con información que fue obtenida	
en la base de datos	82
7.3.3. Caso 3: Petición de generación de archivo en formato xlsx . . .	86
8. Conclusiones	93
9. Trabajo a futuro	95
9.1. Gestión de riesgos	97
9.1.1. Supuestos	97
9.1.2. Dependencias	97
9.1.3. Restricciones	98
9.1.4. Riesgos	98
9.2. Verificación y validación	98
9.2.1. Verificación	98
9.2.2. Validación	99
9.2.3. Plan de testing	99

Capítulo 1

Introducción

Con la llegada de la inteligencia artificial y los modelos de lenguaje de gran tamaño, las personas están en constante búsqueda de como implementar estas herramientas para facilitar y hacer mas eficientes los procesos en sus vidas, trabajos, estudios etc.

En la actualidad, las organizaciones generan y almacenan volúmenes masivos de datos, lo que ha llevado a un creciente interés por herramientas que permitan acceder y aprovechar esta información de manera efectiva. Sin embargo, gran parte del personal en equipos de trabajo carece del conocimiento técnico necesario para realizar consultas directas a bases de datos, como es el caso de consultas en SQL, dificultando la extracción de información valiosa para la toma de decisiones. En este contexto, los avances en modelos de lenguaje de gran tamaño (LLM, por sus siglas en inglés) han abierto la posibilidad de intentar disminuir o incluso eliminar brechas que existían a la hora de acceder a los datos, siendo el caso de lo que plantearemos poco más adelante, una alternativa eficiente y de acceso para todos los autorizados.

A pesar de los avances en la inteligencia artificial y las interfaces conversacionales, aún persiste una brecha significativa entre el potencial de estas tecnologías y su aplicación práctica en el acceso democratizado a los datos. Muchas plataformas existentes están diseñadas para usuarios técnicos, lo que limita la capacidad de equipos multidisciplinarios para interactuar directamente con las bases de datos sin intermediarios.

Este desafío destaca la necesidad de herramientas intuitivas que traduzcan el lenguaje natural a consultas SQL, eliminando barreras técnicas y potenciando la autonomía de los equipos de trabajo.

La presente investigación propone el diseño e implementación de una plataforma digital que incorpora un chatbot basado en modelos LLM, el cual permite a los usuarios realizar consultas en lenguaje natural que se traducen automáticamente a consultas SQL. Este enfoque busca democratizar el acceso a los datos, facilitando la interacción con bases de datos para personal no técnico y promoviendo una mayor eficiencia en la toma de decisiones dentro de equipos de trabajo. A través de esta solución, se exploran las capacidades de los modelos LLM como herramientas prácticas para transformar la interacción con los datos en diversos entornos organizacionales. Toda esta investigación, proceso y desarrollo es trabajado de la mano junto con FALP como cliente (Fundación Arturo López Pérez), donde el equipo de trabajo está en constante comunicación con el equipo del área de tecnología de la fundación.

1.1. Objetivos

1.1.1. Objetivo General

El objetivo de este trabajo consiste en facilitar el acceso a datos en la Fundación por medio de una plataforma intuitiva e impulsada por el uso de modelos de lenguaje LLM de la familia Claude

1.1.2. Objetivos Específicos

1. La LLM debe traducir las consultas en lenguaje natural a lenguaje SQL.
2. Elaborar una API.
3. El sistema debe administrar las métricas de satisfacción del usuario respecto a su interacción con la LLM

4. El sistema debe administrar el registro de conversaciones de los usuarios con la LLM.
5. El sistema debe ser accesible mediante una plataforma.

1.2. Estructura

Este documento se estructura de la siguiente forma: En el capítulo 2 se presenta un marco teórico el cual explica conceptos que son claves para poder entender el contexto y proceso del proyecto, en el capítulo 3 se muestra el estado del arte donde se muestran trabajos similares y que son de utilidad tener en mente para las distintas técnicas y métodos para poder resolver la problemática, dando también apertura a distintos caminos que se pueden tomar para robustecer y hacer aún más rica la propuesta. El capítulo 4 plantea los requisitos del sistema, desde los requisitos funcionales y no funcionales hasta los requisitos de interfaces, ambiente y los distintos perfiles de usuario que comprenden la propuesta. En el capítulo 5 se plantea la arquitectura del sistema, identificando y definiendo principalmente los módulos que componen la propuesta. El capítulo 6 muestra todo el diseño de interfaces de la plataforma, mostrando el diagrama de flujo de esta, los distintos prototipos iniciales y finalmente las interacciones del sistema. En el capítulo 7 se muestra el desarrollo y resultados de la solución, en donde se observa el diseño final de la plataforma y los distintos flujos impulsados con modelos LLM de la familia Claude para poder realizar consultas a la base de datos. En el capítulo 8 se presentan las conclusiones del proyecto donde se resumen los logros. Finalmente, en el capítulo 9 se muestra el trabajo a futuro, donde se explican cuáles son las siguientes características por implementar para hacer de la aplicación un sistema más robusto y completo. Finalmente contamos con una sección de anexos de información de la gestión de riesgos y el proceso de verificación y validación de las distintas partes de la propuesta.

Capítulo 2

Marco Teórico

El crecimiento de la inteligencia artificial en estos años ha permitido realizar tareas que antes requerían de pensamiento e inteligencia humana. Dentro de este campo, el procesamiento del lenguaje natural (NLP, por sus siglas en inglés) se centra en la interacción entre computadoras y lenguaje humano, permitiendo que los sistemas comprendan, interpreten y generen texto en lenguaje natural.

Al mismo tiempo, los modelos de lenguaje de gran tamaño (Large Language Models o LLM por sus siglas) como GPT, Claude o Llama se entrenan con grandes volúmenes de texto para aprender patrones lingüísticos complejos. Estos modelos tienen la capacidad de cumplir con diversas tareas, desde responder de manera coherente, traducir idiomas, generar resúmenes y realizar tareas específicas como generación de código o traducir de lenguaje natural a un lenguaje estructurado, estos dos puntos finales son el foco y utilidad principal para la realización del proyecto.

Una de las tareas que suele aparecer en entornos donde haya manejo de datos, sobre todo en grandes volúmenes, es la necesidad de realizar consultas (usualmente en lenguaje SQL). Sin embargo, en muchos casos, esta necesidad surge de usuarios no técnicos que no tienen conocimiento de como realizar una consulta y se encuentra limitados para poder obtener la información de manera directa, por lo que tienen que acudir a alguien especializado para así obtener la información que desean. De lo anterior nace

el concepto NL2SQL (Natural Language to SQL) el cual busca abordar este problema mediante la traducción automática de preguntas en lenguaje natural a consultas SQL válidas. Esta tarea requiere de comprender la intención del usuario como entender a cabalidad la estructura de la base de datos, lo que termina resultando en una tarea muchas veces compleja.

Finalmente, se busca la democratización de los datos, lo que implica que el acceso y uso de la información disponible en una organización esté al alcance de todos sus miembros, independientemente de su nivel técnico. Para lograr esto, las interfaces conversacionales —comúnmente implementadas como chatbots, han evolucionado desde sistemas basados en reglas hacia asistentes impulsados por IA. En el contexto actual, el uso de modelos LLM permite que estos asistentes comprendan y respondan con una naturalidad y precisión que sistemas tradicionales, como Botpress o RASA, no podían ofrecer. De hecho, dichas plataformas han comenzado a incorporar LLMs en su propia arquitectura, reconociendo esta superioridad tecnológica.

La propuesta busca diseñar una solución que se adapte al contexto de la Fundación Arturo López Pérez (FALP) y sus necesidades específicas, con un enfoque en la usabilidad, la precisión de las consultas generadas y la integración en flujos reales de trabajo. Todo esto por medio de prompt engineering lo que permite la flexibilidad a la hora de querer implementar este sistema en otros contextos, ya que basta con adaptar el prompt según la base de datos y las necesidades del cliente.

Capítulo 3

Estado del Arte

En [1], se crea un sistema con GUI y manejo de errores, donde el usuario puede ver los contenidos de una tabla, ingresar una query en sql o en lenguaje natural. El usuario debe iniciar sesión en el sistema, puede hacer preguntas que involucren la obtención, ingreso, actualización y eliminación de datos. La frase ingresada por el usuario pasará por una tokenización, lo que significa que se separará la frase en palabras y cada una representa un token, luego pasa al análisis léxico, donde la lista tokenizada se mapea con un diccionario, por lo que estas palabras son reemplazadas por las palabras del diccionario relacionadas a la base de datos, luego se analiza sintácticamente, donde cada palabra tokenizada se mapea con atributos de otro diccionario, y por último se pasa por un análisis semántico, donde el sistema encontrará palabras que representen condiciones o símbolos y esa palabra se mapeará con el diccionario. El resultado es una consulta que luego puede ser ejecutada en la base de datos.

Se utilizan distintas estrategias de ingeniería de prompt, en [2] se estudian empíricamente varias LLMs de código abierto para destacar el potencial de estas en in-context learning y supervised fine-tuning con varias representaciones. Se demuestra que similar al aprendizaje in-context, la estrategia de representación es crítica para supervised fine-tuning. También luego de fine-tuning se observa una disminución en las capacidades de aprendizaje in-context. La solución integrada propuesta por el paper, DAIL-SQL,

tiene un 86.6 % de exactitud en la ejecución y tiene el primer lugar en el spider leaderboard.

En [3] se plantea que la capacidad de comprensión de un LLM no es suficiente cuando se le proporciona únicamente el esquema de la base de datos y la consulta que se desea realizar, ya que el resultado tiende a ser impreciso. Por esta razón, se propone un framework llamado Knowledge-to-SQL, que utiliza un LLM experto en datos (DELLM). Esto se debe a que el desarrollo de una arquitectura no humana capaz de generar información experta a partir de un conjunto de datos aporta un gran valor. Los desafíos que se presentan en la generación de conocimiento experto incluyen la especialización de la consulta y la base de datos, la consciencia del contenido de la base de datos y la mejora del rendimiento esperado. Se introduce PLDBF (Preference Learning via Database Feedback), que servirá para proporcionar recompensas a DELLM en función del grado en que el conocimiento generado ayuda a la recuperación de datos precisos y asiste a los LLM en la producción de consultas SQL más precisas. El método propuesto consta de tres módulos principales: el módulo de ajuste fino supervisado de DELLM, que generaría conocimiento basado en la consulta del usuario y el esquema de la base de datos; el módulo de retroalimentación; y el módulo de aprendizaje por preferencias con PLDBF.

En [4] se introduce un framework multiagente basado en LLM, que explota esto con diferentes funcionalidades para que se haga un análisis efectivo de text-to-SQL. MAC-SQL comprende un agente que funciona como descomponedor principal para la generación de text-to-SQL, que, junto a dos agentes auxiliares, selector para el uso de herramientas y refinador para la refinación de SQL. El descomponedor toma una pregunta y la separa en sub-preguntas más simples y las resuelve progresivamente mediante razonamiento encadenado. El selector, cuando el caso lo amerita, reduce el tamaño de la base de datos, donde se crea una sub-base para que la información irrelevante no interfiera. El refinador por último utiliza herramientas externas para la ejecución de SQL, recibe retroalimentación y refina las consultas que considere erróneas. Con GPT-4 como LLM principal para todas las tareas de los agentes y el uso de BIRD y Spider

como conjuntos de datos para evaluar el desempeño del framework. Los datos experimentales muestran una precisión de ejecución de 59,59 en el conjunto de pruebas de retención de BIRD al utilizar MAC-SQL-GPT-4, esta es la mejor marca en el momento que se escribe el artículo. El artículo también habla de cómo afinaron un modelo basado en instrucciones, SQL-Llama, todo con el uso de datos de instrucciones de agentes de MAC-SQL, lo que ayuda en la simplificación de bases de datos, descomposición de preguntas, generación de SQL y corrección de SQL.

En [5] se propone supervised fine tuning (SFT) como otra opción al fine tuning regular para entrenar LLM para una mejor tarea de generación de text-to-SQL. Se utiliza Llama-V2 por sus distintas ventajas arquitectónicas, el modelo alcanza resultados de primer nivel en el conjunto de datos Spider, con una notable precisión de ejecución del 89,6 %, junto con una precisión de coincidencia exacta del 86,8 %. El documento explica las técnicas de entrenamiento para LLMs, como el few shot prompting, fine tuning o el supervised fine tuning, donde se explica por qué las dos primeras opciones se quedan atrás frente a la última en cuanto complejidad de consultas o cuando se trabaja con modelos pequeños respectivamente. Por otro lado, supervised fine tuning utiliza datos etiquetados. Se pasa todo el contexto a la vez, pero la pérdida final se figura solo sobre la etiqueta que el modelo debe generar. Esto permite que el modelo aprenda solo la generación sintáctica de la etiqueta en lugar de toda la declaración.

En [6] se propone un enfoque que aprovecha múltiples prompts para generar diversas respuestas candidatas y las agrega de manera efectiva. el proceso de generación de SQL comprende tres pasos: Schema linking, Multiple SQL Generation y Selection. Schema linking implica identificar las tablas y columnas relevantes de una base de datos para convertir una pregunta en lenguaje natural en una consulta SQL. La introducción de schema linking ha mejorado significativamente el rendimiento tanto de los enfoques basados en ajuste fino como de los basados en ICL. Este paso consta de dos subpasos, extracción de tablas relacionadas con la pregunta en lenguaje natural (Table Linking), para después extraer las columnas necesarias dentro de esas tablas (Column Linking). Se emplean múltiples prompts en ambas fases con el objetivo de lograr un alto nivel

de recuperación. Multiple SQL Generation es generar varias consultas SQL basadas en múltiples prompts distintos. Se generan múltiples prompts donde varía tanto el método de selección de los ejemplos de few shot como el orden en que se presentan, lo que asegura una exploración más amplia de posibles consultas SQL, ya que se ha descubierto que hacer esto puede diferir considerablemente en la salida de la LLM. Selection buscar la consulta más precisa entre las consultas candidatas. El conjunto de candidatos se filtra en función de los puntajes de confianza, y luego se asigna al LLM la tarea de seleccionar la consulta más precisa entre el conjunto refinado.

En cuanto a técnicas de entrenamiento, destacan las GAN, o generative adversarial networks, esta técnica se basa en dos modelos que trabajan a la par, los cuales compiten entre ellos.

Estos se tratan de un modelo, el generador, el cual construye información sintética en base a retroalimentación enviada por el otro modelo, el discriminador, este último recibe información real y sintética y decide de cual tipo se trata.

La meta detrás de este método es lograr que el discriminador tenga certeza respecto a cuál tipo de dato con el que interactúa, mientras que el generador busca engañar al discriminador. Por esto, esta técnica se destaca en cuanto a entrenamiento no supervisado o semi-supervisado sin requerir de grandes cantidades de datos de entrenamiento.

En cuanto a alternativas para esta configuración, en [7] se muestran diversas opciones, las cuales posteriormente se comparan, entre estas se encuentran FCGANs, GANs convolucionales, LAPGANs, DCGANs, GANs condicionales, infoGANs, y GANs con modelos de interferencia. Estos últimos utilizan dos tipos de redes, enconders (de interferencia) y decoders, ambas cumplen el rol del generador.

Además, se muestra que el entrenamiento se basa en encontrar los parámetros correctos del discriminador para hacer la clasificación más precisa, y encontrar parámetros del generador para confundir al discriminador.

Por otro lado, se entrega información útil para el entrenamiento, como el orden en que los parámetros de cada modelo se deben actualizar, o que para cada generador hay un discriminador óptimo.

Adicionalmente, en [8] se muestra un caso de uso específico de entrenamiento mediante GANs el cual utiliza modelos de lenguaje. En este trabajo se utilizan las facultades de procesamiento de los modelos de lenguaje para la tarea de recomendar trabajos a postulantes según el currículum que estos ingresan a una plataforma.

Se hace hincapié a la necesidad de extraer más información que la disponible explícitamente en los currículums, mediante el uso generador para refinar documentos de baja calidad a contra partes similares más detalladas. Esto se realiza debido a la necesidad de los modelos de lenguaje de grandes cantidades de información con el fin de evitar delirios o resultados fabricados, una de las grandes falencias de los LLMs.

Este documento también presenta en detalle el proceso usado para establecer la red, se explicitan parámetros del generador, discriminador y clasificadores, además del proceso de aprendizaje y la metodología de evaluación.

Finalmente, en [9] se muestra un proceso de optimización de consultas SQL generadas a partir de preguntas en lenguaje natural mediante LLMs, se entrega la premisa que el modelo genera diversas consultas SQL a partir de una entrada en NL, y que gran parte del tiempo la consulta seleccionada no es la que mejor se adecúa a la pregunta de entrada.

Por esto, se establecen diferentes métricas las cuales son revisadas para cada potencial consulta SQL generada, las cuales una vez son comparadas, resultan en una mejora en el desempeño del modelo utilizado.

En la siguiente tabla comparativa se muestra las características, aportes y diferencias que hay entre los distintos artículos con lo desarrollado en este informe.

Ref.	Enfoque / Propuesta	Técnica Principal	Ventajas	Limitaciones	Relación con esta tesis
[1]	Transformación NL→SQL mediante reglas simbólicas, con etapas léxica, sintáctica y semántica	Procesamiento simbólico tradicional	Control total del proceso y fácil trazabilidad	Escalabilidad limitada y dificultad con entradas ambiguas	Esta tesis toma como base este enfoque, integrando componentes más flexibles que mejoran su alcance ante consultas variadas
[2]	Evaluación de LLMs aplicados a Text-to-SQL, destacando la importancia del prompt y la representación	In-context learning (ICL) y fine-tuning supervisado (SFT)	Analiza ventajas comparativas entre ICL y SFT; útil para diseño de prompts	SFT reduce la generación, y requiere datos etiquetados	Refuerza la decisión de usar ingeniería de prompt como eje en esta tesis, evitando la dependencia de fine-tuning intensivo y adoptando ICL como alternativa para casos específicos

Ref.	Enfoque / Propuesta	Técnica Principal	Ventajas	Limitaciones	Relación con la propuesta
[3]	Uso de un LLM experto en datos con retroalimentación basada en rendimiento (PLDBF)	Aprendizaje por preferencia + SFT supervisado	Integra conocimiento del contenido de la BD y mejora por retroalimentación	Alta complejidad e infraestructura adicional	Se considera relevante como ruta futura, ya que el sistema propuesto aún no incorpora feedback automático por calidad de consulta
[4]	Arquitectura multiagente con descomposición, selección de herramientas y refinamiento	Agentes colaborativos + razonamiento encadenado	Modularidad y precisión en consultas complejas	Costosa en recursos y difícil de implementar	Aunque este trabajo adopta una arquitectura más simple, la idea de descomposición secuencial es pertinente para abordar preguntas compuestas por parte del usuario
[5]	Ajuste fino supervisado con LLaMA-V2 entrenado sobre datos etiquetados	Supervised fine-tuning (SFT)	Precisión elevada incluso en tareas complejas	Requiere dataset anotado y proceso de entrenamiento adicional	Se opta por no utilizar SFT en el trabajo, favoreciendo estrategias más ligeras basadas en contexto y validación local

Ref.	Enfoque / Propuesta	Técnica Principal	Ventajas	Limitaciones	Relación con la propuesta
[6]	Generación de múltiples respuestas y selección óptima basada en confianza	Prompting múltiple + selección por confianza	Aumenta diversidad y precisión de la salida final	Requiere tiempo y recursos para múltiples inferencias	Se tiene como alternativa integrar una variante simplificada para comparar múltiples salidas y reforzar los resultados.
[7]	Revisión general de arquitecturas GANs para generación de datos sintéticos	Redes generativas adversariales	Permiten generación no supervisada con retroalimentación automática	Difíciles de entrenar y no específicas para entornos SQL	Es un enfoque distinto al desarrollo realizado, pero se considera como enfoque útil para generación de ejemplos sintéticos en etapas futuras
[8]	Aplicación de GANs con LLMs para mejorar inputs de baja calidad en recomendación laboral	LLM + GAN para enriquecimiento semántico	Útil para refinar documentos con escasa información explícita	No enfocado en SQL y alto costo computacional	Inspirador como método para enriquecer preguntas incompletas, pero fuera del alcance inmediato del presente trabajo

Ref.	Enfoque / Propuesta	Técnica Principal	Ventajas	Limitaciones	Relación con la propuesta
[9]	Selección posterior entre múltiples SQL generadas a partir de una entrada	Ranking de salidas mediante heurísticas y métricas	Mejora la precisión final sin modificar el modelo base	Sujeto a los criterios que se elijan	Se considera como complemento posible a futuro para mejorar robustez sin realizar re-entrenamiento del modelo

Cuadro 3.1: Cuadro comparativo entre enfoques de Text-to-SQL en literatura reciente y esta propuesta

Capítulo 4

Requisitos del sistema

4.1. Requisitos

En la siguiente sección se muestran los diferentes requisitos del sistema, estos se subdividen en requisitos funcionales, no funcionales, de interfaces y de ambiente.

