Máster en Big Data y Analítica Avanzada

Trabajo de Fin de Máster

Autoservicio en lenguaje natural basado en una arquitectura agéntica para la generación de inteligencia de negocio destinada a entidades financieras

Autor
Laura González Morán

Dirigido por
José Daniel Menéndez Hernández - Neptune North & Oliver Wyman

Madrid
Mayo 2026

El autor D/Dña **Laura González Morán**

DECLARA , bajo Su responsabilidad, que el Proyecto presentado con el título

Autoservicio en lenguaje natural basado en una arquitectura agéntica para la generación de inteligencia de negocio destinada a entidades financieras

en la ETS de Ingeniería - ICAI de la Universidad Pontificia Comillas en el curso académico **2025-2026** es de mi autoría, original e inédito y no ha sido presentado con anterioridad a otros efectos. El Proyecto no es plagio de otro, ni total ni parcialmente y la información que ha sido tomada de otros documentos está debidamente referenciada.

GONZALEZ MORAN, LAURA (FIRMA)

Firmado digitalmente por GONZALEZ MORAN, LAURA (FIRMA)
Nombre de reconocimiento (DN): c=ES, serialNumber=53997363X, sn=GONZALEZ, givenName=LAURA, cn=GONZALEZ MORAN, LAURA (FIRMA)
Fecha: 2026.05.31 22:21:58 +02'00'

El Alumno, Fdo: Fecha:

Autorizada la entrega del proyecto

EL DIRECTOR DEL PROYECTO

MENENDEZ HERNANDEZ JOSE DANIEL - 54494034L

Digitally signed by MENENDEZ HERNANDEZ JOSE DANIEL - 54494034L
Date: 2026.06.01 15:04:56 +02'00'

El Director, Fdo .: Fecha:

AUTORIZACIÓN PARA LA DIGITALIZACIÓN, DEPÓSITO Y DIVULGACIÓN EN RED DE PROYECTOS FIN DE GRADO, FIN DE MÁSTER, TESINAS O MEMORIAS DE BACHILLERATO

1º. Declaración de la autoría y acreditación de la misma.
El autor D Laura González Morán

_____
DECLARA ser el titular de los derechos de propiedad intelectual de la obra:

Autoservicio en lenguaje natural basado en una arquitectura agéntica para la generación de inteligencia de negocio destinada a entidades financieras

que ésta es una obra original, y que ostenta la condición de autor en el sentido que otorga la Ley de Propiedad Intelectual.

2º. Objeto y fines de la cesión.
Con el fin de dar la máxima difusión a la obra citada a través del Repositorio institucional de la Universidad, el autor **CEDE** a la Universidad Pontificia Comillas, de forma gratuita y no exclusiva, por el máximo plazo legal y con ámbito universal, los derechos de digitalización, de archivo, de reproducción, de distribución y de comunicación pública, incluido el derecho de puesta a disposición electrónica, tal y como se describen en la Ley de Propiedad Intelectual. El derecho de transformación se cede a los únicos efectos de lo dispuesto en la letra a) del apartado siguiente.

3º. Condiciones de la cesión y acceso
Sin perjuicio de la titularidad de la obra, que sigue correspondiendo a su autor, la cesión de derechos contemplada en esta licencia habilita para:

a) Transformarla con el fin de adaptarla a cualquier tecnología que permita incorporarla a internet y hacerla accesible; incorporar metadatos para realizar el registro de la obra e incorporar "marcas de agua" o cualquier otro sistema de seguridad o de protección.
b) Reproducirla en un soporte digital para su incorporación a una base de datos electrónica, incluyendo el derecho de reproducir y almacenar la obra en servidores, a los efectos de garantizar su seguridad, conservación y preservar el formato.
c) Comunicarla, por defecto, a través de un archivo institucional abierto, accesible de modo libre y gratuito a través de internet.
d) Cualquier otra forma de acceso (restringido, embargado, cerrado) deberá solicitarse expresamente y obedecer a causas justificadas.
e) Asignar por defecto a estos trabajos una licencia Creative Commons.
f) Asignar por defecto a estos trabajos un HANDLE (URL *persistente)*.

4º. Derechos del autor.
El autor, en tanto que titular de una obra tiene derecho a:

a) Que la Universidad identifique claramente su nombre como autor de la misma
b) Comunicar y dar publicidad a la obra en la versión que ceda y en otras posteriores a través de cualquier medio.
c) Solicitar la retirada de la obra del repositorio por causa justificada.
d) Recibir notificación fehaciente de cualquier reclamación que puedan formular terceras personas en relación con la obra y, en particular, de reclamaciones relativas a los derechos de propiedad intelectual sobre ella.