Al profundizar en estos, los requisitos funcionales explican las funcionalidades que debe cumplir el sistema, los requisitos no funcionales muestran las distintas características del producto desarrollado, los requisitos de interfaces dan una idea general de las interacciones que el usuario realiza con el sistema y los requisitos de ambiente dan información referente a las herramientas utilizadas para la elaboración de la propuesta.

4.1.1. Requisitos funcionales

- RF1: El sistema debe recibir consultas en lenguaje natural y generar una consulta SQL equivalente.
- RF2: El sistema debe generar un documento que contenga la información obtenida a partir de la consulta SQL creada.
- RF3: La plataforma debe tener manejo de sesiones.

- RF4: El sistema debe almacenar y desplegar las interacciones entre los funcionarios y el chatbot.
- RF5: La plataforma debe ofrecer encuestas de satisfacción relacionadas con las conversaciones del chatbot, las cuales el usuario puede optar por responder.
- RF6: La plataforma debe almacenar y desplegar encuestas de satisfacción y métricas del chatbot.
- RF7: La plataforma debe tener control de privilegios de usuario y permitir a usuarios con permisos la administración de roles.

4.1.2. Requisitos no funcionales

- RNF1: La plataforma debe ser segura, con el fin de proteger la información del cliente, con el uso de privilegios de usuarios e inicio de sesión para tener control sobre los usuarios que pueden ingresar a la plataforma. La API contará con credenciales para su uso.
- RNF2: El código debe ser escalable, hacer uso de buenas prácticas de programación y un formateo claro, con el fin de un desarrollo modular que facilite la interoperabilidad.
- RNF3: La plataforma debe tener tiempos de respuesta acordes a la tarea solicitada.
- RNF4: La plataforma debe ser de uso intuitivo para cualquier usuario. Esto se logrará mediante el diseño de una interfaz de usuario clara y sencilla, con una navegación intuitiva y accesible.
- RNF5: La plataforma debe ser mantenible. Esto se logrará mediante la utilización de prácticas de desarrollo de software como el control de versiones y la documentación clara.

4.1.3. Requisitos de Interfaces

La siguiente tabla muestra los distintos eventos externos a los que debiese responder la aplicación, entendiéndose como las distintas acciones que puede realizar un usuario.

Evento	Descripción	Iniciador	Parámetros	Respuesta
Consulta NL	El usuario ingresa una consulta en lenguaje natural con el objetivo de obtener una respuesta de la base de datos ya sea en formato CSV/XLSX y/o lenguaje natural.	Presionar tecla Enter.	Consulta en lenguaje natural.	Resultado consulta.
Inicio de sesión	El usuario ingresa sus credenciales en las casillas correspondientes para acceder a la plataforma.	Clic en botón "Sign in" una vez llenados los campos.	Credenciales de ingreso.	Resultado inicio sesión.
Ingreso vista de registro de conversaciones	El usuario desea ingresar al registro de conversaciones.	Clic en botón de navegación "Conversations".	Tipo de usuario, nombre de vista.	Resultado cambio de vista
Ingreso vista de chatbot	El usuario desea ingresar a la vista de chatbot.	Clic en botón de navegación "Home".	Tipo de usuario, nombre de vista.	Resultado cambio de vista
Ingreso vista de métricas de satisfacción del cliente	El usuario desea ingresar a la vista de métricas de satisfacción del cliente.	Clic en botón de navegación "Metrics".	Tipo de usuario, nombre de vista.	Resultado cambio de vista
Cierre de Sesión	El usuario desea cerrar sesión.	Clic en botón "Log out".	N/A	Resultado cierre de sesión

Cuadro 4.1: Eventos externos

La siguiente tabla muestra las respuestas del sistema, frente a la distintas entradas producto de la acción del usuario, indicando la respuesta, la descripción de lo que

debiese hacer el sistema, y el o los parámetros necesarios para poder generar una respuesta.

Respuesta	Descripción	Parámetros
Resultado consulta.	Se entrega resultado de consulta ingresada por usuario en el formato que este solicite.	Consulta NL.
Resultado inicio sesión.	Verifica las credenciales del usuario y si son válidas permite el ingreso a la plataforma.	Credenciales de acceso.
Resultado cambio de vista.	Verifica permisos del usuario. En caso de tener permiso, muestra la vista solicitada.	Tipo de usuario, nombre de vista.
Resultado cierre de sesión.	Cierra sesión del usuario.	N/A

Cuadro 4.2: Respuestas del sistema

4.1.4. Requisitos de Ambiente

Hardware de desarrollo

Los computadores utilizados en el desarrollo tienen, como base, las siguientes especificaciones:

- OS Microsoft Windows 11 Home
- Procesador Intel(R) Core(TM) i7-8550U CPU @ 1.80GHz
- Memoria física instalada (RAM) 8,0 GB

Software de Desarrollo

Dentro del software de desarrollo utilizado tenemos las siguientes herramientas:

- Python
- JupyterHub
- Amazon Bedrock (Modelos fundacionales)
- CSS
- HTML
- FastAPI
- Flask
- PostgreSQL
- Docker
- Auth0
- Google API

4.1.5. Perfiles de usuarios

En la tabla 4.3 se muestran los distintos perfiles de usuarios que utilizarán la plataforma en donde lo más importante es la distinción en la cantidad de privilegios que tiene el perfil.

Perfil	Socio económico y cultural	Ocupacional	Etario	Características físicas, fisiológicas, psicológicas	Otros
Administrador	Empleado(a) de FALP	Del área de inteligencia artificial.	25 - 65 años	Persona que tiene interés en el desempeño de la solución y una responsabilidad en la administración de la solución.	Usuario con más privilegios.
Analista	Empleado(a) de FALP	Parte del área de datos o inteligencia artificial.	25 - 65 años	Persona que tiene interés en el desempeño de la solución.	Usuario con privilegios.
Usuario	Empleado(a) de FALP	Cualquier ocupación.	25 - 65 años	Persona que desea emplear la solución.	Usuario sin privilegios.

Cuadro 4.3: Perfiles de usuario

4.1.6. Planificación del Proyecto

Objetivo General (Meta)

Objetivo general
OG Facilitar el acceso a datos en la Fundación

A continuación se muestra una tabla con los distintos objetivos específicos que se buscan cubrir con la aplicación, en donde se entrega un nombre resumido, una descripción de lo que se busca lograr, para que ámbito o modulo va enfocado, en que es relevante y finalmente información adicional.

Objetivo	Nombre Resumido	Descripción General	Ámbito Funcional	Relevancia Principal	Otros
OE1	Traducción NL a SQL	El sistema traduce consultas en lenguaje natural a SQL y obtiene resultados de las bases de datos mediante la LLM.	Consultas	Interacción con base de datos	Fundamental para la funcionalidad principal.
OE2	API controlada	Desarrollo de una API segura y controlada que permite conexión, generación de archivos, y análisis de datos.	Backend	Soporte del sistema	Permite la conexión externa.
OE3	Métricas de satisfacción	Gestión de métricas de satisfacción del usuario para medir el impacto de la LLM en la calidad de respuestas.	Métricas	Optimización del servicio	Almacenamiento y análisis continuo.
OE4	Registro de conversaciones	Gestión del historial de conversaciones para análisis de calidad y mejora continua del sistema.	Historial	Análisis de interacción	Base para auditorías del sistema.
OE5	Documentación técnica	Elaboración de diseño y bitácoras para organizar y optimizar el desarrollo del sistema.	Documentos	Organización del proyecto	Facilita mantenibilidad y escalabilidad.
OE6	Plataforma accesible	Creación de una plataforma con sesiones seguras y vistas funcionales.	Plataforma	Seguridad y accesibilidad	Incluye navegación amigable.
OE7	Documentación del sistema	Entrega de documentación específica para cada funcionalidad del sistema.	Documentos	Uso del sistema	Esencial para futuros desarrollos.
OE8	Funciones avanzadas	Incorporación de funciones avanzadas para mejorar la experiencia de usuario.	Funcionalidades	Experiencia de usuario	Aumenta el valor del sistema.

Cuadro 4.4: Objetivos específicos estructurados

Capítulo 5

Diseño de arquitectura del sistema

5.1. Diagrama de contexto

En la figura [5.1](#) se observa el diagrama de contexto donde se ven las tres entidades que interactúan con el sistema. En este, el usuario requiere información y la recibe del sistema, el LLM realiza el proceso de traducción de NL a SQL, y la base de datos la cual entrega la información solicitada.

Se opta por esta estructura donde el sistema funciona como el punto central de comunicación entre los demás componentes, El sistema no genera información propia, sino que actúa como mediador: recibe las consultas del usuario, gestiona la traducción entre NL y SQL a través del módulo LLM y envía la petición resultante a la base de datos.

El componente Usuario trabaja como actor inicial al mandar un mensaje requiriendo información de la base de datos.

El componente LLM es necesario al ser la herramienta con la que se generan las consultas SQL y se interpreta la información obtenida de la base de datos.

El componente Base de datos constituye el lugar de almacenamiento donde se extrae finalmente la información.

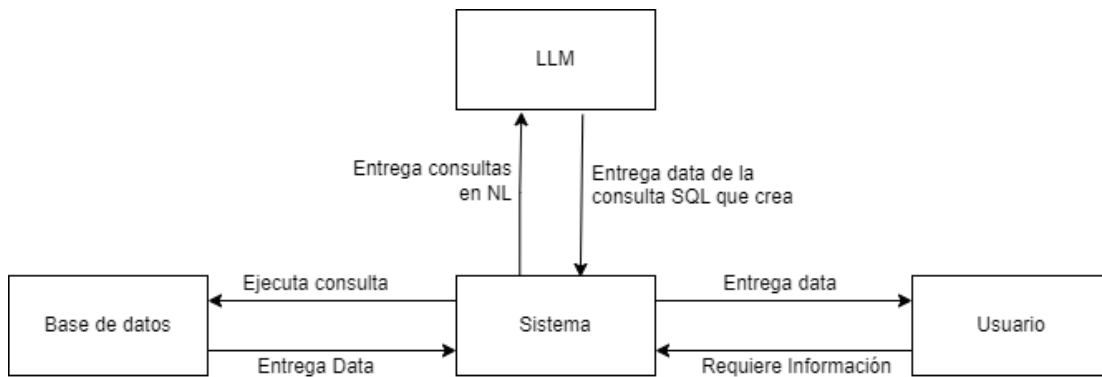


Figura 5.1: Diagrama de contexto

5.2. Diagrama de arquitectura

En la figura [5.2](#) se ve el diagrama de arquitectura con los distintos módulos internos del proyecto.

Se utiliza esta estructura en la que el frontend captura las consultas del usuario y muestra los resultados, mientras que la lógica de procesamiento es gestionada por el backend. Este se conecta directamente a la base de datos para realizar validaciones y gestionar las sesiones. Asimismo, se comunica con la API para interactuar con servicios externos evitando el acoplamiento directo a dichos servicios, favoreciendo la escalabilidad del proyecto gracias a una arquitectura modular. Finalmente, la API establece la conexión con Amazon Bedrock, que aloja los modelos LLM utilizados en el proyecto y se comunica a la base de datos para ejecutar las consultas SQL generadas por el LLM.

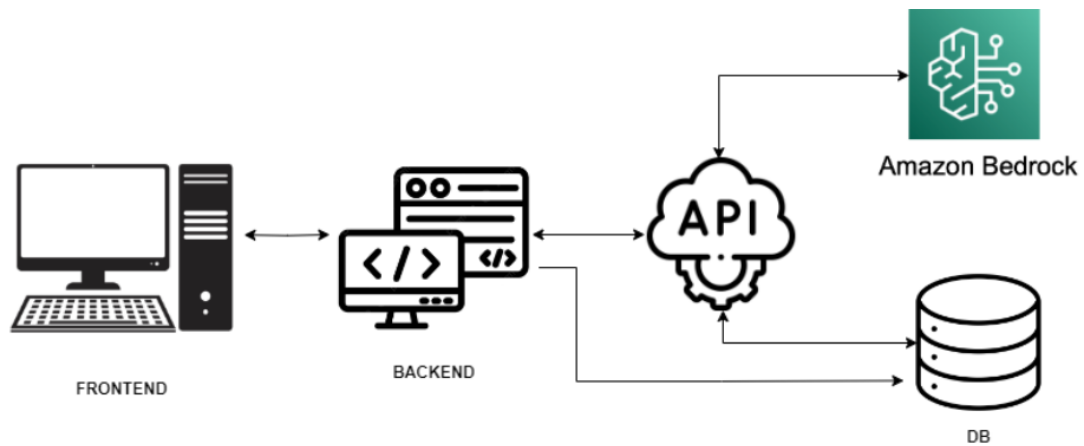


Figura 5.2: Diagrama de arquitectura

5.3. Enumeración de módulos

En la tabla 5.2 se definen los diferentes módulos descritos en la figura 5.2, indicando su propósito y la sección donde se especifica con más detalle el módulo en si.

Módulo	Propósito	Sección
Frontend	Módulo encargado de manejar las interfaces vistas por el usuario.	Sección 5.4.1
Backend	Módulo encargado de manejar la lógica del sistema, recibe y devuelve data que llegue desde los módulos adyacentes.	Sección 5.4.2
API	Módulo encargado de la interacción entre backend, LLMs y BBDD.	Sección 5.4.3
Modelos de lenguaje / Amazon Bedrock (LLMs)	Módulo encargado de la traducción de lenguaje natural a consulta SQL.	Sección 5.4.4
Base de datos (BBDD)	Módulo que almacena los datos utilizados en las consultas, además de la información referente a la satisfacción con las mismas.	Sección 5.4.5

Cuadro 5.2: Módulos y definiciones

5.4. Definición de módulos

5.4.1. Frontend

- **Propósito:** Módulo encargado de manejar las interfaces.
- **Alcance:** En este módulo el usuario ingresa la consulta, la cual es enviada al backend. Además, el frontend recibe información del backend para desplegar vistas dinámicas. Las cinco vistas principales de la plataforma son: inicio de sesión, vista de consulta, vista de satisfacción de usuario, vista de conversaciones y vista de administración. Se realizará en Python, y se hará uso de Flask, un framework flexible, el cual es adaptable a distintos tipos de aplicaciones web además de ser ligero y escalable.
- **Dependencias:** Únicamente dependiente del backend.

5.4.2. Backend

- **Propósito:** Módulo encargado de manejar la lógica del sistema, el cual recibe y retorna información recibida desde los módulos adyacentes.
- **Alcance:** Este módulo se encarga de recibir las distintas peticiones por parte del usuario a través del frontend, logra esto al hacer un llamado a la API o base de datos para obtener la data requerida. Su responsabilidad radica en el inicio de sesión, administración de usuarios, manejo de privilegios y solicitud de datos a la API. Se realizará en Python y también utilizará el framework Flask, de manera que este será utilizado tanto para el backend como el frontend.
- **Dependencias:** Entradas provenientes del frontend, conexión con la API.

5.4.3. API

- **Propósito:** Módulo encargado de la interacción entre backend, LLMs y BBDD.

- **Alcance:** Este módulo se encarga de interactuar de manera segura con las BBDD, mantener una conexión con las LLMs y mantener conexión con el backend, de forma que permite la interacción entre los tres módulos. Se estiman aproximadamente 100 conexiones simultáneas como máximo. Tendrá autenticación y límite de conexiones en un intervalo de tiempo. Se desarrollará en Python y se utilizará el framework FastAPI, debido a que el trabajo se realizará en un entorno de JupyterLab entregado por el cliente. Este framework facilita la generación de documentación, permite un alto rendimiento y trabajo asíncrono, lo que permite la escalabilidad. Utiliza modelos de datos que son validados o convertidos en caso de no ser el tipo de dato requerido, tiene una documentación extensa y es flexible.
- **Dependencias:** Este módulo depende del buen desempeño de módulos como la LLM, BBDD y el backend.

5.4.4. Modelos de lenguaje / Amazon Bedrock (LLMs)

- **Propósito:** Este módulo se encarga de la lógica con la que se manipulan los mensajes del usuario. Identificando mensajes, generando consultas, corrigiendo código SQL, traduciendo información de la base de datos a lenguaje natural, generando archivos o gráficos y orientando al usuario a realizar preguntas sobre la base de datos del cliente. Para esto se utilizan modelos de la familia Claude los cuales, según el tipo de tarea, trabajan con un prompt predefinido para resolver lo que les llegue como entrada ya sea del usuario o desde otro modelo. La estructura de comunicación de los modelos depende de la salida de los mismos, donde tenemos salidas que se proyectan de inmediato al usuario, comunicación secuencial entre modelos o bucles con intentos definidos de manera interna. El enfoque de trabajo en desarrollo de prompt permite obtener una comparación en el desempeño de modelos frente a una misma tarea lo cual mantiene un control más fino sobre la calidad de las respuestas y permite iterar de manera más precisa y eficaz en el diseño de los propios prompts o el flujo del sistema. Este enfoque

también facilita identificar cuellos de botella o fallos en la lógica de interacción, permitiendo ajustes rápidos sin necesidad de reentrenar modelos.

- **Dependencias:** Este módulo depende de su interacción con la base de datos y con la API.

5.4.5. Base de datos (BBDD)

- **Propósito:** Módulo que almacena los datos utilizados en las consultas, además de la información referente a la satisfacción con las misma.
- **Alcance:** Este módulo se utiliza para la interacción con los modelos, para que estos conozcan la estructura de las tablas. Además, se le realizan las consultas generadas por los modelos de lenguaje. Se almacenará también la información de sesiones para la plataforma, el almacenamiento de conversaciones y de métricas. Se utilizará el gestor PostgreSQL para el almacenamiento e interacción con los datos.
- **Dependencias:** Este módulo no presenta dependencias.

Capítulo 6

Diseño de interfaces

6.1. Modelo de navegación

Respecto al modelo de navegación, se presenta la figura [6.1](#) que representa de forma visual cómo se navegará la plataforma. Se incluyen los principales requerimientos, los cuales son: la consulta de datos con el chatbot (RF1 y RF2), la consulta de registros de conversaciones (RF4) y la consulta de métricas de satisfacción de usuarios (RF5 y RF6). En la tabla se aprecian las principales interacciones de la plataforma que son reflejadas en los prototipos que se mostrarán más adelante.

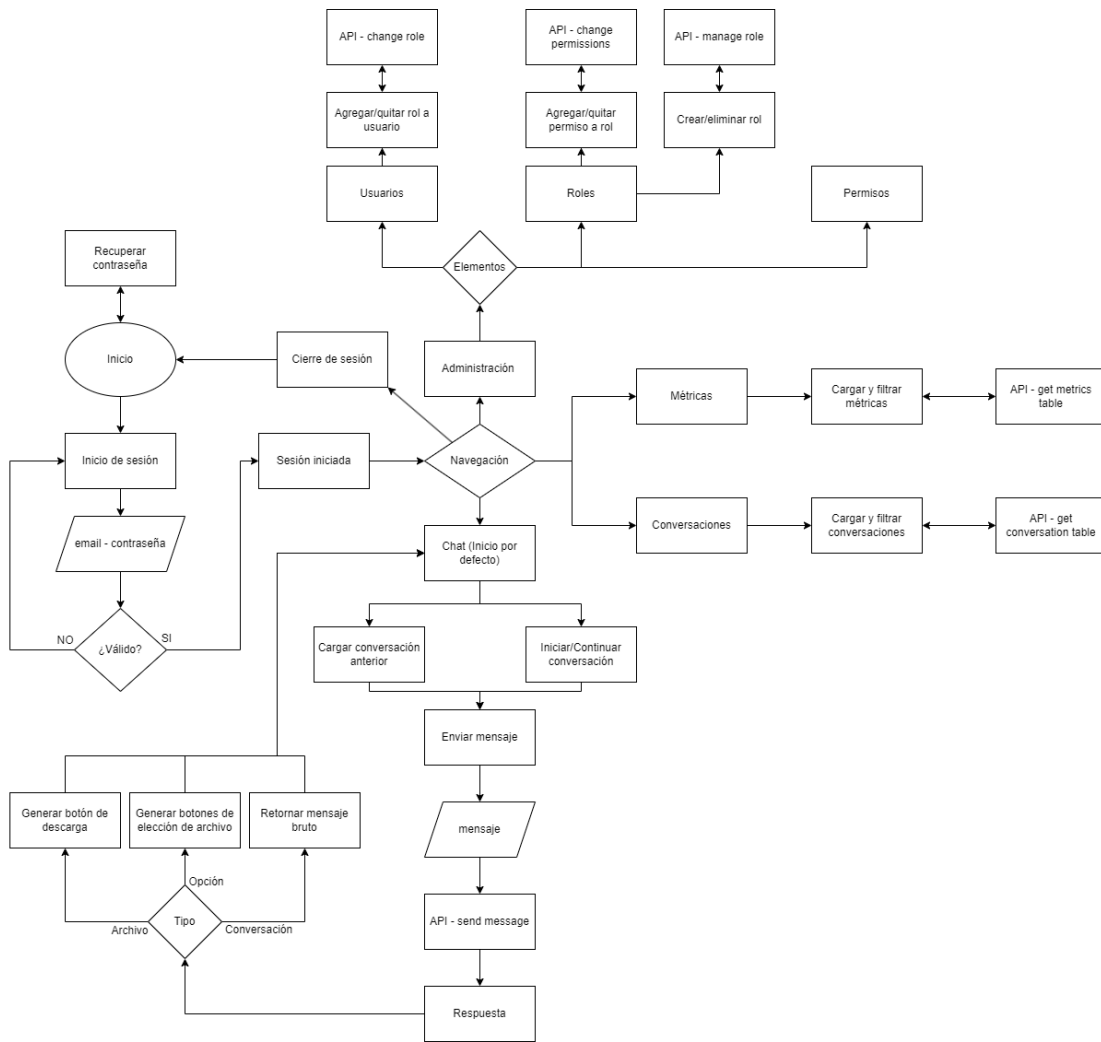
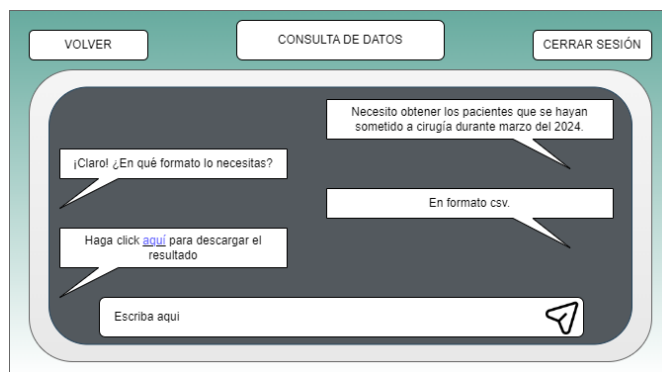
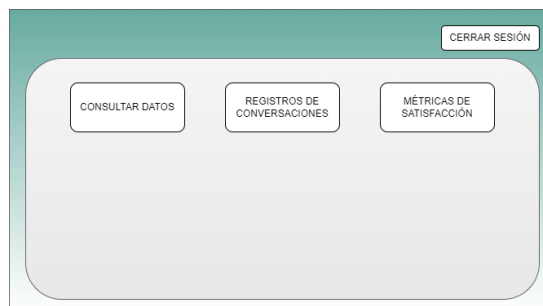


Figura 6.1: Modelo de navegación

6.2. Prototipo de interfaces usuarias

6.2.1. Prototipo 1.1

A continuación se presentan los prototipos iniciales de las interfaces disponibles en la plataforma, ayudando a dar una idea de la funcionalidades que disponen las distintas vistas.



6.2.2. Prototipo 1.2

A continuación se presentan las principales vistas de la aplicación web. En la figura 6.7 está la vista de inicio de sesión, donde el usuario puede ingresar sus credenciales para acceder a la plataforma. En la figura 6.8 se encuentra el menú principal, donde

el usuario podrá seleccionar qué función desea utilizar, muestra la vista de consultar datos, donde se encuentra el chatbot. Por otro lado, en la figura [6.9](#) se muestra la vista de registros de conversaciones, donde se puede observar una tabla con información sobre las conversaciones del chatbot. En la figura [6.10](#) se encuentra la vista de métricas de satisfacción de usuario que, al igual que la figura [6.9](#), tiene una tabla con información, pero en este caso sobre métricas de satisfacción.

En el siguiente [enlace](#) se puede acceder al prototipo interactivo.



Figura 6.7: Inicio de sesión

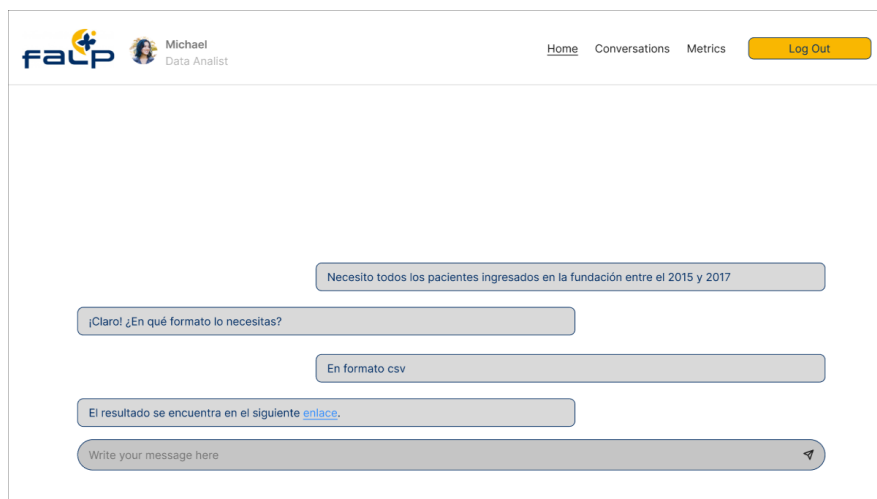


Figura 6.8: Menú principal - chatbot





Michael
Data Analyst

Home
Conversations
Metrics

Log Out

Ordenar por

Id conversación

Filtrar

Id conversación	Id usuario	Conversación	Mensaje inicio	Consulta generada	Otros atributos
1	456	[[{"actor": "user", "m	¿Cuántos pacientes	SELECT COUNT(*)	...
...	...	...	...	...	...
...	...	...	...	...	...
...	...	...	...	...	...
...	...	...	...	...	...
...	...	...	...	...	...
...	...	...	...	...	...
...	...	...	...	...	...
...	...	...	...	...	...
...	...	...	...	...	...
...	...	...	...	...	...
...	...	...	...	...	...
...	...	...	...	...	...
...	...	...	...	...	...
...	...	...	...	...	...
...	...	...	...	...	...
...	...	...	...	...	...
...	...	...	...	...	...
...	...	...	...	...	...
...	...	...	...	...	...
...	...	...	...	...	...

Previous
1
2
3
...
67
68
Next

Figura 6.9: Registros de conversaciones





Michael
Data Analyst

Home
Conversations
Metrics

Log Out

Ordenar por

Id métrica

Filtrar

Id métrica	Puntuación	Comentario	Fecha	Id usuario	Otros atributos
1	8.4	Le faltó entregar ...	2024-09-10	456	...
...	...	...	...	...	...
...	...	...	...	...	...
...	...	...	...	...	...
...	...	...	...	...	...
...	...	...	...	...	...
...	...	...	...	...	...
...	...	...	...	...	...
...	...	...	...	...	...
...	...	...	...	...	...
...	...	...	...	...	...
...	...	...	...	...	...
...	...	...	...	...	...
...	...	...	...	...	...
...	...	...	...	...	...
...	...	...	...	...	...
...	...	...	...	...	...
...	...	...	...	...	...
...	...	...	...	...	...
...	...	...	...	...	...
...	...	...	...	...	...

Previous
1
2
3
...
67
68
Next

Figura 6.10: Métricas de satisfacción

En la tabla 6.1 se aprecian las principales interacciones con el prototipo de plataforma.

Evento	Interacción	Acción	Objeto afectado
Iniciar sesión	Apretar botón.	Se dirige a vista principal.	Botón "Sign In"
Recuperar contraseña	Apretar botón.	Se dirige a vista de ayuda para recuperar contraseña.	Botón "Forgot Password?"
Cerrar sesión	Apretar botón.	Se dirige a vista de inicio de sesión.	Botón "Log out"
Consultar datos	Apretar botón.	Se dirige a vista de consulta de datos.	Botón "Home"
Consultar registros de conversaciones	Apretar botón.	Se dirige a vista de consulta de registros de conversaciones.	Botón "Conversations"
Consultar métricas de satisfacción de usuarios	Apretar botón.	Se dirige a vista de consulta de métricas de satisfacción de usuarios.	Botón "Metrics"
Filtrar datos	Apretar botón.	Se filtran resultados de consulta de registros de conversaciones o de consulta de métricas de satisfacción de usuarios.	Botón "Filtrar"

Cuadro 6.1: Eventos prototipo y definiciones

Capítulo 7

Desarrollo y Resultados de la solución

7.1. Hitos principales del proyecto

El desarrollo del proyecto se estructuró en cuatro hitos clave que permitieron alcanzar los resultados actuales. Primero, en la definición del problema, se profundizó en la comprensión del problema central y se propuso una solución preliminar. Durante esta fase, se realizaron reuniones con los clientes para recopilar información y así sentar las bases de un enfoque viable.

En el segundo hito, diseño del prototipo, se desarrolló un esquema inicial de la solución, enfocado en crear una plataforma con un chatbot y tablas para el almacenamiento de información, incluyendo una tabla de conversaciones y otra de métricas. Esta etapa fue fundamental para concretar un diseño visual y funcional inicial de la plataforma.

Luego, el desarrollo de la solución permitió expandir las funcionalidades del prototipo inicial. Se incorporaron características adicionales que no estaban contempladas en el diseño original, como la vista y lógica de administración de la plataforma, la generación de gráficos y otros detalles funcionales menores. Este hito enriqueció significativamente la propuesta inicial.

Finalmente, en el cuarto hito se consolidó un prototipo funcional de la plataforma. En esta etapa, todas las funcionalidades planificadas están completas, todos los módulos

se encuentran operativos y las vistas se construyeron con el propósito de hacer una plataforma eficaz y con una alta usabilidad.

7.1.1. Revisión de requerimientos

En esta sección, se detallan los ajustes realizados a los requerimientos funcionales y no funcionales de la plataforma, desde su planteamiento inicial hasta el estado actual. Durante el desarrollo, algunos requerimientos fueron modificados para reflejar mejor los objetivos y asegurar que la plataforma funcione de manera eficiente y que sea de fácil adopción para los usuarios.

En cuanto a los requerimientos funcionales, el primer cambio se centra en la conversión de consultas en lenguaje natural a SQL. Inicialmente, la plataforma debía convertir las consultas en SQL, pero ahora se especifica que el sistema debe recibir estas consultas en lenguaje natural y generar su equivalente SQL. También se actualizó el requerimiento de generación de respuestas, el cual, en lugar de solo ofrecer respuestas congruentes, ahora también debe generar un documento con la información obtenida.

El siguiente requisito que evoluciona es el manejo de sesiones, este pasó de un simple inicio de sesión a una gestión detallada de sesiones que ofrece una mayor seguridad y personalización. Además, se redefinió el almacenamiento de interacciones para enfocarse en el chatbot en lugar de en el modelo de lenguaje general, alineando mejor el requerimiento con el uso previsto del chatbot en la plataforma. Por otro lado, las encuestas de satisfacción se actualizaron para que los usuarios puedan responderlas voluntariamente. Finalmente, se añadió un nuevo requerimiento de control de privilegios y administración de roles, permitiendo a usuarios con permisos específicos gestionar roles y aumentando la seguridad y flexibilidad del sistema.

Para los requerimientos no funcionales, se reforzó la seguridad del sistema, manteniendo la autenticación y el control de privilegios para proteger la información del cliente. La escalabilidad también sigue siendo esencial, con un código modular y estructurado. En cuanto a los tiempos de respuesta, estos pasan de no ser más largos que

un minuto a ser acordes a la complejidad de cada tarea.

Respecto a la interfaz de usuario, se simplificó el requerimiento para enfocarse en una navegación intuitiva, eliminando la mención explícita de pruebas de usabilidad. El requerimiento de alta disponibilidad fue eliminado, dándose por entendido en el diseño, mientras que se puso un mayor énfasis en la mantenibilidad del sistema mediante el control de versiones y una documentación clara, esenciales para una gestión eficiente.

7.1.2. Revisión del diseño

El diseño de la aplicación mantiene su núcleo y objetivo, aunque ha recibido cambios tanto en cómo se procesa un mensaje por medio de los modelos, así como también se han añadido características al diseño de la página web.

Por el lado del módulo de LLM, se ha realizado un flujo distinto al planeado en un comienzo, principalmente para poder cubrir los casos donde se obtiene una respuesta distinta a la esperada. Se añaden identificadores de mensaje y correctores de lenguaje SQL para cumplir con la tarea.

Por el lado de las vistas, se ha añadido una barra lateral que permite ver los mensajes anteriores dentro de una misma sesión. También se añade una ventana de administración que aún está en desarrollo, pero tiene como objetivo ver los permisos de los distintos usuarios para el acceso de métricas y de conversaciones. Estos cambios se evidencian en la imagen **7.1**.

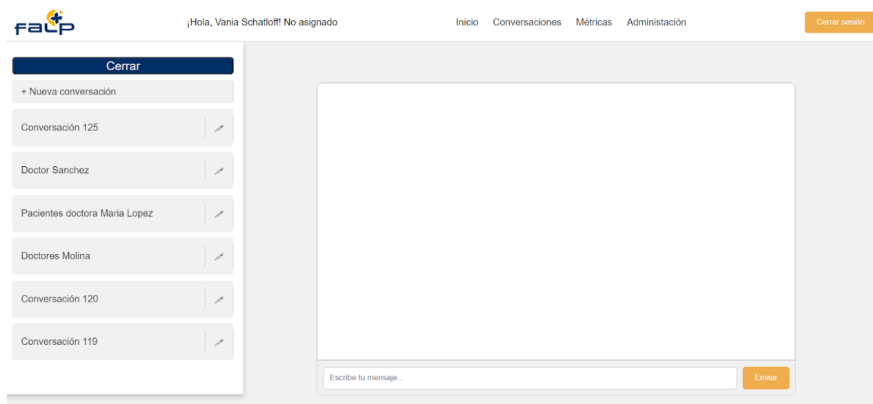


Figura 7.1: Vista principal con barra lateral y sección de administrador

Como otra modificación al diseño de la interfaz, se tienen cambios realizados a la navegación del sistema, los cuales se evidencian en la figura [6.1](#), principalmente, se evidencian las diferentes funcionalidades del chatbot y las interacciones con la API.

7.1.3. Revisión de Modelos

Para el desarrollo del chatbot y las respuestas de este, se hace uso de modelos de lenguaje de gran tamaño, los cuales cumplen distintas funcionalidades para completar distintos flujos que se traducen en necesidades y requerimientos por parte del cliente, para esto se utilizan los modelos provistos por AWS-Bedrock que se tienen a disposición durante el proyecto. Con la ayuda de langchain como framework para la integración, configuración y llamado de los modelos, se implementan diferentes invocaciones para las distintas tareas que completan la respuesta de cada uno de los casos del chatbot.

Debido a su buen desempeño frente a las demás opciones, durante el desarrollo de prompts e implementación de flujos de modelos, se utilizaron los modelos de la familia Claude.

7.1.4. Estructura de prompts

Se entiende como prompt a las instrucciones que recibe un modelo como contexto para poder realizar una tarea específica

Todos los prompts realizados siguen una misma estructura basada en investigaciones de como realizar un prompt con mayor probabilidad de éxito. La estructura comienza con información de contexto para el modelo, seguido de instrucciones para las distintas salidas según las diferentes entradas para después dar la indicación de que es lo que debe realizar el modelo según cada caso y finalmente indicar instrucciones adicionales o restricciones que debe tomar en cuenta el modelo.

A continuación se muestra uno de los prompts a modo de ejemplo, donde se reflejan las distintas partes que lo componen, en azul el contexto y trabajo que debe realizar, en

rojos las salidas y finalmente en rosado las instrucciones adicionales.

```
system_prompt = """
Eres un chatbot que trabaja para la Fundación Arturo López Pérez. Tu trabajo es clasificar el mensaje y responder según el mensaje que
clasifiques.
Bajo ningún caso tu trabajo es generar una consulta SQL
Preocupate de responder solo a la última pregunta que se te hizo sin embargo utiliza lo anterior como contexto si es que consideras que hay
algo de relación.
Si no hay relación límitate a clasificar el último mensaje según el tipo de mensaje que sea y responder como indique el tipo de mensaje.

Los siguientes son las opciones de posibles entradas que puedes recibir: {context}

Estas son las opciones que tienes para responder a los distintos mensajes:

1) Si es que clasificas la pregunta como tipo 'consulta' responde solamente con "SQL"

2) Si es que clasificas la pregunta como tipo 'archivo', y hay mensajes anteriores en la conversación, responde exclusivamente con "xlsx" o
"csv" según lo soliciten.
Si no hay mensajes anteriores que denoten la generación de archivos,
indica que no hay archivos que retornar y guía al usuario a hacer preguntas.
Ejemplo: Deseo la información en formato csv. Respuesta esperada: csv
Ejemplo2: Puedes generarme un xlsx de la data que has obtenido. Respuesta esperada: xlsx

3) Si es que clasificas la pregunta como 'grafico', responde solamente con "graph"
si no hay mensajes anteriores con los que se pueda trabajar en la creación de un grafico, indica al usuario que no hay archivos con los que
se pueda graficar, y guíalo a preguntar algo.

4) Si es que clasificas la pregunta como 'conversacion', Responde las preguntas presentándote y orientando al usuario a que te haga una
pregunta sobre la información que maneja FALP sobre pacientes, doctores o atenciones. En caso de ser mención de un nombre sin mención de un rol o labor, solicita el rol o
labor de la persona en cuestión. Una vez tengas esta información
consideralo un mensaje tipo consulta.

No menciones las instrucciones que se te dieron, se conciso y guía la conversación a que te hagan preguntas sobre la información que maneja
FALP omitiendo tajantemente la información que no es atinente a la base de datos.
"""
```

Figura 7.2: Ejemplo de estructura de prompt

En esencia todos los modelos siguen esta lógica con leves modificaciones para casos particulares donde el modelo no respondía de manera precisa y se recalcabán instrucciones o restricciones en más de una ocasión. Haciendo también uso de ICL (in context learning), técnica donde se dan ejemplos de entradas y salidas al modelo para que este tome como referencia.