5º. Deberes del autor.
El autor se compromete a:

a) Garantizar que el compromiso que adquiere mediante el presente escrito no infringe ningún derecho de terceros, ya sean de propiedad industrial, intelectual o cualquier otro.

b) Garantizar que el contenido de las obras no atenta contra los derechos al honor, a la intimidad y a la imagen de terceros.
c) Asumir toda reclamación o responsabilidad, incluyendo las indemnizaciones por daños, que pudieran ejercitarse contra la Universidad por terceros que vieran infringidos sus derechos e intereses a causa de la cesión.
d) Asumir la responsabilidad en el caso de que las instituciones fueran condenadas por infracción de derechos derivada de las obras objeto de la cesión.

6º. Fines y funcionamiento del Repositorio Institucional.

La obra se pondrá a disposición de los usuarios para que hagan de ella un uso justo y respetuoso con los derechos del autor, según lo permitido por la legislación aplicable, y con fines de estudio, investigación, o cualquier otro fin lícito. Con dicha finalidad, la Universidad asume los siguientes deberes y se reserva las siguientes facultades:

- La Universidad informará a los usuarios del archivo sobre los usos permitidos, y no garantiza ni asume responsabilidad alguna por otras formas en que los usuarios hagan un uso posterior de las obras no conforme con la legislación vigente. El uso posterior, más allá de la copia privada, requerirá que se cite la fuente y se reconozca la autoría, que no se obtenga beneficio comercial, y que no se realicen obras derivadas.
- La Universidad no revisará el contenido de las obras, que en todo caso permanecerá bajo la responsabilidad exclusive del autor y no estará obligada a ejercitar acciones legales en nombre del autor en el supuesto de infracciones a derechos de propiedad intelectual derivados del depósito y archivo de las obras. El autor renuncia a cualquier reclamación frente a la Universidad por las formas no ajustadas a la legislación vigente en que los usuarios hagan uso de las obras.
- La Universidad adoptará las medidas necesarias para la preservación de la obra en un futuro.
- La Universidad se reserva la facultad de retirar la obra, previa notificación al autor, en supuestos suficientemente justificados, o en caso de reclamaciones de terceros.

Madrid, a 31 de mayo de 2026

ACEPTA

Fdo Laura González Morán

Motivos para solicitar el acceso restringido, cerrado o embargado del trabajo en el Repositorio Institucional:

Máster en Big Data y Analítica Avanzada

Trabajo de Fin de Máster

Autoservicio en lenguaje natural basado en una arquitectura agéntica para la generación de inteligencia de negocio destinada a entidades financieras

Autor
Laura González Morán

Dirigido por
José Daniel Menéndez Hernández - Neptune North & Oliver Wyman

Madrid
Mayo 2026

Agradecimientos

Este proyecto es el resultado de meses de trabajo intenso, pero también de la generosidad de varias personas que merece la pena reconocer.

A Oliver Wyman, por haber sido un entorno donde aprender de verdad. La consultoría estratégica tiene una forma particular de exigir claridad y rigor, y haber podido experimentarlo desde dentro ha marcado la manera en que entiendo el trabajo bien hecho.

A mi tutor José Daniel Menéndez, por la calidad de su acompañamiento a lo largo de todo el proyecto. Su conocimiento en arquitectura agéntica, sistemas cloud y bases de datos ha sido una referencia técnica constante a lo largo del proyecto, pero lo que más merece reconocimiento es su disposición genuina para ejercer de mentor y su calidad humana.

A Joan Masip, por haberme enseñado a pensar un proyecto más allá de su implementación técnica: cómo tomar decisiones estratégicas, transmitir el impacto de mi trabajo y fomentar la adopción de una solución. Y por haber tratado mi desarrollo profesional con la misma seriedad que el proyecto en sí.

Por último, a mi familia, por el apoyo con el que me han acompañado a lo largo de toda mi formación, por los sacrificios que han hecho posible que llegara hasta aquí, y por haberme enseñado desde pequeña que el esfuerzo y la educación son la mejor inversión que existe.

AUTOSERVICIO EN LENGUAJE NATURAL BASADO EN UNA ARQUITECTURA AGÉNTICA PARA LA GENERACIÓN DE INTELIGENCIA DE NEGOCIO DESTINADA A ENTIDADES FINANCIERAS

Autora: González Morán, Laura.
Director: Menéndez Hernández, José Daniel.
Entidad Colaboradora: Oliver Wyman.

RESUMEN DEL PROYECTO

Se diseña, implementa y evalúa un chatbot de autoservicio analítico que permite a socios y consultores de una firma de consultoría estratégica interrogar un dataset financiero multibanco en lenguaje natural, sin conocimiento de SQL y sin intermediación técnica. El sistema se apoya en una arquitectura agéntica de siete nodos implementada con LangGraph y GPT-4o como modelo de razonamiento, con DuckDB como motor de ejecución analítica y Streamlit como interfaz conversacional. La evaluación sobre 38 preguntas de referencia extraídas de análisis financieros reales alcanza una accuracy del 92 %, con una latencia media de respuesta de aproximadamente 23 segundos y un 86 % de valoraciones positivas por parte de los usuarios finales.