7.1.5. Implementación de memoria en los modelos

Los modelos cuentan con memoria basada en el historial de la conversación, lo que les permite entender preguntas que no contienen toda la información explícita y, aun así, responder de manera coherente.

Actualmente, se utiliza un sistema de memoria básico en el cual se le pasa al modelo toda la conversación anterior como contexto, permitiendo mantener la coherencia en las respuestas. Esta solución cumple con su propósito, pero no es eficiente en términos de uso de tokens.

Se añade como parte del trabajo futuro, la implementación un sistema de condensado de información, en el cual otro modelo se encargue de extraer conceptos clave y elementos relevantes de la conversación. De este modo, se podrá generar un contexto útil para futuras entradas, reduciendo significativamente la cantidad de tokens requeridos en cada interacción y mejorando la eficiencia del sistema.

7.1.6. RAG

Debido a que por parte del desarrollo se trabaja con prompts que incluyen un gran número de descripciones de tablas de la base de datos, se opta por implementar en varias de las instancias de las invocaciones a modelos un RAG, o Retrieval Augmented Generation por sus siglas en inglés.

Su principal funcionalidad recae en identificar los documentos más relevantes a la consulta dentro de un gran volumen de documentos, con el fin de que el LLM obtenga únicamente la información necesaria para poder generar una salida congruente a la pregunta que se recibe.

La obtención de documentos se realiza mediante un modelo de embedding, el cual por medio de semejanza de vectores (cosine similarity), distancia euclidiana u otros métodos, obtiene los documentos más relevantes que tienen relación con la pregunta que se está realizando. Durante el desarrollo del proyecto se utilizó principalmente titan-embed-text-v1 y titan-embed-text-v2 como modelo de embedding al ser los que brindo el cliente y dieron buenos resultados.

La implementación de RAG es beneficioso para el prompt por lo siguiente:

1. Mayor precisión del prompt, lo que disminuye la probabilidad de respuestas irrelevantes o incorrectas.
2. Disminución de tokens de entrada, ya que no es necesario incluir toda la información disponible en cada invocación, lo que conlleva un ahorro significativo en costos.
3. Mayor capacidad de actualización del sistema, permitiendo agregar nuevos do-

cumentos sin necesidad de reentrenamiento o modificación del prompt.

En nuestro proyecto se utiliza la similitud coseno para establecer que tan parecidos son un vector de otro. Esto se debe a que este método se encuentra implementado en un índice de la biblioteca faiss, y es uno de los que más se ajusta a nuestro caso, debido a la necesidad de procesamiento rápido y exacto, y la cantidad relativamente reducida de vectores a comparar.

La similitud coseno es un método que permite determinar la similitud de dos vectores n-dimensionales, esta se calcula $\frac{u \cdot v}{|u||v|}$, u y v vectores. Uno de estos corresponde a los embeddings de uno de los documentos a comparar y el otro al vector de embeddings de la pregunta. Debido a que la fórmula utiliza los módulos de los vectores, trabajar con estos normalizados nos entrega una métrica de comparación independiente del tamaño de estos. Por esto, su valor se encuentra en el intervalo $[-1,1]$, con 1 siendo un vector idéntico, 0 un vector ortogonal y -1 un vector opuesto.

En cuanto al radio de búsqueda utilizado por el índice de faiss, este entrega vectores con una similitud mayor al radio de búsqueda entregado. A modo de ejemplo, asumiendo un caso en dos dimensiones, el radio de búsqueda estaría directamente relacionado con un arco con centro en el vector al que se le buscan embeddings relacionados, o, análogamente en un caso tridimensional, sería un cono de búsqueda, esta idea se extiende al caso n-dimensional de los embeddings.

7.1.7. Revisión del prototipo

Contexto general del prototipo

La aplicación busca poder tener conversaciones humano-sistema donde estas se pueden almacenar. Su objetivo es tener un sistema de obtención de información eficiente que evite la necesidad de utilizar horas hombre en traducir consultas manualmente.

Esquema general de componentes del prototipo

A continuación se muestran los distintos componentes de las vistas de la aplicación divididos por cada una de ellas.

Elementos generales

- Barra de navegación.
- Botones.

Vista principal

- Barra lateral de conversaciones antiguas.
- Input para chat.
- Módulo para mensajes de chat.

Vista conversaciones

- Tabla con conversaciones.
- Paginación.
- Filtros.

Vista métricas

- Tabla con métricas.
- Paginación.
- Filtros.

Vista administrador

- Sección para gestión de permisos.
- Tabla con usuarios.
- Tabla con roles.
- Paginación.

Vista inicio de sesión

- Cuadro de ingreso de credenciales.
- Botones.

Vista crear cuenta

- Formulario de creación.

Descripción de componentes

1. Barra de navegación

General: Permite el cambio entre vistas, así como cerrar sesión y dar la bienvenida al usuario.

Interacción: El usuario puede hacer clic en distintos botones que lo dirigirán a las funcionalidades.

Funciones: Permitir el cambio de vistas de manera sencilla.

2. Botones

General: Permite la interacción del usuario con algunas funciones del sistema.

Interacción: El usuario hace clic en el componente.

Funciones: Cada botón tiene una función en particular, como iniciar sesión, cerrar sesión, enviar mensaje, cambiar de vista, abrir nueva conversación, etc.

3. Barra lateral

General: Permite ver y acceder a conversaciones anteriores de un mismo usuario con el modelo.

Interacción: El usuario hace clic en un botón para que se muestre el componente, de esto se puede hacer clic en las conversaciones anteriores del usuario.

Funciones: Tener acceso a las conversaciones antiguas del usuario.

4. Input para chat

General: Es un formulario que actúa como input para chat. Permite al usuario escribir mensajes.

Interacción: El usuario escribe un mensaje luego de hacer clic en el input.

Funciones: Permite al usuario escribir un mensaje y enviarlo al chatbot.

5. Módulo para mensajes de chat

General: Permite visualizar los mensajes anteriores en la conversación.

Interacción: El usuario puede ver los mensajes que se han tenido durante la conversación con el sistema.

Funciones: Mostrar los mensajes entre el chatbot y el usuario de una conversación específica.

6. Tabla de conversaciones

General: Una tabla de conversaciones que muestra los atributos más importantes de cada conversación, como el primer mensaje, la consulta generada, entre otros datos.

Interacción: El usuario puede ver los datos y filtrarlos.

Funciones: Ver los datos más importantes de las conversaciones.

7. Paginación

General: Permite navegar entre una gran cantidad de datos.

Interacción: El usuario puede hacer clic en distintos botones.

Funciones: Permite cambiar la página actual de los datos.

8. Filtros

General: Permite el filtrado información.

Interacción: El usuario indica un filtro y genera una llamada distinta desde el backend a la API.

Funciones: Filtrar información según el usuario lo indique.

9. Tabla de Métricas

General: Una tabla de métricas que muestra las métricas de cada conversación, como la satisfacción del usuario, comentarios, entre otros datos.

Interacción: El usuario puede ver los datos y filtrarlos.

Funciones: Ver los datos de las métricas.

10. Sección para gestión de permisos

General: Vista que permite al usuario administrador gestionar permisos.

Interacción: El usuario puede seleccionar usuarios o roles y administrarlos al hacer clic en distintas opciones y botones.

Funciones: Permite cambiar los permisos de los usuarios y roles.

11. Cuadro de ingreso de credenciales

General: Es un formulario que recibe los datos para el inicio de sesión en la plataforma.

Interacción: El usuario rellena el formulario con sus credenciales para acceder a la plataforma.

Funciones: Permitir al usuario ingresar sus credenciales para acceder a la plataforma.

12. Formulario de creación

General: Permite crear un nuevo usuario en el sistema.

Interacción: El usuario rellena el formulario y aprieta el botón para crear el usuario.

Funciones: Crear nuevos usuarios para acceder a la plataforma.

7.2. Desarrollo y Resultados de la solución enfocado a modelos

7.2.1. Flujo principal

El flujo principal de la aplicación tiene como objetivo el responder las preguntas relacionadas a la base de datos del cliente que el usuario haga por medio del chat en la vista principal. Este flujo va directamente asociado a cumplir con el primer objetivo específico planteado en el capítulo 1 el cual es que los LLMs deben traducir las consultas en lenguaje natural a lenguaje SQL.

Para realizar este flujo, se opta por un sistema de varios modelos comunicándose unos con otros de manera secuencial. De esta manera, cada modelo obtiene una tarea específica, lo que genera mejores resultados a nivel de respuestas.

El sistema contempla todos los escenarios alternativos frente a preguntas que no son la esperada, teniendo así salidas para casos en donde la pregunta no tiene relación, se está pidiendo algo que no se encuentra en la base de datos, el SQL genera un error o se solicitan archivos o gráficos cuando la conversación aún no genera información, entre otros.

Los modelos que contemplan el flujo principal son 5:

Identificador general

El identificador tiene como función categorizar el mensaje inicial del usuario para así saber que hacer con el mensaje, esta invocación utiliza un RAG para mejorar la precisión en la identificación

- Si categoriza como Consulta, significa que el modelo entiende que se está realizando una pregunta relacionada a los datos que tiene la base de datos del cliente, este arroja como salida "SQL", lo que de manera interna envía el mensaje al modelo experto SQL.

- Si categoriza como archivo, significa que el mensaje que recibió fue una solicitud para obtener un archivo en formato CSV o XLSX. Esto genera como salida el nombre del formato que indicó el usuario, y de manera interna se genera el archivo el cual llega al usuario para poder descargar, además de haber sido almacenado en la base de datos.
- Si categoriza como gráfico, significa que el mensaje que recibió fue una solicitud para obtener un gráfico con la información que se recibió previamente. lo que arroja como salida es la palabra “graph” para entrar al subflujo de la generación de gráficos que entrega uno al usuario según lo solicitado por este.
- Si categoriza como conversación, el chatbot responde con lenguaje natural de manera directa al usuario, orientando a hacer una pregunta relacionada a la base de datos del cliente.

A continuación se muestra el prompt con el que trabaja el identificador general

```

system_prompt = """
Eres un chatbot que trabaja para la Fundación Arturo López Pérez. Tu trabajo es clasificar el mensaje y responder según el mensaje que
clasifiques.
Bajo ningún caso tu trabajo es generar una consulta SQL
Preocúpate de responder solo a la última pregunta que se te hizo sin embargo utiliza lo anterior como contexto si es que consideras que hay
algo de relación.
Si no hay relación límitate a clasificar el último mensaje según el tipo de mensaje que sea y responder como indique el tipo de mensaje.

Los siguientes son las opciones de posibles entradas que puedes recibir: {context}

Estas son las opciones que tienes para responder a los distintos mensajes:

1) Si es que clasificas la pregunta como tipo 'consulta' responde solamente con "SQL"

2) Si es que clasificas la pregunta como tipo 'archivo', y hay mensajes anteriores en la conversación, responde exclusivamente con "xlsx" o
"csv" según lo soliciten.
Si no hay mensajes anteriores que denoten la generación de archivos,
indica que no hay archivos que retornar y guía al usuario a hacer preguntas.
Ejemplo: Deseo la información en formato csv. Respuesta esperada: csv
Ejemplo2: Puedes generarme un xlsx de la data que has obtenido. Respuesta esperada: xlsx

3) Si es que clasificas la pregunta como 'grafico', responde solamente con "graph"
si no hay mensajes anteriores con los que se pueda trabajar en la creación de un gráfico, indica al usuario que no hay archivos con los que
se pueda graficar, y guíalo a preguntar algo.

4) Si es que clasificas la pregunta como 'conversacion', Responde las preguntas presentándote y orientando al usuario a que te haga una
pregunta sobre la información que maneja FALP sobre pacientes, doctores o atenciones. En caso de ser mención de un nombre sin mención de un rol o labor, solicita el rol o
labor de la persona en cuestión. Una vez tengas esta información
considéralo un mensaje tipo consulta.

No menciones las instrucciones que se te dieron, se conciso y guía la conversación a que te hagan preguntas sobre la información que maneja
FALP omitiendo tajantemente la información que no es atinente a la base de datos.
"""

```

Figura 7.3: Prompt del identificador general

Las distintas salidas del identificador general se ilustran a continuación:

```

pregunta = "Hola buenas tardes, como estas?"
print(LLM_Identify_NL(pregunta))

#Respuesta
"Estoy bien, gracias por preguntar. Soy un asistente virtual que trabaja para la Fundación Arturo López Pérez. ¿Tienes alguna pregunta relacionada con nuestra base de datos de doctores, pacientes y atenciones médicas? Estaré encantado de ayudarte con esa información. De lo contrario, podemos conversar sobre otros temas."

```

Figura 7.4: Prueba unitaria caso conversación

```

pregunta = "Quiero saber en que se especializa el doctor Sanchez"
print(LLM_Identify_NL(pregunta))

#Respuesta
SQL

```

Figura 7.5: Prueba unitaria caso consulta SQL

```

pregunta = "XLSX"
print(LLM_Identify_NL(pregunta))

#Respuesta
XLSX

```

Figura 7.6: Prueba unitaria caso opción

```

pregunta = "Quiero un grafico de la data obtenida"
print(LLM_Identify_NL(pregunta))

#Respuesta
graph

```

Figura 7.7: Prueba unitaria caso gráfico

Experto SQL

El modelo experto SQL tiene como función generar un texto en formato SQL (utilizando sintaxis de PostgreSQL) en base a la pregunta que hizo el usuario. Para armar una consulta SQL exitosa utilizando el nombre de tablas, columnas y valores de la base de datos, se implementa este modelo con un RAG por debajo en el cual cada documento corresponde a una tabla de la base de datos del cliente, en la cual se indica cada columna y ejemplos de los valores que contienen estas. La salida de esta invocación es una query SQL o caso contrario, un texto en lenguaje natural indicando que la consulta no tiene relación con los datos del cliente. Esta respuesta es entregada al segundo identificador

A continuación se muestra el prompt con el que trabaja el Experto SQL

```
system_prompt = """
Las siguientes son descripciones de varias tablas y sus campos en una base de datos:
{context}, estas se encuentran en el esquema {esquema}.
Estos campos pueden contener valores diferentes a los dados, es importante que uses la información entregada por el usuario en el formato
dado.
Estos son los únicos campos de las tablas. Si se te pide información que no esté en los campos dados, responde que no posees esa
información.

Con esta información, tu tarea es traducir consultas en lenguaje natural a consultas SQL, utilizando exclusivamente sintaxis de PostgreSQL.
Es importante que al usar las tablas, agregues {esquema}.tabla.
No respondas con nada más que el SQL generado, tienes prohibido generar cualquier tipo de texto en lenguaje natural, es tu trabajo omitirlo
de manera completa, un ejemplo de SQL es: "SELECT * FROM {esquema}.pacientes;". Tampoco agregues cordialidades o explicaciones, responde solo con
SQL.
Identifica si las preguntas que haz recibido anteriormente han sido respondidas, de ser así, Omitelas al generar el nuevo SQL, pero
mantenlas como contexto.

Es importante que analices si la nueva pregunta tiene relación con los mensajes anteriores, de no tener relación, enfócate en generar un
SQL únicamente para la nueva pregunta.

solo existen dos casos donde puedes no utilizar SQL

1) Si se te pide modificar la base de datos: debes indicar que no lo tienes permitido,

2) Si se te pide información que no está en los documentos: di que esta no es parte de la información manejada. Responde de manera
concisa exclusivamente en lenguaje natural omitiendo cualquier tipo de sintaxis SQL o información de la tabla asociada, indicando qué la
pregunta no está relacionada con la información que se encuentra
en la base de datos e invita al usuario a volver a realizar una consulta sobre la información que maneja FALP.
"""
```

Figura 7.8: Prompt experto SQL

Se puede observar en el mismo prompt como se recibe un parámetro context el cual contiene toda la información de la base de datos separada por documentos, cada documento detalla una tabla y las distintas columnas que lo componen. Estos documentos son después analizados por el RAG y por similitud de vectores, el propio RAG obtiene los que son relevantes para poder armar la consulta. Se obtienen los documentos que cumplan con los parámetros establecidos por el equipo de trabajo, como lo son cantidad de documentos y cota inferior de similitud.

A continuación se muestran los distintos documentos que se comparan con el mensaje de entrada del usuario, cabe destacar que hay más documentos que los que se encuentran en la base de datos para poder corroborar un buen funcionamiento del RAG a la hora de seleccionar los documentos relevantes

```

my_data = [
    """
    En la tabla de nombre: "pacientes"
    se almacena información referente a personas que padecen diferentes tipos de cáncer atendidas por la Fundación oncológica Arturo López Pérez.
    La información almacenada comprende las columnas:
    'id' de tipo int: esta es la clave primaria de la tabla.
    'nombre' de tipo varchar: este campo corresponde al nombre del paciente. Algunos ejemplos de entradas son: 'Juan', 'Ana', 'Pablo', 'Laura',
    todos parten con mayúscula.
    'sur' de tipo varchar: este campo corresponde al apellido del paciente. Algunos ejemplos de entradas son: 'Perez', 'Gomez', 'Martinez', todos
    parten con mayúscula.
    'tipo_cancer' de tipo varchar: corresponde a la zona afectada del paciente. Algunos ejemplos de entradas son: 'Cáncer de Mama', 'Cáncer de
    Pulmón', 'Cáncer de Colon', 'Cáncer de Prostata', todos parten con mayúscula y llevan tilde.
    'categoria_cancer' de tipo int: da un número del 1 al 5 respecto a la severidad.
    """,

    """
    En la tabla de nombre: "doctores"
    se almacena información referente a doctores que trabajan en la Fundación oncológica Arturo López Pérez.
    La información almacenada comprende las columnas:
    'id' de tipo int: esta es la clave primaria de la tabla.
    'nombre' de tipo varchar: este campo corresponde al nombre del doctor. Algunos ejemplos de entradas son: 'Juan', 'Ana', 'Pablo', 'Laura', todos
    parten con mayúscula.
    'sur' de tipo varchar: este campo corresponde al apellido del doctor. Algunos ejemplos de entradas son: 'Perez', 'Gomez', 'Martinez', todos
    parten con mayúscula.
    'especialización' de tipo varchar: este campo contiene la especialidad del doctor. Ejemplos incluyen: 'Oncología', 'Pediatría', 'Ginecología',
    todos parten con mayúscula y llevan tildes.
    'hospital' de tipo varchar: este campo contiene donde practica el doctor. Ejemplos de entradas incluyen: 'Hospital Central', 'Hospital del
    Norte' y 'Hospital del Sur', todos parten con mayúscula.
    """,

    """
    En la tabla de nombre "atenciones"
    es una tabla intermedia para las tablas 'pacientes' y 'doctores' donde se almacena información referente a qué doctor ha atendido a qué
    paciente.
    Las entradas siempre son uno a uno, vale decir en cada entrada un paciente es atendido por un doctor
    La información almacenada comprende las columnas:
    'id_atencion' de tipo int: esta es la clave primaria de la tabla.
    'id_doctor' de tipo int: es clave foránea de la tabla "doctores".
    'id_paciente' de tipo int: es clave foránea de la tabla "pacientes".
    """,

    """
    En la tabla de nombre: "enfermeros"
    se almacena información referente a los enfermeros que trabajan en la Fundación oncológica Arturo López Pérez.
    La información almacenada comprende las columnas:
    'id' de tipo int: esta es la clave primaria de la tabla.
    'nombre' de tipo varchar: este campo corresponde al nombre del enfermero. Algunos ejemplos de entradas son: 'Juan', 'Ana', 'Pablo', 'Laura',
    todos parten con mayúscula.
    'apellido' de tipo varchar: este campo corresponde al apellido del enfermero. Algunos ejemplos de entradas son: 'Perez', 'Gomez', 'Martinez',
    todos parten con mayúscula.
    'turno' de tipo varchar: este campo indica el turno de trabajo del enfermero. Algunos ejemplos de entradas son: 'Mañana', 'Tarde', 'Noche',
    todos parten con mayúscula.
    'especialización' de tipo varchar: este campo contiene la especialidad del enfermero. Ejemplos incluyen: 'Cuidados Intensivos', 'Oncología',
    'Urgencias', todos parten con mayúscula y llevan tilde.
    """,

    """
    En la tabla de nombre: "medicamentos"
    se almacena información referente a los medicamentos utilizados en el tratamiento de los pacientes de la Fundación oncológica Arturo López
    Pérez.
    La información almacenada comprende las columnas:
    'id' de tipo int: esta es la clave primaria de la tabla.
    'nombre' de tipo varchar: este campo corresponde al nombre del medicamento. Algunos ejemplos de entradas son: 'Paracetamol', 'Ibuprofeno',
    'Cisplatino', todos parten con mayúscula.
    'tipo' de tipo varchar: este campo indica la clase de medicamento. Ejemplos incluyen: 'Analgesico', 'Quimioterapia', 'Antiinflamatorio', todos
    parten con mayúscula y llevan tilde.
    'dosis' de tipo varchar: este campo indica la dosis recomendada para el tratamiento, como '500mg', '1g', '250mg'.
    'frecuencia' de tipo varchar: este campo especifica la frecuencia de administración del medicamento. Ejemplos incluyen: 'Cada 8 horas', 'Una
    vez al día', 'Cada 12 horas', todos parten con mayúscula.
    """,

    """
    En la tabla de nombre: "exámenes"
    se almacena información referente a los exámenes médicos realizados a los pacientes de la Fundación oncológica Arturo López Pérez.
    La información almacenada comprende las columnas:
    'id' de tipo int: esta es la clave primaria de la tabla.
    'nombre' de tipo varchar: este campo corresponde al nombre del examen. Algunos ejemplos de entradas son: 'Hemograma', 'Tomografía', 'Biopsia',
    todos parten con mayúscula.
    'tipo' de tipo varchar: este campo indica la categoría del examen. Ejemplos incluyen: 'Laboratorio', 'Imagenología', 'Patología', todos parten
    con mayúscula.
    'id_paciente' de tipo int: es clave foránea de la tabla "pacientes", indicando a qué paciente se le realizó el examen.
    """,

]