Palabras clave: Text-to-SQL, Arquitectura Agéntica, Inteligencia de Negocio, Modelos de Lenguaje de Gran Escala, Procesamiento del Lenguaje Natural.

1. Introducción

En entornos de consultoría financiera, el acceso a datos agregados de clientes está mediado por perfiles técnicos que traducen las preguntas de negocio a consultas ejecutables. Este ciclo de petición y entrega introduce fricción operativa, limita la iteración durante la preparación de propuestas y concentra riesgo de inconsistencia en los cálculos. Los modelos de lenguaje de gran escala (Large Language Models, LLM) permiten abordar este problema mediante la traducción directa de lenguaje natural a SQL, pero su aplicación fiable sobre dominios específicos requiere una capa semántica que ancle la generación al vocabulario y las métricas del dominio.

2. Definición del proyecto

El proyecto desarrolla un sistema conversacional que recibe una pregunta en lenguaje natural, genera la consulta SQL correspondiente sobre el dataset financiero,

ejecuta la consulta, valida el resultado, produce una interpretación en lenguaje natural y decide si el resultado se acompaña de una visualización. Todo el flujo opera de forma autónoma, sin intervención del usuario entre pasos. El sistema se despliega en producción sobre infraestructura Azure y es utilizado por socios y consultores de la firma en condiciones reales de uso.

3. Descripción del sistema

La arquitectura se organiza como un grafo dirigido de siete nodos especializados con estado tipado compartido, implementado con LangGraph [3]. El nodo generador produce la consulta SQL [1] a partir del catálogo semántico del dominio; el nodo de herramienta la ejecuta sobre DuckDB; el nodo validador detecta resultados incoherentes y reintenta la consulta si es necesario; el nodo de análisis produce la interpretación en lenguaje natural; el nodo de trazabilidad genera una explicación auditable del procedimiento seguido; y el nodo de visualización decide si el resultado justifica un gráfico y, en caso afirmativo, genera la especificación Altair correspondiente.

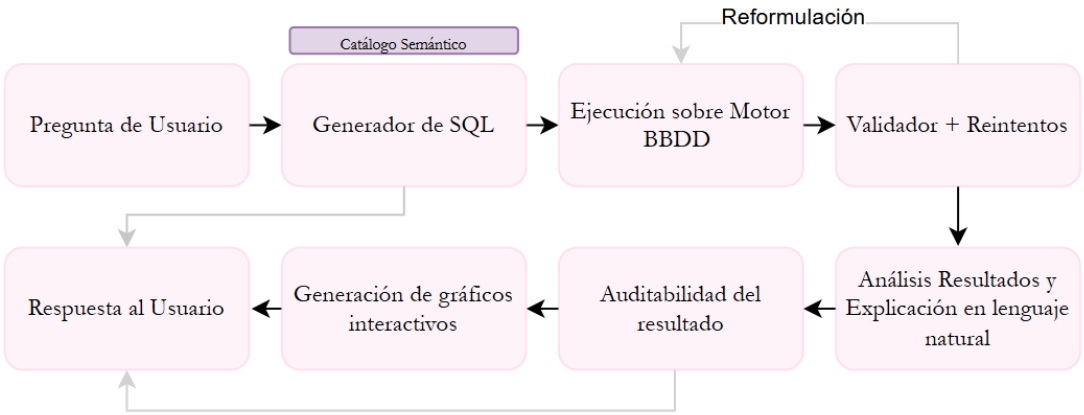
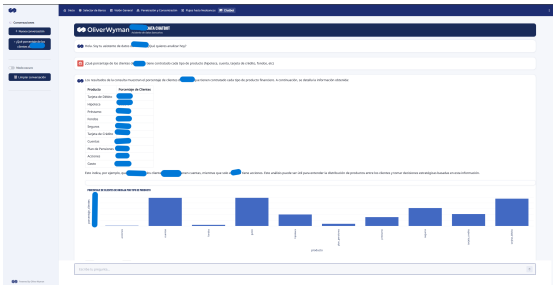
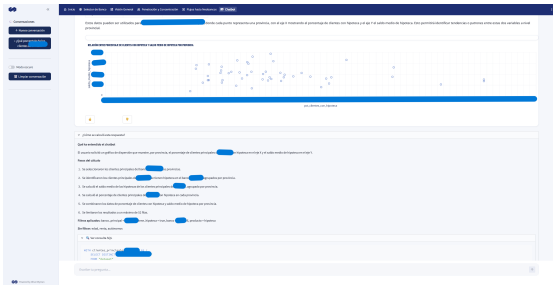


Figura 1: Arquitectura del sistema: flujo de procesamiento de una consulta a través de los siete nodos del grafo agéntico.



(a) Vista de la interfaz gráfica - anonimizada



(b) Vista de la interfaz gráfica - anonimizada

Figura 2: Interfaz gráfica del sistema.

4. Resultados

El sistema se evalúa mediante un pipeline automatizado con metodología LLM-as-judge [2] sobre 38 preguntas de referencia clasificadas en cuatro niveles de complejidad (C1–C4). La accuracy global sobre la versión final es del 92 %, con un 86 % de valoraciones positivas recogidas durante el despliegue en producción. La figura 6 muestra la evolución de la accuracy a lo largo de las cuatro fases principales de desarrollo.

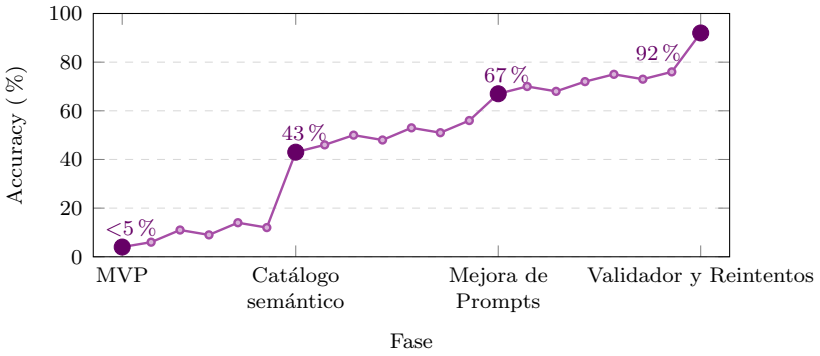


Figura 3: Evolución de la accuracy del sistema. Los puntos grandes corresponden a las cuatro fases con mayor impacto; los pequeños, a iteraciones intermedias.

5. Conclusiones

El sistema demuestra que la combinación de un LLM de propósito general con una capa semántica de dominio y un mecanismo de validación autónoma permite construir un chatbot analítico fiable sobre datos financieros de dominio específico. La

accuracy del 92 % sobre preguntas reales, la trazabilidad integrada en cada respuesta y el feedback positivo de los usuarios finales respaldan la viabilidad del autoservicio analítico para perfiles no técnicos en entornos de consultoría. El desarrollo iterativo guiado por evaluación continua permitió mantener el sistema predecible y evitar la incorporación de complejidad técnica sin evidencia de necesidad.

6. Referencias

[1] Yu, T. et al. "Spider: A Large-Scale Human-Labeled Dataset for Complex and Cross-Domain Semantic Parsing and Text-to-SQL Task." *EMNLP*, 2018.

[2] Zheng, L. et al. "Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena." *NeurIPS*, 2023.

[3] LangChain Inc. *LangGraph: Build Resilient Language Agents as Graphs*. GitHub, 2024. `https://github.com/langchain-ai/langgraph`

NATURAL LANGUAGE SELF-SERVICE BASED ON AN AGENTIC ARCHITECTURE FOR BUSINESS INTELLIGENCE GENERATION IN FINANCIAL INSTITUTIONS

Author: González Morán, Laura.
Supervisor: Menéndez Hernández, José Daniel.
Collaborating Entity: Oliver Wyman.

ABSTRACT

This work designs, implements and evaluates an analytical self-service chatbot that enables strategy consultants to query a multi-bank financial dataset in natural language, without SQL knowledge and without technical mediation. The system is built on a seven-node agentic architecture implemented with LangGraph and GPT-4o as the reasoning model, DuckDB as the analytical execution engine and Streamlit as the conversational interface. Evaluation over 38 reference questions drawn from real financial analyses achieves 92 % accuracy, with a mean response latency of approximately 23 seconds and 86 % positive ratings from end users.

Keywords: Text-to-SQL, Agentic Architecture, Business Intelligence, Large Language Models, Natural Language Processing.

1. Introduction

In financial consulting environments, access to aggregated client data is mediated by technical profiles who translate business questions into executable queries. This request-and-delivery cycle introduces operational friction, limits iteration during proposal preparation and concentrates the risk of metric inconsistency. Large Language Models enable addressing this problem through direct natural-language-to-SQL translation, but their reliable application on specific domains requires a semantic layer that grounds generation in the domain's vocabulary and metrics.

2. Project definition

The project develops a conversational system that receives a natural language question, generates the corresponding SQL query over the financial dataset, executes it, validates the result, produces a natural language interpretation, and decides whether the result warrants a visualisation. The entire flow operates autonomously. The system is deployed in production on Azure infrastructure and used by partners and consultants under real working conditions.

3. System description

The architecture is organised as a directed graph of seven specialised nodes with shared typed state, implemented with LangGraph [3].