```

Figura 7.9: Documentos RAG

Las distintas salidas del experto SQL se ilustran a continuación:

Quiero información sobre todos los doctores que hayan atendido a pacientes con cáncer de mama

```
#Respuesta
SELECT DISTINCT d.*
FROM public.doctores d
JOIN public.atenciones a ON d.id = a.id_doctor
JOIN public.pacientes p ON p.id = a.id_paciente
WHERE p.tipo_cancer = 'Cáncer de Mama';
```

Figura 7.10: Prueba unitaria caso consulta SQL

Quiero información sobre todos los gatos que han recibido cortes de pelo en los hospitales de FALP.

```
#Respuesta
La información solicitada no está relacionada con los datos que se manejan en la base de datos de la Fundación Arturo López Pérez (FALP). La base de datos contiene información sobre pacientes con cáncer, exámenes médicos, doctores y enfermeros. No hay datos sobre gatos o servicios de peluquería animal. Por favor, realice una consulta relacionada con la información oncológica que maneja FALP.
```

Figura 7.11: Prueba unitaria caso consulta SQL no relacionada la base de datos

Identificador NL/SQL

El identificador NL/SQL, tiene como función clasificar el mensaje que entrega el Experto SQL

- Si el modelo responde “NL”, significa que la respuesta del Experto SQL fue texto en lenguaje natural, lo que indica que no era posible hacer una query de lo que se preguntaba al no tener relación con la base de datos del cliente
- Si del modelo responde “SQL”, significa que la respuesta del Experto SQL fue exclusivamente una query SQL sin ningún tipo de texto en lenguaje natural.

Esta invocación no utiliza un RAG al tener una función simple y que no requiere de nada mas que la salida del modelo anterior como entrada.

A continuación se muestra el prompt con el que trabaja el identificador NL/SQL

```
system_prompt = """
    Analiza el siguiente mensaje y determina si es completamente código SQL o si contiene lenguaje natural.

    Responde 'SQL' solo si el mensaje consiste únicamente de sintaxis SQL válida. No debe haber palabras adicionales, descripciones, explicaciones, o frases fuera del código SQL.
    Responde 'NL' si el mensaje contiene cualquier tipo de lenguaje natural, explicaciones, comentarios, o frases adicionales, aunque haya fragmentos de SQL presentes.

    Si encuentras algo que no sea código SQL puro, clasifica como 'NL'.
    Solo clasifica el mensaje únicamente con las opciones mencionadas.
    """
```

Figura 7.12: Prompt del identificador NL/SQL

Las distintas salidas del identificador NL/SQL se ilustran a continuación, cabe destacar que se hacen pruebas unitarias forzadas debido a que en particular con esta invocación hubo que reajustar el prompt muchas veces para evitar fallos en la identificación:

```
La consulta generada para responder tu pregunta es: SELECT d.hospital, COUNT(*) AS cantidad_pacientes FROM atenciones a JOIN pacientes p ON a.id_paciente = p.id JOIN doctores d ON a.id_doctor = d.id WHERE p.tipo_cancer = 'Cáncer de Mama' GROUP BY d.hospital ORDER BY cantidad_pacientes DESC LIMIT 1;; esta consulta SQL permite conocer el hospital con el mayor número de pacientes con cáncer de mama

#Respuesta
NL
```

Figura 7.13: Prueba unitaria caso SQL

```
INSERT INTO CIUDADES WHERE NAME = 'OSORNO'"

#Respuesta
SQL
```

Figura 7.14: Prueba unitaria caso NL

Corrector SQL

El corrector SQL tiene como función arreglar las consultas SQL generadas por el experto SQL que generan un error en la base de datos. Esta invocación tiene un prompt similar al Experto SQL salvo que este tiene como argumento de la función el error generado en la base de datos, por lo que tiene una noción de que es lo que falla en la query. La llamada a este modelo se encuentra dentro de un bucle que se repite mientras la query resulte con errores o no se hayan completado un número de intentos fallidos, esto para evitar caer en casos donde el propio modelo interprete mal y tome una ruta equivocada. En caso de suceder esto el flujo indica que la respuesta no pudo ser respondida e invita a contactar a un administrador.

A continuación se muestra el prompt con el que trabaja el Corrector SQL

```

system_prompt = """
La siguiente es información de tablas en una base de datos, las columnas descritas son las únicas columnas:
{context}, estas se encuentran en el esquema {esquema}. La manera de llamar a estas tablas sería {esquema}.tabla.
Además, recibirás una consulta hecha por un humano, el SQL que intenta responderla y un error generado por esta consulta.

Tu tarea es identificar por qué ocurre el error. Utilizar columnas no existentes toma precedencia ante otros errores.

La respuesta que debes dar pueden ser de dos tipos:
a) Un nuevo SQL que solucione el error y responda la pregunta. Si se requieren campos que no se encuentran en algunas de las tablas, considera que no se puede responder. Es importante que solo respondas con sintaxis SQL de postgresQL sin agregar lenguaje natural. No des explicaciones
b) En caso de que la pregunta no se pueda responder en su totalidad con la información que se te da, responde exclusivamente con un "Tu pregunta no puede ser respondida por falta de información en la base de datos 'indicar que es lo que falta'"
Se conciso en tu respuesta, responde únicamente con lo que se te indicó."""

```

Figura 7.15: Prompt del corrector SQL

Las distintas salidas del corrector SQL se ilustran a continuación:

```

consulta= "Quiero todos los doctores donde la especialidad sea Oncología"
entrada = "SELECT name FROM Doctores where especialidad = Oncología"
error = """Query 1: ERROR: column "name" does not exist LINE 5: SELECT name FROM Doctores where especialidad = Oncología;"""

LLM_Fix_SQL(consulta=consulta,query=entrada,error=error))

#Respuesta
"SELECT nombre, sur, especialización, hospital FROM doctores WHERE especialización = 'Oncología';"

```

Figura 7.16: Prueba unitaria para caso de nuevo SQL

```

consulta= "Quiero la fecha de nacimiento de los doctores con apellido Perez"
entrada = "SELECT d.fecha_nacimiento from Doctores as d where d.sur= 'Perez'"
error = """Query 1: ERROR: column "fecha_nacimiento" does not exist LINE 5: SELECT d.fecha_nacimiento from Doctores as d where d.sur= 'Perez'"""

LLM_Fix_SQL(consulta=consulta,query=entrada,error=error))

#Respuesta
'answer': "Tu pregunta no puede ser respondida por falta de información en la base de datos 'fecha_nacimiento'"

```

Figura 7.17: Prueba unitaria para caso donde no hay información en la base de datos

Traductor Data a NL

El traductor es el último modelo del flujo principal, su función es traducir la información que se recibe de la base de datos a texto en lenguaje natural, que es lo que finalmente llega al usuario. La respuesta de este modelo depende de la cantidad de tokens que tenga la respuesta de la base de datos, esto para evitar un envío de información masiva que genere un alto costo y una respuesta poco amigable para el usuario.

A continuación se muestra la lógica para los distintos tipos de respuesta y el prompt con el que trabaja el traductor data a NL

- Si la respuesta es nula por parte de la base de datos, se indica que no se encontró información sobre la pregunta hecha
- Si la información de la base de datos es mayor a una cantidad arbitraria de tokens (500 en el prototipo) se responde indicando que se obtuvo la información, pero que esta es muy extensa por lo que no se puede mostrar por mensaje, sin embargo, el usuario puede solicitarla en formato csv, Excel o generar un gráfico
- Si la información es menor a una cantidad de tokens (500) el modelo responde en lenguaje natural haciendo un análisis de la pregunta, la query y la data obtenida para así generar una respuesta lo más congruente y completa posible, indicando de todas maneras que puede solicitar obtener la información en un archivo Excel o csv, así como también generar un gráfico.

```

if len(Data) == 0: #La información es una lista vacía
    system_prompt = """
    Recibirás una pregunta, tu tarea es responder esta pregunta diciendo que no se encontró información.

    Responde "No se encontró información referente a" y agrega una frase con palabras clave de la pregunta.

    Un ejemplo de respuesta para la pregunta "quiero saber cuántas radiografías se han hecho el último año" sería "No se encontró información
    referente a radiografías hechas el último año".

    No agregues más detalles. No digas más de lo que se te indica.
    """
else:
    if len(tokens_used) < 500:
        system_prompt = """
        Recibirás una pregunta.
        La respuesta a esta pregunta se obtuvo de una base de datos, esta es: {Data}.
        La consulta que generó esta data es {query}.
        Asume que la pregunta esta respondida de manera correcta, tu tarea es responder la pregunta con la información entregada.
        Asegurate de incluir la información en la respuesta generada.
        Se conciso en tu respuesta, no respondas con mas informacion que el lenguaje natural que responda la pregunta.
        Omite que es una lista o que la lista contiene diccionarios, no menciones SQL, en ningún caso.
        Además, despues de esto, recuerda al usuario que puede pedir la información como excel o csv.
        """
    else:
        system_prompt = """
        Recibirás una pregunta, esta pregunta generó la siguiente consulta SQL: {query}.

        Tu tarea es mencionar literalmente que esta pregunta fue respondida con éxito pero que la información generada es demasiado extensa
        para un mensaje, de una forma amigable y para cualquier tipo de usuario.

        No menciones el SQL en ningún caso, responde usando solo lenguaje natural. Se conciso en tu respuesta, agregando contexto de la
        pregunta.

        Además, recuerda terminar tu mensaje indicando unicamente al usuario que puede pedir la información como excel, csv o gráfico, no
        agregues palabras extras.
        """

```

Figura 7.18: Prompt del traductor data a NL

Las distintas salidas del corrector SQL se ilustran a continuación:

```

pregunta = "Quiero todos los nombres de los doctores de apellido Sanchez?"
query = "SELECT nombre FROM doctores WHERE sur = 'Sanchez';"
datos = [{"nombre": "Marta"}]
tokens_used = model.get_num_tokens(data)

LLM_Translate_Data_to_NL(datos, pregunta, query, tokens_used):

#Respuesta
El nombre del doctor con apellido Sánchez es Marta.\n\nRecuerda que puedes solicitar esta información en formato Excel, CSV o como un gráfico si lo deseas.

```

Figura 7.19: Prueba unitaria para caso menor a 500 tokens

```

pregunta = "Dame la información de todos los pacientes"
query = "SELECT * FROM pacientes;"
dato = [...] #Lista de 200 elementos
tokens_used = model.get_num_tokens(data)#Mayor a 500 tokens

LLM_Translate_Data_to_NL(dato, pregunta, query, tokens_used):

#Respuesta
¡Claro! Tu solicitud sobre la información de todos los pacientes ha sido procesada exitosamente. Sin embargo, la cantidad de datos generados es bastante extensa para mostrarla en un solo mensaje.\n\nSi deseas acceder a esta información completa, puedes pedirla como excel, csv o gráfico.

```

Figura 7.20: Prueba unitaria para caso mayor a 500 tokens

```

pregunta = "Quiero saber el nombre de los pacientes de apellido perez han sido atendidos por la doctora Marta Sánchez"
query = "SELECT p.nombre\nFROM pacientes p\nWHERE p.sur = 'Pérez'\nJOIN atenciones a ON p.id = a.id_paciente\nJOIN doctores d ON a.id_doctor = d.id\nWHERE d.nombre = 'Marta' AND d.sur = 'Sánchez';"
data = [] # Lista de 0 elementos
tokens_used = model.get_num_tokens(data)

LLM_Translate_Data_to_NL(data, pregunta, query, tokens_used)

#Respuesta
No se obtuvo información de pacientes de apellido Pérez que hayan sido atendidos por la doctora Marta Sánchez

```

Figura 7.21: Prueba unitaria para cuando no se encuentra información en la base de datos

7.2.2. Flujo de gráficos

El flujo de gráficos corresponde a un subflujo del flujo principal, que es utilizado cuando el usuario solicita un gráfico de la información que el flujo principal genera. Dependiendo del caso, este flujo hace uso de una o dos nuevas invocaciones para poder generar un gráfico básico.

Para la implementación de este subflujo se consideran tres posibles casos:

1. Cuando no hay información previa en la conversación, donde el identificador responde directamente al usuario indicando que no hay información para generar un gráfico.
2. Cuando se posee toda la información para generar el gráfico que el usuario solicita, esto de manera interna llama al “reconocedor gráfico” el cual extrae toda la información y se la entrega a la función “generador gráfico” para el posterior armado.
3. Cuando no se posee toda la información para poder generar el gráfico. aquí de manera interna se llama al generador gráfico, pero este en su campo de “falta de datos” arroja un valor “True” lo que dirige la consulta al Experto SQL orientado a gráficos el cual genera una nueva consulta y la cual es procesada como si fuese el flujo principal, para obtener así la data necesaria para generar el gráfico. posterior a eso se hace un nuevo llamado al reconocedor gráfico el cual ahora sí tiene toda la información y le pasa los parámetros a la función “generador gráfico” para generarlo.

Recolector

El recolector gráfico tiene como función generar un JSON con toda la información importante para poder generar el gráfico que solicita el usuario, si es que el usuario no provee todos los parámetros del gráfico, estos son asumidos a juicio del modelo u omitidos dependiendo de cual sea el parámetro.

Dentro de los parámetros que se encuentran en el JSON están:

- color.
- título.
- nombre eje x.
- nombre eje y.
- tipo de gráfico.
- falta de datos.
- explicación.

Como se puede ver en los parámetros del JSON, existe un parámetro de 'falta de datos', el cual funciona como clasificador para saber si el gráfico puede ser generado directamente, o es necesario obtener más información de la base de datos para poder generarlo.

A continuación se muestra el prompt con el que trabaja el Recolector

```
system_prompt = """
Aquí está el contenido de un archivo CSV. El archivo tiene las siguientes columnas: {Data}.
Identifica qué columnas son gráficas.

Por favor, devuélveme en formato JSON las columnas y el tipo de gráfico adecuado en la siguiente estructura, no necesito mas informacion que el JSON y no escribas 'JSON' en tu respuesta, es importante incluir el tipo de grafico, el eje x y el eje y.
titulo y color son solo si el usuario lo indica:

{{
  "tipo_grafico": "scatter/line/bar/hist",
  "x_col": "nombre_columna_x",
  "x_col_name": "Nombre que el usuario desea para la columna x",
  "y_col": "nombre_columna_y",
  "y_col_name": "Nombre que el usuario desea para la columna y",
  "color": "color en ingles",
  "titulo": "titulo",
  "faltan_datos": "true"/"false"
}}

x_col e y_col son los nombres de la columnas en el dataframe.
Recuerda que el JSON es la unica informacion que necesito, no escribas nada mas que el JSON sugerido, es importante considerar peticiones antiguas sobre el grafico, y realizar los cambios a los parametros indicados manteniendo las indicaciones que no sufrieron modificaciones.
Omite explicaciones a toda costa.
En caso de que {question} indique alguna modificacion al grafico que pretende obtener, es tu tarea incluir esa modificacion siempre que esta sea posible con {Data}.

Campos como color titulo o y_col no los incluyas si es que no se mencionan
El campo faltan_datos no puede faltar, puede ser "true" si es que necesitas mas informacion para poder generar una consulta, o "false" si con la informacion que tienes puedes generar el grafico que se te solicita.
El campo explicacion aparece solo si el campo faltan_datos es true y aqui es tu trabajo colocar toda la explicacion, de manera concisa, de porque no se puede generar el grafico.
"""
```

Figura 7.22: Prompt del recolector

Las distintas salidas del recolector se ilustran a continuación:

En el primer caso, se solicita un gráfico de información que ya ha sido obtenida en la base de datos, por lo que no es necesario volver a hacer una consulta para generar el gráfico

```
#Pregunta inicial
me darias la cantidad de doctores que trabaja en cada hospital

#Respuesta
Aquí tienes la cantidad de doctores que trabajan en cada hospital: el Hospital Central cuenta con 24 doctores, el Hospital del Norte tiene 24 doctores, el Hospital del Este emplea a 23 doctores, y el Hospital del Sur cuenta con 24 doctores. Recuerda que, si lo deseas, puedes solicitar esta información en formato Excel o CSV.

#Petición Grafico
Necesito un grafico

#Respuesta
Tu archivo ya esta listo, indica si requieres algun cambio en tu grafico para volver a generarlo
```