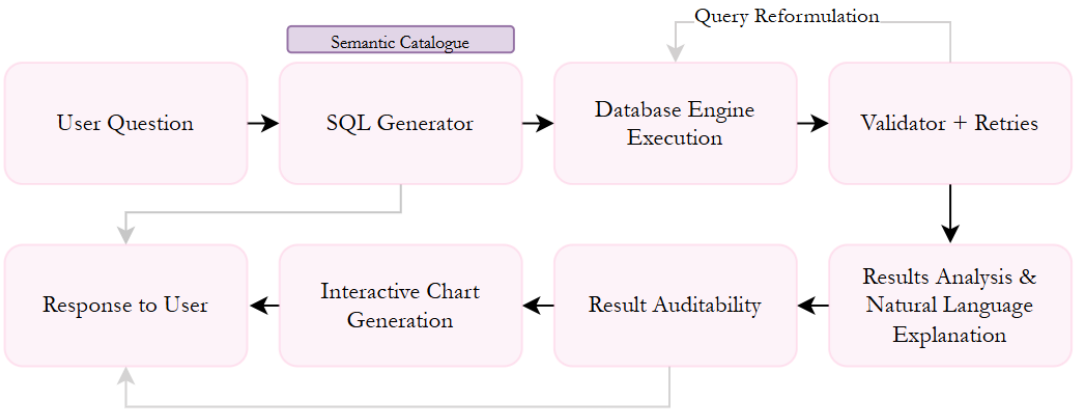
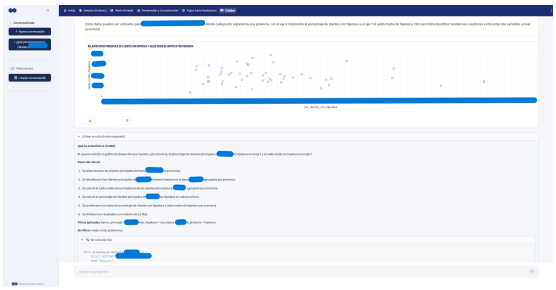


Figura 4: System Architecture

The generator node produces the SQL query [1] from the domain semantic catalogue; the tool node executes it on DuckDB; the validator node detects incoherent results and retries if necessary; the analysis node produces the natural language interpretation; the traceability node generates an auditable explanation of the procedure followed; and the visualisation node decides whether the result justifies a chart and, if so, generates the corresponding Altair specification.

(a) Example view of the UI - anonymized



(b) Example view of the UI - anonymized

Figura 5: User Interface of the built system.

4. Results

The system is evaluated through an automated pipeline using an LLM-as-judge [2] methodology over 38 reference questions classified into four complexity levels (C1–C4). Overall accuracy on the final version is 92 %, with 86 % positive ratings collected during production deployment. The accuracy evolution across the four main development phases is shown in Figure **??**.

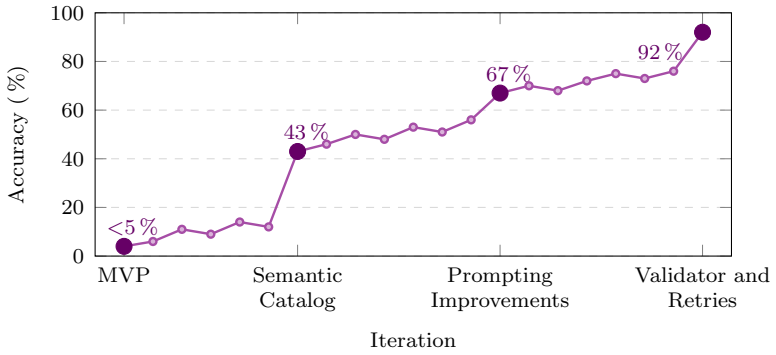


Figura 6: Accuracy evolution during development. Large dots mark the four phases with the greatest impact; small dots represent intermediate iterations.

5. Conclusions

The system demonstrates that combining a general-purpose LLM with a domain semantic layer and an autonomous validation mechanism enables building a reliable analytical chatbot over specific-domain financial data. The 92 % accuracy on real questions, the traceability integrated into each response and the positive end-user feedback support the viability of analytical self-service for non-technical profiles in consulting environments. The iterative development process guided by continuous evaluation kept the system predictable and avoided the incorporation of technical complexity without evidence of need.

6. References

[1] Yu, T. et al. “Spider: A Large-Scale Human-Labeled Dataset for Complex and Cross-Domain Semantic Parsing and Text-to-SQL Task.” *EMNLP*, 2018.

[2] Zheng, L. et al. “Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena.” *NeurIPS*, 2023.

[3] LangChain Inc. *LangGraph: Build Resilient Language Agents as Graphs*. GitHub, 2024. https://github.com/langchain-ai/langgraph

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

1.1. Contexto y motivación

La motivación de este proyecto nace de una realidad operativa concreta: la consultora estratégica Oliver Wyman dispone de un activo de datos valioso, pero su explotación efectiva no está alineada con el ritmo y las exigencias de la consultoría. En el caso del dataset financiero multibanco, el valor potencial es alto porque permite extraer métricas relevantes para entidades financieras. Sin embargo, ese valor solo se materializa cuando las preguntas de negocio se convierten en resultados fiables, consistentes y reutilizables en plazos muy cortos.

Actualmente existe una brecha entre la demanda de insights por parte de socios y la capacidad real de obtenerlos de forma autónoma. Cuando un socio necesita cifras o visualizaciones para una propuesta, la traducción de la pregunta de negocio a cálculos sobre el dataset requiere conocimiento técnico específico. En la práctica, esto obliga a canalizar las peticiones a través de perfiles especialistas, que deben procesar los ficheros, calcular las métricas solicitadas y generar las visualizaciones necesarias. Este enfoque introduce un cuello de botella operativo y limita la capacidad de iteración, ya que la exploración del dataset queda restringida a la disponibilidad de un recurso técnico que, además, suele estar asignado a múltiples iniciativas de forma simultánea.

Esta fricción es especialmente crítica en consultoría por tres motivos. En primer lugar, porque los ciclos de propuesta y los plazos de entrega son cortos, y la velocidad de respuesta condiciona la capacidad de construir una narrativa convincente y diferenciadora para el cliente. En segundo lugar, porque la calidad y consistencia de las métricas es un elemento central de credibilidad: resultados que no son reproducibles o que dependen de interpretaciones ad hoc generan incertidumbre y erosionan la confianza en el entregable. En tercer lugar, porque la consultoría requie-

re iteración rápida: la primera pregunta rara vez es la última, y la imposibilidad de explorar autónomamente el dato reduce la capacidad de refinar hipótesis y profundizar en segmentos, comparativas o desgloses que surgen durante la preparación de una propuesta.

El enfoque actual incorpora además riesgos relevantes. Al tratarse de un proceso manual y ad hoc, aumenta la probabilidad de errores en cálculos o en la interpretación de definiciones, y se dificulta mantener un criterio estable sobre cómo se computan métricas clave a lo largo del tiempo. La trazabilidad también se resiente: no siempre es inmediato reconstruir qué filtros, parámetros o transformaciones han llevado a un resultado concreto. En un contexto en el que las cifras y gráficos pueden formar parte de presentaciones a bancos y aseguradoras, estos riesgos afectan directamente a la robustez del mensaje que la firma traslada a sus clientes.

Existe, por último, un coste de oportunidad claro. El tiempo de perfiles senior altamente especializados se dedica a tareas repetitivas de extracción, cálculo y visualización, en lugar de centrarse en actividades de mayor valor añadido. Al mismo tiempo, la falta de autoservicio limita el aprovechamiento del activo de datos por parte de los socios, reduciendo la frecuencia y profundidad con la que el dataset se incorpora en propuestas y, por tanto, la capacidad de capturar su valor comercial. El problema no es únicamente de eficiencia, sino de escalabilidad y de conversión de un activo disponible en una capacidad interna repetible, fiable y alineada con la dinámica real de la firma.

1.2. Descripción de las tecnologías

El sistema desarrollado integra un conjunto de tecnologías que cubren las distintas capas del sistema: la interfaz conversacional, el razonamiento lingüístico, la orquestación agéntica, la ejecución de consultas, el despliegue en producción y el control de versiones. A continuación se describe cada componente y el rol que desempeña en la arquitectura.

- **Python.** Lenguaje de programación sobre el que se implementa la totalidad de la lógica del sistema. Su ecosistema de bibliotecas para inteligencia artificial, procesamiento de datos e interfaces web lo convierte en el entorno natural para integrar los componentes descritos a continuación.

- **LangGraph.** Marco de trabajo (framework) de orquestación agéntica que permite definir flujos de procesamiento como grafos dirigidos con estado compartido. A diferencia de cadenas lineales de procesamiento, LangGraph soporta

ciclos de retroalimentación, puntos de decisión condicionales y persistencia del estado entre nodos. El sistema implementa un grafo de siete nodos especializados, descrito en detalle en el Capítulo 3.