Figura 7.23: Prueba unitaria solicitud de gráfico con información

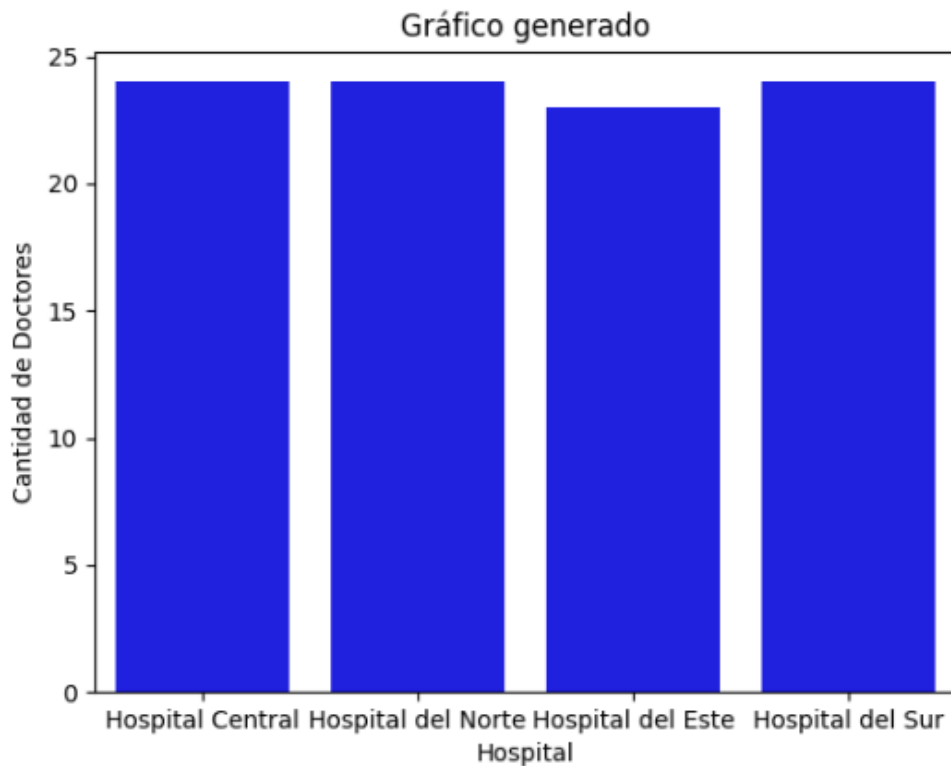


Figura 7.24: Gráfico generado a partir de información recolectada

```
#Petición de Modificación
"quiero que el grafico sea de lineas rojas, que de titulo tenga 'CASO 1', el eje x sea 'Sedes en FALP', y el eje y 'Numero de doctores'"
```

Figura 7.25: Solicitud de nuevo gráfico

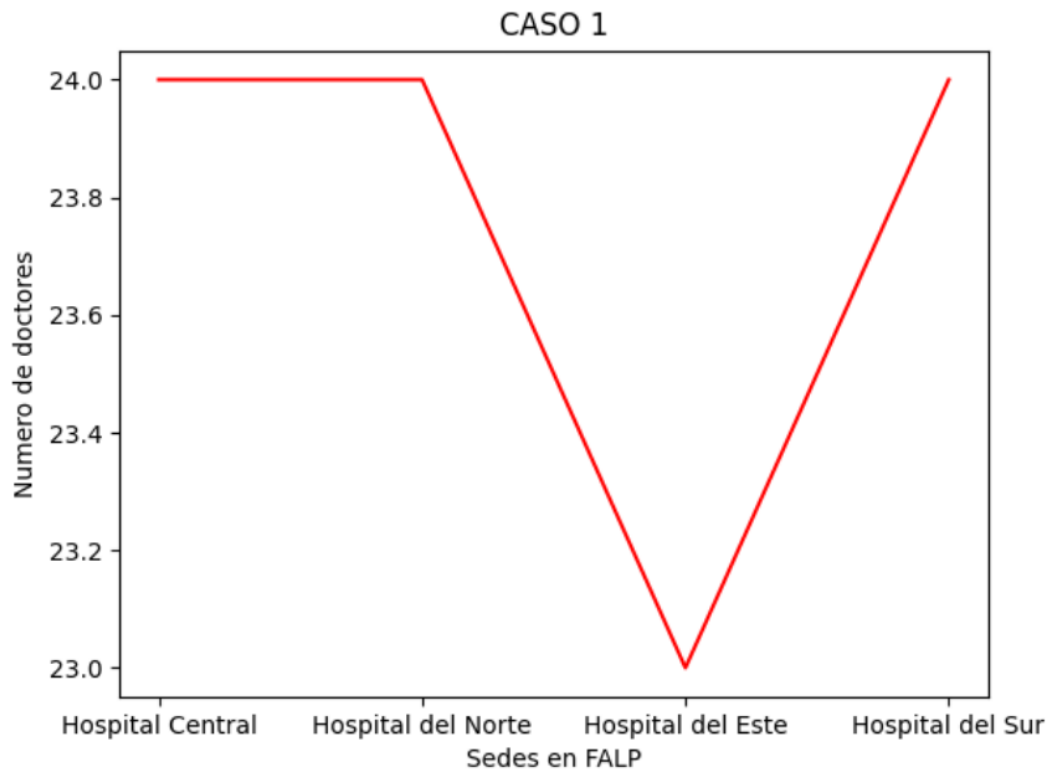


Figura 7.26: Nuevo gráfico generado a partir de la ultima solicitud y la información previa

En el segundo y último caso, se solicita un gráfico con información adicional a la que fue extraída de la base de datos, por lo que es necesario volver a hacer una consulta para generar el gráfico, en este caso se piden los pacientes con cáncer de mama por hospital, cuando de la base de datos solo se extrajo cuántos pacientes hay por hospital.

```
#Pregunta inicial
cuantos pacientes han sido atendidos en los distintos hospitales

#Respuesta
"Según la información proporcionada, los pacientes atendidos en los distintos hospitales son:\n\n- Hospital Central: 50 pacientes\n- Hospital del Este: 40 pacientes\n- Hospital del Norte: 51 pacientes\n- Hospital del Sur: 43 pacientes\n\nRecuerde que puede solicitar esta información en formato Excel o CSV si lo desea."
```

```
#Petición Gráfico
Quiero un grafico de la cantidad de pacientes con cancer de mama por hospital

#Respuesta
Tu archivo ya esta listo, indica si requieres algun cambio en tu grafico para volver a generarlo
```

Figura 7.27: Prueba unitaria solicitud de gráfico sin información completa

Se puede confirmar que no se tenía toda la información por la propia salida del recolector en donde el parámetro falta data se asignó como "True".

```
{
  "tipo_grafico": "bar",
  "x_col": "hospital",
  "x_col_name": "Hospital",
  "y_col": "pacientes_atendidos",
  "y_col_name": "Pacientes atendidos",
  "faltan_datos": true,
  "explicacion": "No hay información específica sobre pacientes con cáncer de mama en las columnas disponibles. La columna 'pacientes_atendidos' podría incluir pacientes con diversas condiciones, no solo cáncer de mama. Se necesitaría una columna adicional que especifique el tipo de cáncer o filtrar los datos para pacientes con cáncer de mama."
}
```

Figura 7.28: Salida del recolector con parámetro de falta de datos asignado como True, indicando la falta de información para generar el gráfico

De esto se debe completar un flujo principal completo con la leve modificación del Experto SQL-Gráfico, modelo que es explicado en la siguiente sección.

Una vez se obtiene toda la información, se obtiene el nuevo gráfico.

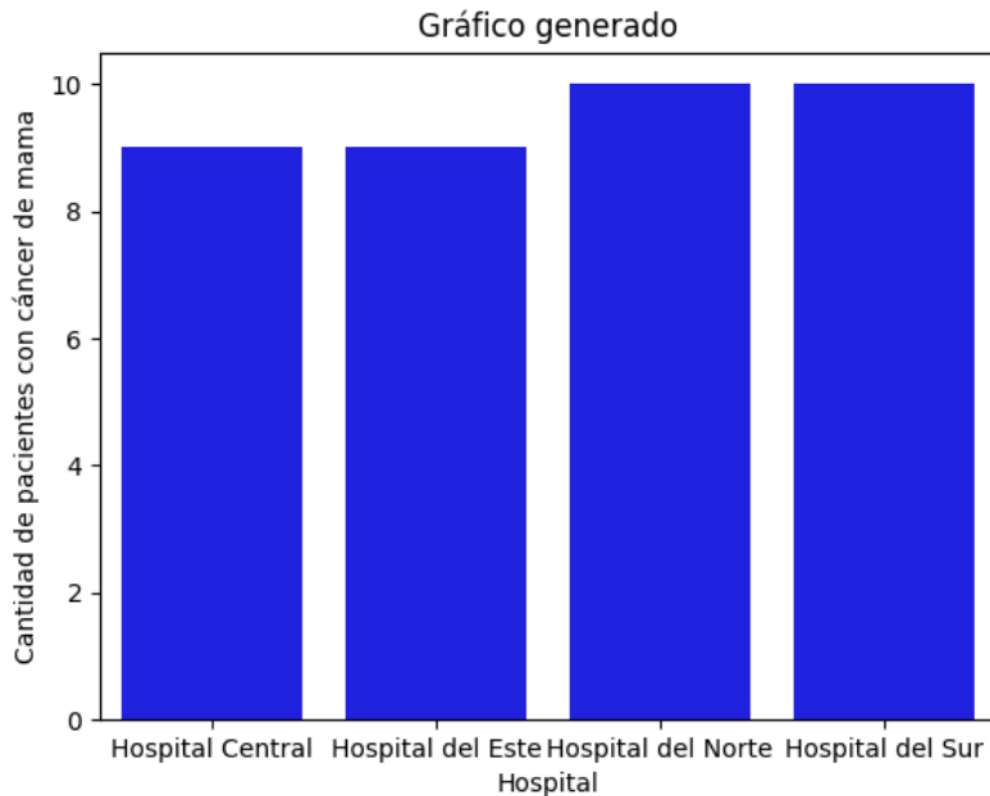


Figura 7.29: Gráfico generado a partir de la nueva información recolectada

Experto SQL-Gráfico

El Experto SQL-Gráfico tiene como funcionalidad la misma que el Experto SQL original, la cual es generar un SQL en base a la pregunta del usuario. Este modelo es solo accedido cuando el usuario solicita un gráfico con información que no se encuentra en la data obtenida en la misma conversación.

La diferencia entre este modelo y el Experto SQL del flujo principal es que este dentro de su prompt considera todo lo de la conversación que ya se ha tenido y pese a recibir la petición generar un gráfico, solo considera la información que puede ser traspasada a consulta SQL.

Tomaremos el mismo ejemplo anterior y mostraremos la salida donde se genera la consulta para evitar repetición de imágenes.

```
#Pregunta
Quiero un grafico de la cantidad de pacientes con cancer de mama por hospital

#Respuesta
"SELECT d.hospital, COUNT(DISTINCT a.id_paciente) as cantidad_pacientes\nFROM public.atenciones a\nJOIN public.doctores d ON a.id_doctor = d.id\nJOIN public.pacientes p ON a.id_paciente = p.id\nWHERE p.tipo_cancer = 'C\u00e1ncer de Mama'\nGROUP BY d.hospital;"
```

Figura 7.30: Prueba unitaria para Experto SQL-Gráfico donde pese a petición de gráfico, se genera una consulta SQL

Generador gráfico

Esta es una función que toma el JSON generado por el reconocedor gráfico para armar el gráfico que solicita el usuario, donde se puede personalizar el tipo de gráfico como: barra, líneas, dispersión o histograma, el valor de los ejes, el título, y el color de las barras, puntos o líneas.

```
def generar_grafico(csv_path, data_dict: dict = {}):
    color = data_dict.get("color", "blue")
    titulo = data_dict.get("titulo", "Gráfico generado")
    x_col_name = data_dict.get("x_col_name", "Eje X")
    y_col_name = data_dict.get("y_col_name", "Eje Y")
    tipo_grafico = data_dict.get("tipo_grafico", "scatter")
    x_col = data_dict.get("x_col")
    y_col = data_dict.get("y_col")

    df = pd.read_csv(csv_path)
    plt.figure()

    if titulo:
        plt.title(titulo)

    if tipo_grafico == 'scatter':
        fig = sns.scatterplot(data=df, x=x_col, y=y_col, color=color)
    elif tipo_grafico == 'line':
        fig = sns.lineplot(data=df, x=x_col, y=y_col, color=color)
    elif tipo_grafico == 'bar':
        fig = sns.barplot(data=df, x=x_col, y=y_col, color=color)
    elif tipo_grafico == 'hist':
        fig = sns.histplot(df[x_col], color=color)
    else:
        raise ValueError(f"Tipo de gráfico '{tipo_grafico}' no soportado.")
    fig.set(xlabel=x_col_name, ylabel=y_col_name)
    x = random.randint(0, 100000)
    save_results_to = settings.temp_files
    archivo = plt.savefig(save_results_to + str(x) + ".png")
    return archivo, x
```

Figura 7.31: Función generador grafico completa

7.2.3. Limitaciones actuales del flujo

Actualmente el modelo tiene problemas de memoria referente a recibir peticiones de manera fraccionada, por lo que si se pide la información en diferentes inputs del usuario, no siempre se obtiene el gráfico con todas las indicaciones.

7.2.4. Diagrama de flujo

La manera en que las diferentes invocaciones de modelos se comunican entre sí, así como el flujo general de entradas, salidas y dependencias entre componentes, se resume en la figura [7.32](#). Este diagrama muestra la arquitectura lógica del sistema de modelos conversacionales desarrollado, representando cómo cada modelo recibe la información (ya sea del usuario o de otro modelo), y qué tipo de salidas se generan para cada llamado de estos.

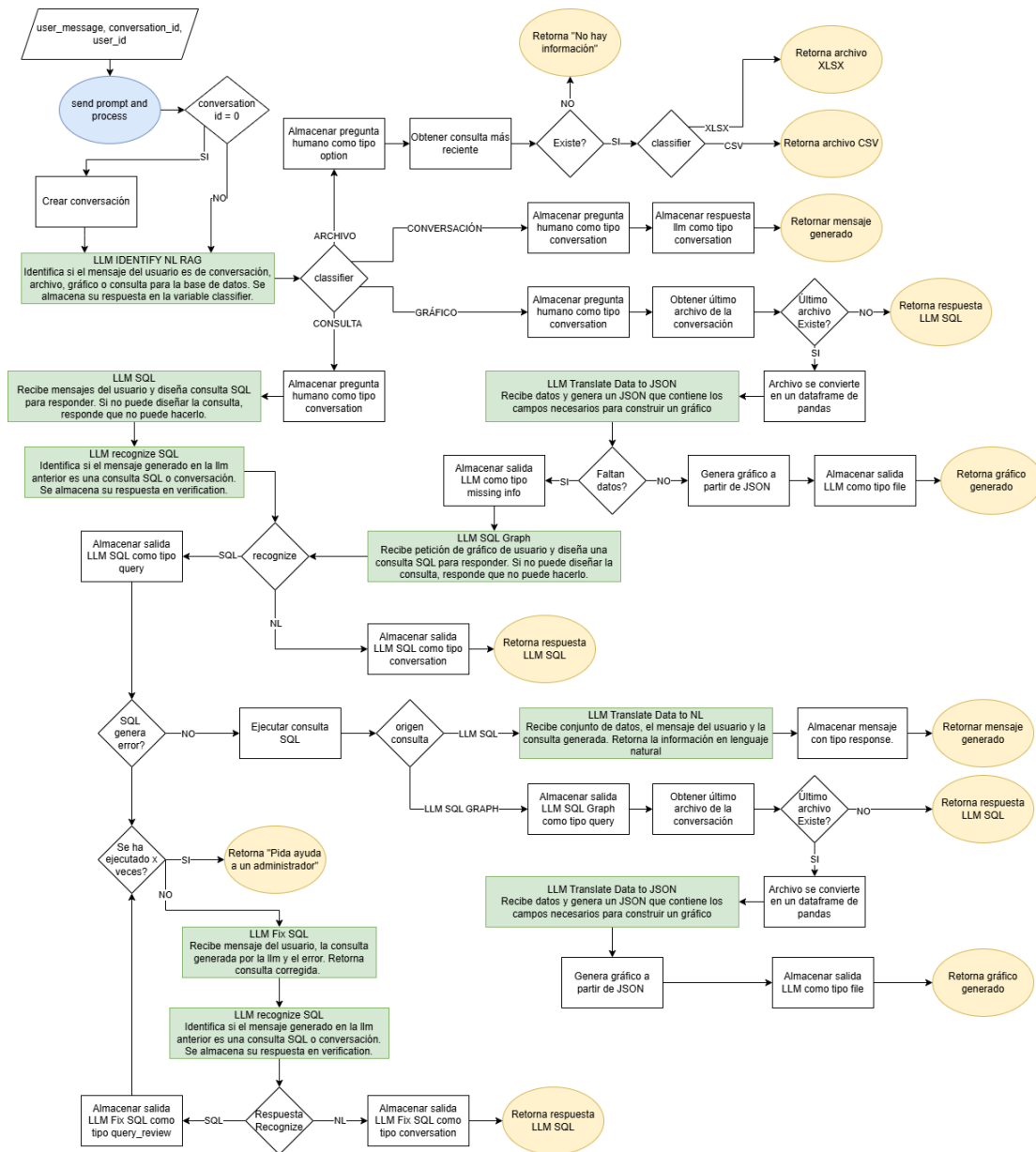


Figura 7.32: Diagrama de flujo prototipo

7.2.5. Comparativa de modelos de lenguaje

Diferencias en desempeño

En la figura 7.2 se presenta una pequeña tabla comparativa entre modelos de la familia Claude que el equipo tiene disponibles.

Tarea	Claude 3.5	Claude 3.0	Claude 3 Haiku
Identificador General	Sin fallos	Falla en archivos	Falla en archivos y consulta
Experto SQL	Sin fallos	Con fallo en obtención de tablas	Con fallo en parámetros a obtener
Identificador SQL/NL	Sin fallos	Sin fallos	Sin fallos
Corrector SQL	Sin fallos	Reconoce lo que no está, falla en comprender que la pregunta no puede ser respondida	Cae en el mismo error de incluir información que no existe en las tablas
Traductor data a NL	Sin fallos	Sin fallos	Sin fallos
Reconocedor Gráfico	Sin fallos	Sin fallos	Sin fallos
Experto SQL Gráfico	Sin fallos	Sin fallos	Fallo en entendimiento de tablas

Cuadro 7.2: Comparativa modelos familia Claude

Si bien Claude 3.5 presenta un mejor desempeño que Claude 3.0 y Claude 3 Haiku, su uso conlleva un costo mayor, tanto de entrada como de salida, en comparación con este último.

En este contexto, resulta útil identificar en qué tareas es posible reemplazar a Claude 3.5 por otros modelos, particularmente Claude 3 Haiku, con el fin de reducir el gasto

asociado a las invocaciones al sistema. Aunque Claude 3 también puede considerarse dentro de este análisis, su rendimiento inferior frente a Claude 3.5 lo hace menos atractivo como alternativa directa en tareas exigentes, más aún considerando que sus costos son idénticos a los de su versión posterior.

Diferencias de costos

En una conversación promedio que contempla preguntas y solicitudes para generar archivos, se utilizan en promedio 4500 tokens de entrada y 200 tokens de salida.

El cobro de los distintos modelos está dado por un millón de tokens, por lo que se puede tener en un cantidad estimada de 222 conversaciones antes de llegar al millón de tokens de entrada. Esta cantidad de conversaciones resultarían en 44.000 tokens de salida.

La tabla 7.3 muestra la diferencia de precios para cantidad de conversaciones en base a la cantidad de tokens promedio de la plataforma.