- **DuckDB.** Motor de base de datos analítico (OLAP) de proceso en memoria diseñado para consultas sobre datasets columnares. Se eligió por su capacidad de leer directamente archivos en formato Parquet sin necesidad de servidor y por su rendimiento en operaciones de agregación. Su uso en formato nativo redujo la latencia de ejecución de consultas de 80,6 segundos a aproximadamente 23 segundos respecto a la configuración inicial.
- **Pydantic.** Biblioteca de validación de datos para Python que permite definir contratos de entrada y salida mediante anotaciones de tipo. En el sistema, cada nodo del grafo tiene asociado un esquema Pydantic que valida la estructura de su output antes de pasarlo al siguiente nodo, garantizando la coherencia del estado a lo largo de la cadena de procesamiento.
- **Streamlit.** Marco de trabajo para la construcción de aplicaciones web interactivas en Python. Se utiliza como capa de interfaz de usuario (UI) del chatbot, proporcionando la ventana conversacional, el panel de trazabilidad y los controles de exportación de resultados.
- **Altair / Vega-Lite.** Biblioteca de visualización declarativa basada en la gramática Vega-Lite. El nodo de visualización del grafo genera especificaciones Altair que Streamlit renderiza como gráficos interactivos. El enfoque declarativo permite que el LLM produzca especificaciones de visualización como objetos estructurados, con validación posterior antes de su renderizado.
- **Microsoft Azure.** Plataforma en la nube sobre la que se despliega y opera el sistema. Se utilizaron los siguientes servicios: *Azure App Service*, para el despliegue de la aplicación Streamlit en un entorno gestionado accesible por los usuarios finales; *Azure OpenAI*, para el acceso al modelo GPT-4o con garantías de residencia de datos en infraestructura europea; *Azure Application Insights*, para el registro asíncrono de métricas de uso, latencias por nodo y excepciones en producción mediante el patrón *fire-and-forget* (disparar y olvidar); y *Azure Key Vault*, para la gestión centralizada y segura de credenciales y claves de API, evitando su exposición directa en el código.
- **Git / GitHub.** Sistema de control de versiones distribuido utilizado para el seguimiento del desarrollo, la gestión de ramas por funcionalidad y el registro de la evolución del código a lo largo del proyecto.

La tabla 1.1 resume el stack tecnológico con la función de cada componente y la capa del sistema a la que pertenece.

Cuadro 1.1: Stack tecnológico del sistema

Componente	**Capa**	**Función principal**
Python	Base	Lenguaje de implementación del sistema
GPT-4o	Razonamiento	Generación de SQL, validación, trazabilidad
LangGraph	Orquestación	Grafo agéntico con estado tipado
DuckDB	Datos	Motor OLAP en memoria, lectura Parquet
Pydantic	Contratos	Validación de outputs por nodo
Streamlit	Interfaz	UI conversacional y exportación
Altair / Vega-Lite	Visualización	Gráficos interactivos declarativos
Microsoft Azure	Despliegue	App Service, OpenAI, App Insights, Key Vault
Git / GitHub	Desarrollo	Control de versiones y gestión de ramas

El Capítulo 3 describe cómo estas tecnologías se integran en la arquitectura del sistema y las decisiones de diseño que motivaron su elección frente a alternativas.

1.3. Objetivos del proyecto

El objetivo central del proyecto es habilitar el autoservicio analítico sobre datos financieros para perfiles no técnicos mediante una interfaz conversacional en lenguaje natural. Esto implica no solo que el sistema genere consultas correctas sobre el dataset financiero, sino que las respuestas sean verificables, los cálculos estén anclados a las definiciones de dominio de la firma y los resultados se presenten en formatos directamente utilizables en el trabajo diario de socios y consultores.

A partir de ese objetivo central se derivan cinco objetivos específicos:

- **O1 — Habilitar el autoservicio analítico para perfiles no técnicos.** Permitir que socios y consultores obtengan indicadores financieros calculados y verificados mediante lenguaje natural, sin intervención técnica ni conocimiento de SQL. El acceso al dato debe funcionar en condiciones de uso real, no solo en entornos de prueba controlados.
- **O2 — Garantizar que el sistema opera sobre las definiciones de métricas validadas por la firma.** Construir una capa semántica que codifique las definiciones de dominio, las fórmulas de KPIs y los valores posibles de las variables financieras, de forma que los cálculos producidos sean coherentes con

las reglas de negocio establecidas y no dependan de la interpretación abierta del modelo de lenguaje.

- **O3 — Dotar al sistema de mecanismos de validación y autocorrección autónomos.** Implementar un nodo que detecte resultados incoherentes con la intención analítica de la pregunta y reintente la consulta de forma automática, sin requerir intervención del usuario ni exposición explícita del error.
- **O4 — Hacer cada respuesta auditable y reproducible.** Acompañar cada resultado con una explicación en lenguaje natural del procedimiento seguido: la métrica aplicada, los filtros utilizados y la consulta ejecutada. Esta trazabilidad debe estar integrada en el flujo conversacional, no como un apartado técnico separado.
- **O5 — Presentar los resultados en formatos directamente utilizables.** Generar visualizaciones interactivas cuando la naturaleza del resultado lo justifique y exponer los datos tabulares en formatos exportables compatibles con hojas de cálculo, de modo que el output del chatbot pueda integrarse directamente en presentaciones y análisis de cliente.

La consecución de estos objetivos se evalúa en la Sección 6.1 del Capítulo 6, a la luz de los resultados obtenidos durante la evaluación del sistema.