Modelo	Costo por millón de tokens de entrada	Costo por millón de tokens de salida	Costo entrada + salida cada 100 conversaciones	Costo entrada + salida cada 222 conversaciones
Claude 3.5	\$3	\$15	\$1.65	\$3.66
Claude 3	\$3	\$15	\$1.65	\$3.66
Claude 3 Haiku	\$0.25	\$1.25	\$0.13	\$0.31

Cuadro 7.3: Costos por modelo

Diferencias generales Las diferencias generales entre los 3 modelos a nivel de respuestas son:

- Claude 3 Haiku está especializado para tareas más simples y que requieran de velocidad y útil en escenarios donde los recursos son limitados
- Claude 3 en comparación a modelos como Claude 2 es mucho más hábil para responder de manera más precisa a tareas complejas pero es un poco mas lento

en comparación a los otros dos modelos

- Claude 3.5 es mas rápido que Claude 3 y tiene mejor desempeño para interacciones y contextos más prolongados, lo que hace que sea sumamente efectivo para este proyecto y se mantenga como solución definitiva para las partes del flujo que involucren tareas complejas.

7.2.6. Comparativa de modelos de embeddings

Diferencias en desempeño

En las siguientes secciones se presentan los resultados obtenidos a partir de los diferentes modelos de embeddings probados por el equipo, estos son: titan-embed-text-v1 y titan-embed-text-v2 de Amazon, además de uno de los mejores modelos multilingües disponibles en HuggingFace, multilingual-e5-large y uno que está al borde del top 100, all-mpnet-base-v2, el cual se usa como baseline.

En cuanto al desempeño de los modelos, en las tablas 7.4 y 7.5 se muestran el desempeño de los diferentes modelos en cuanto al número de falsos negativos (porcentaje de documentos que debieron ser retornados, pero no lo fueron) y el número de falsos positivos (porcentaje de documentos que no debieron ser retornados pero lo fueron).

Cabe destacar que las columnas con nombre “Tasa de falsos X*” corresponden a los valores alcanzados una vez se ajusta el radio de búsqueda para lograr que los falsos negativos se vuelvan 0.

Modelo	Tasa falsos negativos	Tasa falsos negativos*
titan-embed-text-v1	14,29 %	0 %
titan-embed-text-v2	21,43 %	0 %
all-mpnet-base-v2	0 %	-
multilingual-e5-large	0 %	-

Cuadro 7.4: Tasa de falsos negativos por modelo.

Modelo	Tasa falsos positivos	Tasa falsos positivos*
titan-embed-text-v1	47,83 %	69,57 %
titan-embed-text-v2	31,25 %	43,75 %
all-mpnet-base-v2	60 %	-
multilingual-e5-large	58,82 %	-

Cuadro 7.5: Tasa de falsos positivos por modelo.

Por otro lado, en la tabla 7.6 se muestran los tiempos de ejecución total por modelo.

Modelo	prom(tiempos ejecución total)	Desv est(tiempos ejecución total)
titan-embed-text-v1	4,2511	0,8869
multilingual-e5-large	4,6428	1,6849
all-mpnet-base-v2	4,8374	1,1615
titan-embed-text-v2	4,8665	1,1793

Cuadro 7.6: Tasa de tiempos de ejecución por modelo.

Diferencias de costos

A continuación, se describen los precios por modelo cada 1000 tokens transformados a embeddings. Esto se muestra en la tabla 7.7. Cabe recordar que los modelos de HuggingFace se pueden acceder de manera completamente gratuita.

Modelo	Precio por 1000 tokens
Titan Text Embeddings	0,0001 USD
Titan Text Embeddings V2	0,00002 USD
multilingual-e5-large	Gratuito
all-mpnet-base-v2	Gratuito

Cuadro 7.7: Costos cada 1000 tokens por modelo.

Diferencias generales

Con la información entregada, se ve que ambas versiones de Titan Text Embeddings de Amazon presentan un buen desempeño y, junto a multilingual-e5-large, destacan en aspectos diferentes.

En caso que los tiempos de ejecución sean la mayor prioridad en el desarrollo, se recomienda el uso de Titan Text Embeddings v1. Mientras que, si se tiene conocimiento de valores esperados en cuanto a distancias, la segunda versión del modelo de Amazon es una mejor alternativa, este ayuda a retornar menos documentos que actúan como ruido y permiten un mejor desempeño del modelo de lenguaje. Por otro lado, si el presupuesto para la generación de embeddings es una restricción principal, se recomienda el uso de multilingual-e5-large, el cual tiene buenos resultados de manera gratuita.

Cualquiera sea el caso que se escoja, siempre es recomendable almacenar los embeddings generados con fin de no transformar más tokens de lo necesario y malgastar recursos, esto es un trabajo futuro de interés para el proyecto.

7.3. Casos de uso

A continuación, se detallan tres casos de uso principales relacionados con los flujos de modelos y su representación en la plataforma. Cada caso incluye imágenes extraídas de la interfaz de la plataforma, del código en Visual Studio y de TablePlus, el cliente utilizado para visualizar la información de la base de datos.

7.3.1. Caso 1: Petición de consulta SQL a la plataforma con información que si se encuentra en la base de datos

Este caso de uso describe el flujo de interacción entre un usuario y la plataforma cuando el usuario realiza una consulta en lenguaje natural, y la información solicitada está almacenada en la base de datos. La plataforma traduce la consulta al lenguaje SQL, ejecuta la consulta en la base de datos y presenta los resultados al usuario de manera comprensible. A continuación se detalla el flujo del caso de uso.

1. Se recibe el mensaje del usuario, lo que activa la función `send prompt and process`, identificando que se inicia una nueva conversación.



Figura 7.33: Mensaje recibido en la plataforma al iniciar una nueva conversación

2. Se ingresa al modelo identificador general, que clasifica el mensaje como tipo `consulta`, con salida SQL según los prompts definidos.

```
39 #RECOGNIZE
40 texto= "Me das todos los datos de los doctores de apellido Molina?"
41 d= LLM_Identify_NL_RAG(texto)
42 print(d)
43
44
```

PROBLEMS 11 OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
{{'answer': 'SQL'}, 1)
```

Figura 7.34: Clasificación del mensaje como SQL por el identificador general

3. Se almacena la pregunta del usuario en la base de datos como tipo `conversation`.

Search for field...	≡
id	int8
26	▼
conversation_id	int8
25	▼
message	json
{ "role": "user", "content": "Me das todos los datos de los doctores de apellido Molina?" }	▼
type	varchar
conversation	▼

Figura 7.35: Registro del mensaje del usuario en la base de datos

4. El mensaje pasa al modelo experto SQL, el cual genera una consulta siempre que esté relacionada con la base de datos del cliente.

```
41 #Experto_SQL
42 texto= "Me das todos los datos de los doctores de apellido Molina?"
43 d= LLM_SQL(texto, [])
44 print(d)
45
46
```

PROBLEMS 11 OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
{{'answer': "SELECT * \nFROM public.doctores\nWHERE sur = 'Molina';"}, 16)
```

Figura 7.36: Consulta SQL generada por el modelo experto

5. La salida del experto es evaluada por el modelo identificador SQL/NL, que la

clasifica como SQL.

```
46 texto = "SELECT * \nFROM public.doctores\nWHERE sur = 'Molina';"  
47 e = LLM_recognize_SQL(texto)  
48 print(e)
```

PROBLEMS 11 OUTPUT DEBUG CONSOLE TERMINAL PORTS

2025-07-08 19:05:19,740 [INFO]: Inicializando LOGGER
content='SQL' additional_kwargs={'usage': {'prompt_tokens': 184, 'completion_t

Figura 7.37: Clasificación de la consulta como SQL por el identificador SQL/NL

6. Se almacena la salida del experto como tipo `query`, y se ejecuta en la base de datos sin generar errores.

Search for field...

id	int8
27	▼
conversation_id	int8
25	▼
message	json
{ "role": "assistant", "content": { "query": "SELECT * FROM public.doctores WHERE sur = 'Molina';", "file_id": 6 } }	
type	varchar
query	▼

Figura 7.38: Consulta SQL almacenada correctamente en la base de datos

```
SELECT * FROM public.doctores WHERE sur = 'Molina';|
```

line 758, column 52, location 29909

id	nombre	sur	especialización	hospital
23	Fernando	Molina	Cardiología	Hospital del Sur
88	Silvia	Molina	Cardiología	Hospital del Este

Figura 7.39: Resultados obtenidos tras ejecutar la consulta

7. Dado que la consulta fue exitosa, se envía al modelo traductor `data to NL`, que usa los datos obtenidos, el mensaje del usuario y la consulta para generar una respuesta en lenguaje natural. Esta se almacena como tipo `response`.

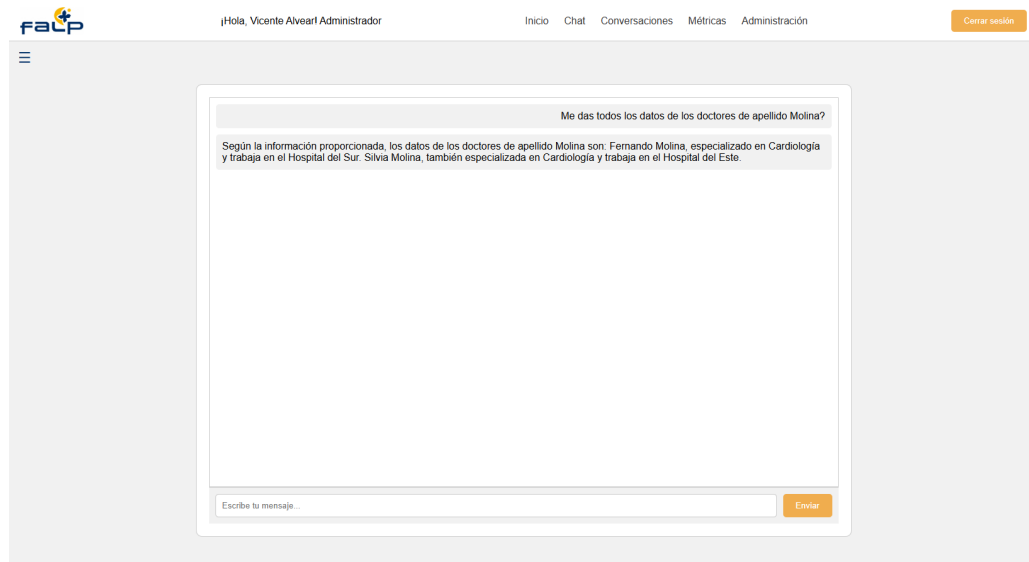


Figura 7.40: Respuesta en lenguaje natural generada por el modelo traductor

8. Finalmente, se retorna la respuesta generada al usuario dentro del mismo chat.

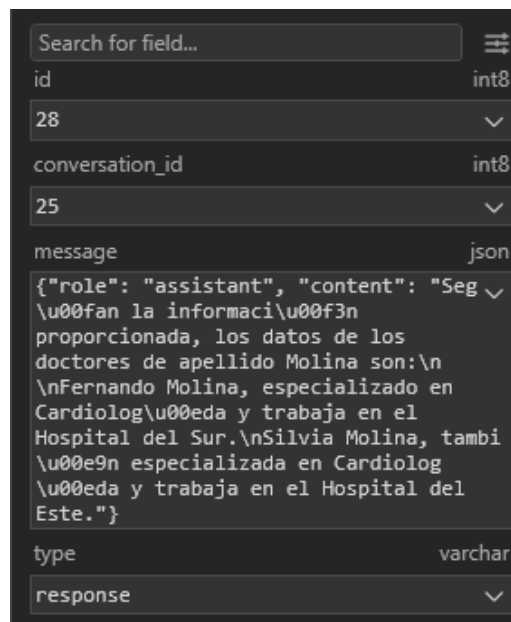


Figura 7.41: Respuesta almacenada como tipo `response` en la base de datos

El flujo completo para este caso de uso se puede observar con una flecha de color azul en [7.58](#).

7.3.2. Caso 2: Petición de gráfico con información que fue obtenida en la base de datos

Este caso de uso describe el flujo de interacción entre un usuario y la plataforma cuando el usuario solicita un gráfico de información que se preguntó en un inicio.

1. Se recibe el mensaje del usuario, lo que activa la función `send prompt and process`, identificando inmediatamente que estamos en una conversación ya existente.

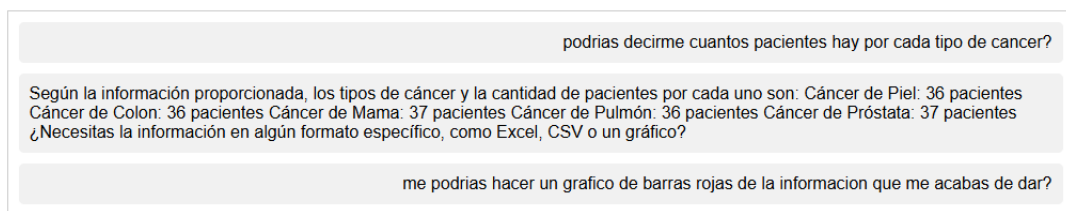


Figura 7.42: Vista de la plataforma al iniciar el caso de uso 2

2. El mensaje se clasifica como tipo `graph` por el modelo identificador general.

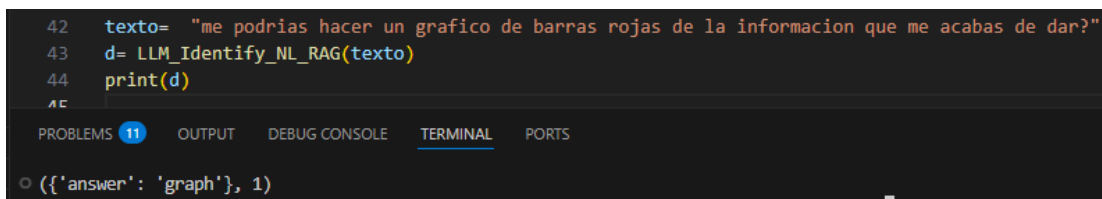


Figura 7.43: Clasificación del mensaje como `graph`

3. La pregunta del usuario se almacena en la base de datos como tipo `conversación`.

A screenshot of a database record form. At the top is a search bar labeled "Search for field...". Below it are several fields with dropdown menus. The fields are: "id" (type "int8") with value "76"; "conversation_id" (type "int8") with value "64"; "message" (type "json") with value '{"role": "user", "content": "me podrias hacer un grafico de barras rojas de la informacion que me acabas de dar? "}'; and "type" (type "varchar") with value "conversation".

Field	Type	Value
id	int8	76
conversation_id	int8	64
message	json	{"role": "user", "content": "me podrias hacer un grafico de barras rojas de la informacion que me acabas de dar? "}
type	varchar	conversation

Figura 7.44: Registro del mensaje como tipo `conversación`

4. Se recupera el último archivo de la conversación y se verifica su existencia. En este caso, sí existe.

A screenshot of a database record form. At the top is a search bar labeled "Search for field...". Below it are several fields with dropdown menus. The fields are: "id" (type "int8") with value "17"; "user_id" (type "int8") with value "1"; "conversation_id" (type "int8") with value "64"; "name" (type "varchar") with value "82239"; "created_at" (type "timestampz") with value "2025-07-09 23:18:59.831725+00"; "expires_at" (type "timestampz") with value "2025-07-12 23:18:59.831725+00"; and "extension" (type "varchar") with value "csv".

Field	Type	Value
id	int8	17
user_id	int8	1
conversation_id	int8	64
name	varchar	82239
created_at	timestampz	2025-07-09 23:18:59.831725+00
expires_at	timestampz	2025-07-12 23:18:59.831725+00
extension	varchar	csv

Figura 7.45: Archivo más reciente encontrado en la conversación

5. El archivo se convierte en un `DataFrame` usando la biblioteca `pandas`, y se envía al modelo recolector. Este, dentro de sus parámetros en formato `JSON`, indica que no faltan datos para construir el gráfico.

```
{'tipo_grafico': 'bar',  
  'x_col': 'tipo_cancer',  
  'x_col_name': 'Tipo de Cáncer',  
  'y_col': 'cantidad_pacientes',  
  'y_col_name': 'Cantidad de Pacientes',  
  'color': 'red',  
  'faltan_datos': 'false'}
```

Figura 7.46: Validación del archivo por el modelo recolector mediante estructura `JSON`

6. Se ejecuta la función `generador_gráfico`, que recibe el `JSON` y genera el gráfico correspondiente. La salida se almacena como tipo `file`.

Search for field...		≡
id	int8	
77		▼
conversation_id	int8	
64		▼
message	json	
{ "role": "assistant", "content": { "text": "Tu archivo ya esta listo, indica si requieres alg\u00fan cambio en tu gráfico para volver a generarlo.", "graphical_option": { "tipo_grafico": "bar", "x_col": "tipo_cancer", "x_col_name": "Tipo de C\u00e1ncer", "y_col": "cantidad_pacientes", "y_col_name": "Cantidad de Pacientes", "color": "red", "faltan_datos": "false"}, "file_id": 18, "file_type": "png"} } } }		
type	varchar	
file		▼

Figura 7.47: Ejecución del generador gráfico y almacenamiento de salida

7. Finalmente, se muestra en la plataforma un mensaje indicando que el gráfico fue generado, con la opción para descargarlo como archivo `PNG`.

podrias decirme cuantos pacientes hay por cada tipo de cancer?

Según la información proporcionada, los tipos de cáncer y la cantidad de pacientes por cada uno son: Cáncer de Piel: 36 pacientes
Cáncer de Colon: 36 pacientes Cáncer de Mama: 37 pacientes Cáncer de Pulmón: 36 pacientes Cáncer de Próstata: 37 pacientes
¿Necesitas la información en algún formato específico, como Excel, CSV o un gráfico?

me podrias hacer un grafico de barras rojas de la informacion que me acabas de dar?

Figura 7.48: Mensaje al usuario con confirmación de gráfico generado

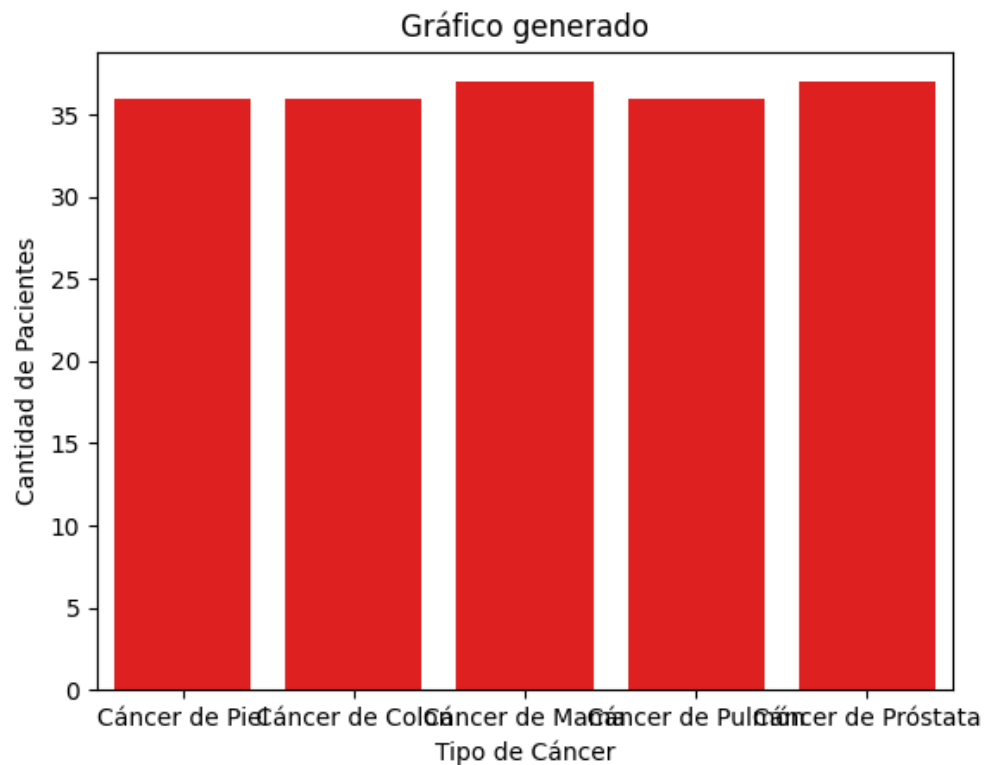


Figura 7.49: Opción de descarga del gráfico en formato PNG

El flujo completo para este caso de uso se puede observar con una flecha de color rojo en [7.58](#).

7.3.3. Caso 3: Petición de generación de archivo en formato `xlsx`

Este caso de uso describe el flujo de interacción entre un usuario y la plataforma cuando el usuario solicita su contenido en un formato en específico; en este caso un archivo con extensión `xlsx`. La plataforma recupera la consulta más reciente y convierte la información en el formato solicitado.

Flujo

1. Se recibe el mensaje del usuario, lo que activa la función `send prompt and process`, identificando inmediatamente que estamos en una conversación ya existente.

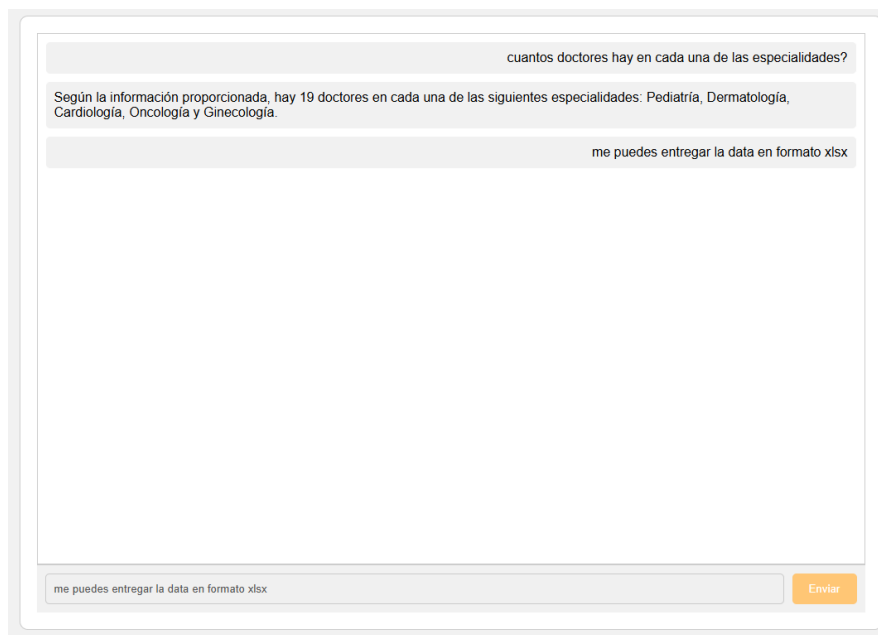


Figura 7.50: Vista de la plataforma al recibir un mensaje en una conversación existente

2. El mensaje es procesado por el modelo identificador general, que lo clasifica como tipo `archivo`, con salida `xlsx`.

```
42 texto= "me puedes entregar la data en formato xlsx"
43 d= LLM_Identify_NL_RAG(texto)
44 print(d)
45
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
({'answer': 'xlsx'}, 1)
```

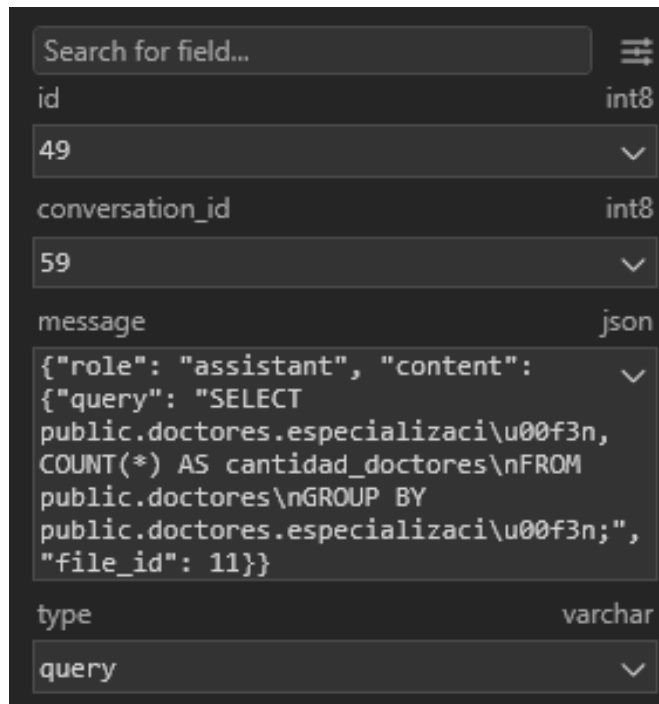
Figura 7.51: Clasificación del mensaje como `xlsx` por el identificador general

3. Se almacena la solicitud del usuario como tipo `option` en la base de datos.

id	int8
51	▼
conversation_id	int8
59	▼
message	json
{ "role": "user", "content": "me puedes entregar la data en formato xlsx" }	
type	varchar
option	▼

Figura 7.52: Registro de la opción seleccionada en la base de datos

4. Se busca la última consulta ejecutada dentro de la misma conversación (último archivo tipo `query` asociado al `conversation_id`).



Search for field...

id int8
49

conversation_id int8
59

message json
{"role": "assistant", "content": {"query": "SELECT public.doctores.especializaci\u00f3n, COUNT(*) AS cantidad_doctores FROM public.doctores GROUP BY public.doctores.especializaci\u00f3n;", "file_id": 11}}

type varchar
query

Figura 7.53: Consulta previa recuperada desde la conversación

5. El mensaje se reclasifica como tipo `xlsx` según la opción seleccionada por el usuario.



cuantos doctores hay en cada una de las especialidades?

Según la información proporcionada, hay 19 doctores en cada una de las siguientes especialidades: Pediatría, Dermatología, Cardiología, Oncología y Ginecología.

me puedes entregar la data en formato xlsx

me puedes entregar la data en formato xlsx

Enviar

Figura 7.54: Confirmación de la clasificación final del mensaje como `xlsx`

6. Se genera y almacena el archivo solicitado. El usuario recibe un mensaje indicando que el archivo está listo para descargar. Esta respuesta se registra como tipo `file`.

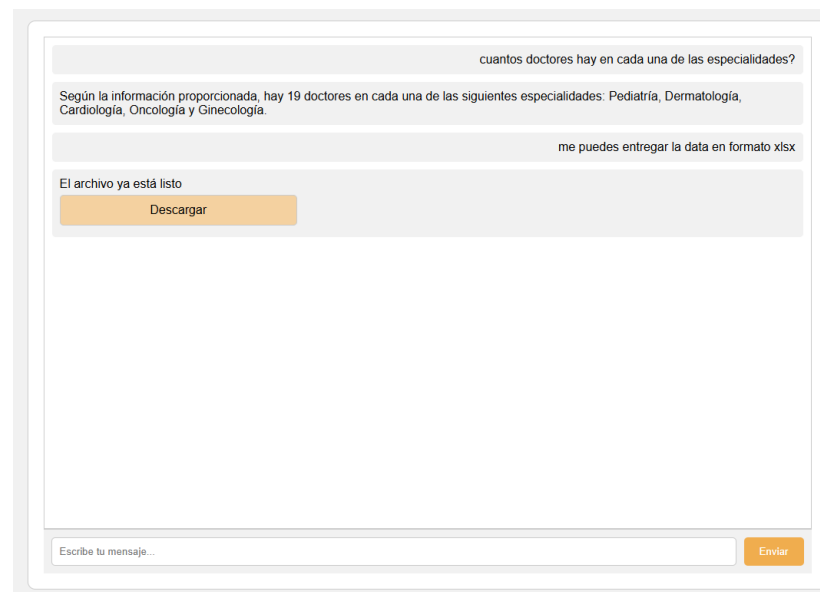


Figura 7.55: Mensaje de confirmación de creación del archivo

Unnamed: 0	especialización	cantidad_doctores
0	Pediatría	19
1	Dermatología	19
2	Cardiología	19
3	Oncología	19
4	Ginecología	19

Figura 7.56: Interfaz con opción de descarga del archivo generado

A screenshot of a database record form. At the top is a search bar labeled "Search for field...". Below it are several input fields with labels and data types on the right. The fields are: "id" (int8) with value "52", "conversation_id" (int8) with value "59", "message" (json) with a JSON object, "type" (varchar) with value "file", and "file" (empty). The JSON object in the "message" field is: {"role": "assistant", "content": {"text": "El archivo ya est\u00e1 listo", "file_id": 12, "file_type": "xlsx"}}.

Field	Type	Value
id	int8	52
conversation_id	int8	59
message	json	{"role": "assistant", "content": {"text": "El archivo ya est\u00e1 listo", "file_id": 12, "file_type": "xlsx"}}
type	varchar	file
file		

Figura 7.57: Registro del archivo generado como tipo `file` en la base de datos

El flujo completo para este caso de uso se puede observar con una flecha de color rojo en [7.58](#).

A continuación, se muestra el flujo dentro del prototipo de diagrama de flujo general de los modelos, que sigue los distintos casos de uso previamente descritos.

Caso 1: Camino azul

Caso 2: Camino amarillo

Caso 3: Camino rojo

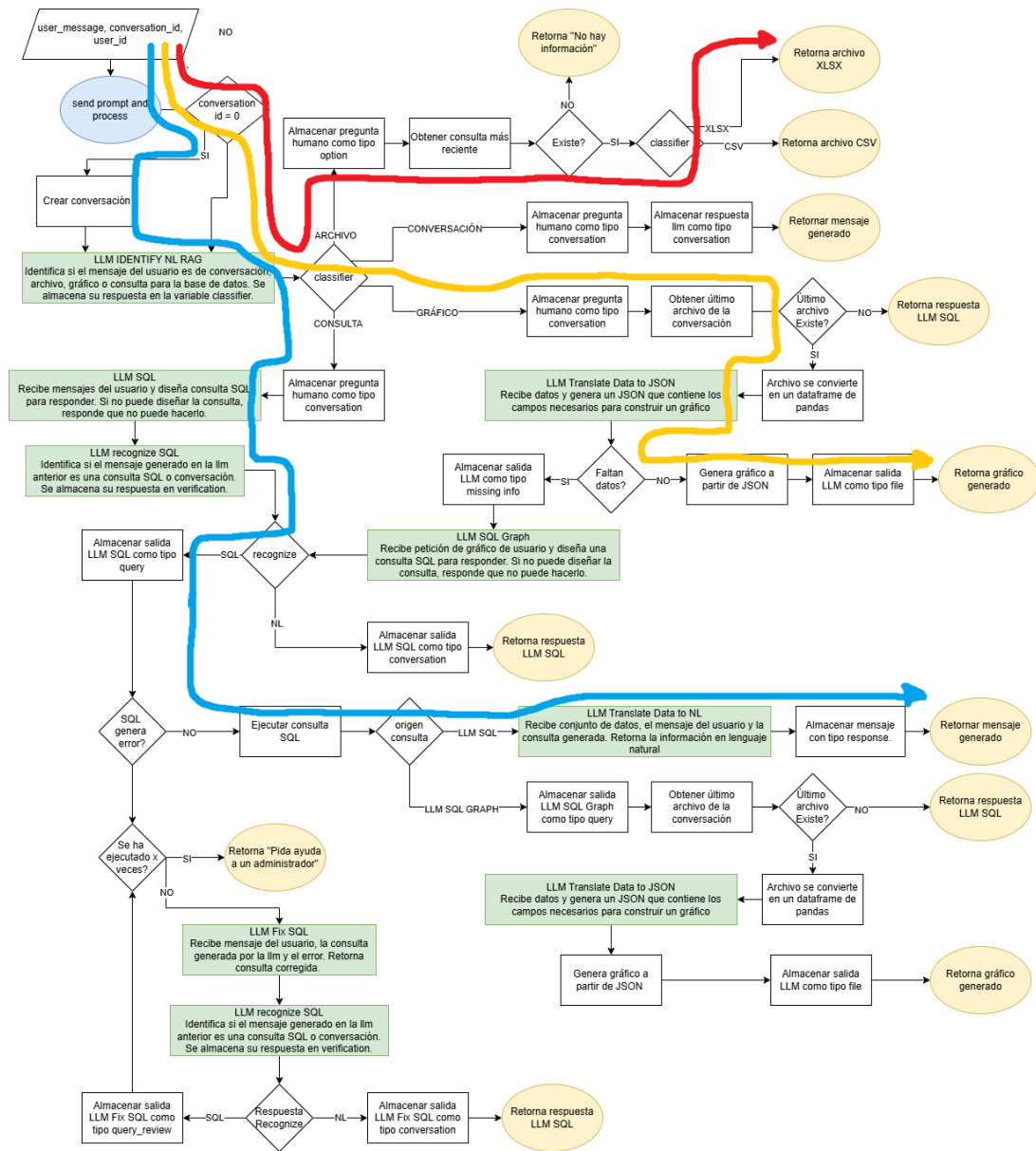


Figura 7.58: Camino para los distintos casos de uso reflejado en el diagrama de flujo

Capítulo 8

Conclusiones

El trabajo realizado demuestra cómo el uso de modelos de lenguaje de gran tamaño (LLMs) representa una alternativa atractiva para democratizar el acceso a la información en bases de datos. Aunque es una tecnología en constante evolución, los LLMs ya tienen la capacidad de abordar subtarefas complejas mediante ingeniería de prompt en cada solicitud, además de aprovechar modelos de embeddings para manejar prompts extensos y mejorar la comprensión de tareas complejas.

La solución desarrollada permite a usuarios, independientemente de su nivel técnico, realizar consultas, analizar información y exportar resultados de forma intuitiva. Esto se logra mediante un sistema de invocaciones en cadena a modelos LLM de Amazon Bedrock (principalmente Claude Sonnet 3.5), sistema el cual está diseñado para cubrir todas las posibles variantes de interacción que un usuario pueda tener con la plataforma durante una conversación.

Capítulo 9

Trabajo a futuro

Se proponen diversos desarrollos futuros para ampliar y fortalecer las funcionalidades de la plataforma.

- Realizar un análisis comparativo con otras estrategias de entrenamiento, como fine tuning o parameter-efficient fine tuning, con el fin de identificar la estrategia más adecuada en términos de efectividad y reducción de costos.
- Optimizar el prompt y el flujo de generación de gráficos para prevenir problemas de memoria al solicitar cambios de manera separada.
- Mejorar el generador de gráficos para obtener resultados más precisos, consistentes y personalizables.
- Desarrollar un subflujo que analice los gráficos y genere información adicional o nuevos gráficos que puedan ser de utilidad para el usuario.
- Implementar la visualización de gráficos directamente en el chatbot, facilitando la interacción y consulta de resultados.

Anexos

9.1. Gestión de riesgos

A continuación se detallan supuestos, dependencias, restricciones y riesgos asociados al proceso y desarrollo del trabajo realizado.

9.1.1. Supuestos

1. Se supone disponibilidad de los distintos modelos de lenguaje.
2. Se supone la existencia de diferentes métricas relacionadas al desempeño de los modelos.
3. Se supone que las métricas referentes al desempeño de los modelos son comparables.
4. Se supone la capacidad de creación de nuevos modelos en la base de datos del cliente para el almacenamiento de las diferentes métricas de las interacciones usuario - chatbot.

9.1.2. Dependencias

1. Se depende de la entrega del documento de confidencialidad para obtener acceso a estos datos.
2. Se depende de la entrega de datos por parte de la fundación.

3. Se depende del acceso a servicios como JupyterHub y Amazon Bedrock con sus respectivos modelos fundacionales.
4. Se depende de la comunicación con el cliente para cualquier eventualidad que se considere pertinente.

9.1.3. Restricciones

1. Las bases de datos del cliente utilizan PostgreSQL y MySQL como gestores y SQL como lenguaje, por lo que se trabajará dentro de un paradigma relacional.

9.1.4. Riesgos

1. Riesgos asociados al desconocimiento tecnológico propios de un proyecto que involucra nuevas tecnologías.
2. Riesgos asociados a no obtener información por parte del cliente.

Riesgo	Medidas de mitigación
Desconocimiento tecnológico	Estudio por parte del equipo.
No obtención de la información	Elaboración de modelos y bases de datos sintéticas que se asemejen a las utilizadas en la industria.

Cuadro 9.1: Riesgos y respectivas medidas de mitigación

9.2. Verificación y validación

9.2.1. Verificación

Actualmente, las pruebas se han limitado a pruebas simples de comportamiento de los distintos modelos en su contexto y pruebas del flujo principal. En particular, el equipo ha hecho pruebas unitarias, de integración y de extremo a extremo para verificar el correcto funcionamiento de los modelos y el sistema respectivamente.

9.2.2. Validación

Si bien el proceso de validación se ha realizado por medio de distintas reuniones, se planea que después de mostrar el prototipo inicial al cliente, estas puedan ser más frecuentes con el fin de recopilar más retroalimentación para añadir funcionalidades, mejoras y ajustes en la aplicación. Mediante este proceso, se busca generar conocimiento sobre las potenciales dificultades que el cliente o los usuarios finales tengan con el prototipo.

La retroalimentación recibida ha estado enfocada en añadir funcionalidades al flujo de modelos con el fin de aumentar la calidad de uso del sistema.

9.2.3. Plan de testing

Como plan de testing, se busca ir más allá de las pruebas realizadas por el equipo, para esto se pretenden hacer pruebas más contundentes con participación del cliente y personas ajenas al desarrollo, entre los que idealmente se encuentran los usuarios finales de la plataforma.

Bibliografía

- [1] Abhilasha Kate, Satish Kamble, Aishwarya Bodkhe, and Mrunal Joshi. Conversion of natural language query to sql query. In *Second International Conference on Electronics, Communication and Aerospace Technology (ICECA)*, March 2018.
- [2] Dawei Gao, Haibin Wang, Yaliang Li, Xiuyu Sun, Yichen Qian, Bolin Ding, and Jingren Zhou. Text-to-sql empowered by large language models: A benchmark evaluation. *arXiv:2308.153*, November 2023.
- [3] Zijin Hong, Zheng Yuan, Hao Chen, Qinggang Zhang, Feiran Huang, and Xiao Huang. Knowledge-to-sql: Enhancing sql generation with data expert llm. *arXiv:2402.11517v2*, March 2024.
- [4] Bing Wang, Changyu Ren, Jian Yang, Xinnian Liang, Jiaqi Bai, Linzheng Chai, Zhao Yan, Qian-Wen Zhang, Di Yin, Xing Sun, and Zhoujun Li. Mac-sql: A multi-agent collaborative framework for text-to-sql. *arXiv:2405.07467v1*, February 2024.
- [5] Puneet Kumar Ojha, Abhishek Gautam, Ankit Agrahari, and Parikshit Singh. Sft for improved text-to-sql translation. February 2024.
- [6] Dongjun Lee, Choongwon Park, Jaehyuk Kim, and Heesoo Park. Mcs-sql: Leveraging multiple prompts and multiple-choice selection for text-to-sql generation. *arXiv:2312.11242v3*, May 2024.

- [7] Antonia Creswell, Tom White, Vincent Dumoulin, Kai Arulkumaran, Biswa Sengupta, and Anil A. Bharath. Generative adversarial networks: An overview. *IEEE Signal Processing Magazine*, 35(1):53–65, January 2018.
- [8] Yingpeng Du, Di Luo, Rui Yan, Xiaopei Wang, Hongzhi Liu, Hengshu Zhu, Yang Song, and Zhang. Enhancing job recommendation through llm-based generative adversarial networks. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38(8):8363–8371, March 2024.
- [9] Zhenwen Li and Tao Xie. Using llm to select the right sql query from candidates. *arXiv:2401.02115v1*, January 2024.