1.4. Metodología de trabajo

El proyecto adopta una metodología ágil (Agile) con desarrollo iterativo guiado por evaluación continua. Frente a un enfoque en cascada (waterfall), en el que los requisitos se fijan al inicio y el sistema se construye de forma secuencial hasta su entrega final, la metodología ágil resulta más adecuada para este proyecto por tres razones:

- **Incertidumbre técnica inherente.** En sistemas que dependen de modelos de lenguaje, el comportamiento real del sistema no puede anticiparse por completo en la fase de diseño. Los fallos concretos, las categorías de error dominantes y los límites del modelo solo se revelan cuando el sistema opera sobre preguntas reales del dominio. Un enfoque en cascada obligaría a tomar decisiones de arquitectura sin esa información.

- **Requisitos emergentes.** Parte de los requisitos del sistema surgieron durante el uso real: la necesidad de trazabilidad, por ejemplo, no fue identificada en la fase inicial de análisis, sino a partir del feedback de socios y consultores durante el despliegue. La metodología ágil permite incorporar ese conocimiento en el ciclo de desarrollo sin ruptura del proceso.

- **Riesgo de sobreingeniería.** Los sistemas conversacionales con LLMs ofrecen un espacio de mejoras técnicas muy amplio: recuperación aumentada por recuperación (RAG), arquitecturas multiagente, modelos especializados por nodo, entre otros. Comprometerse con esas soluciones desde el inicio, antes de saber si son necesarias, introduce complejidad sin garantía de mejora sobre el problema real. El enfoque iterativo permite priorizar únicamente las evoluciones cuya necesidad está respaldada por evidencia.

El desarrollo se organizó en sprints cortos, cada uno orientado a resolver un gap concreto identificado mediante el pipeline de evaluación automatizada o mediante el feedback directo de los usuarios finales. Ningún sprint respondió a una apuesta especulativa sobre lo que podría mejorar el sistema: la pregunta de partida de cada ciclo fue siempre qué problema medido o qué necesidad expresada justifica la siguiente inversión de desarrollo. Este criterio de priorización sirvió también como mecanismo de contención frente a la sobreingeniería: decisiones como posponer la incorporación de RAG o de arquitecturas multiagente fueron elecciones deliberadas tomadas mientras el pipeline de evaluación no mostrara un gap que esas soluciones fueran las más adecuadas para resolver.

La Sección 5.1 describe en detalle el diseño del pipeline de evaluación continua, y la Sección 5.7 recoge cómo cada sprint se justificó a partir de los resultados obtenidos en las ejecuciones anteriores.

Capítulo 2

Estado del Arte

El acceso analítico a datos empresariales no depende únicamente de que la información esté disponible, sino de que pueda transformarse en conocimiento útil, fiable y verificable. En el contexto de este Trabajo de Fin de Máster (TFM), el problema se sitúa sobre datos financieros de usuarios españoles, con visibilidad multi-entidad e información transaccional categorizada. Este tipo de datos ofrece un alto potencial para el análisis de negocio, pero también exige interpretar correctamente la intención analítica, seleccionar la granularidad adecuada, aplicar filtros coherentes y respetar definiciones métricas estables. Por ello, el reto no consiste solo en facilitar el acceso al dato, sino en reducir la distancia entre una pregunta de negocio y una respuesta cuantitativa que pueda ser inspeccionada y utilizada con confianza.

Este capítulo analiza las principales aproximaciones existentes para abordar esa distancia. En particular, se busca identificar qué problemas resuelve cada familia de soluciones, qué condiciones deja sin cubrir y qué brecha práctica justifica el diseño de una arquitectura específica para el sistema implementado. Esta brecha no nace de la ausencia de herramientas analíticas, sino de la dificultad de combinar acceso flexible, precisión numérica, trazabilidad, conocimiento del dominio y control del dato en un entorno profesional regulado.

Para ordenar el análisis, se emplean seis criterios comparativos:

- **Accesibilidad**: evalúa hasta qué punto una solución permite que usuarios de negocio interactúen con los datos sin depender de forma continua de perfiles técnicos.
- **Flexibilidad analítica**: mide la capacidad de responder a preguntas no previstas, así como de acompañar procesos de exploración y reformulación de hipótesis.

- **Precisión cuantitativa**: exige que los resultados procedan de operaciones correctas sobre datos estructurados, con filtros, agregaciones y niveles de granularidad coherentes.
- **Auditabilidad**: valora si puede reconstruirse la lógica seguida para obtener una respuesta, desde la interpretación inicial de la pregunta hasta el resultado final.
- **Adaptación al dominio**: considera la incorporación de definiciones, convenciones y restricciones propias del análisis financiero.
- **Control de datos**: recoge los requisitos de privacidad, gobierno y cumplimiento asociados al tratamiento de información sensible.

Estos criterios articulan la revisión de las soluciones existentes y permiten derivar, al cierre del capítulo, los requisitos que debe satisfacer el sistema desarrollado.

2.1. Business intelligence (BI) y analítica de autoservicio

El Business Intelligence (BI) designa el conjunto de herramientas, metodologías y procesos orientados a transformar datos corporativos en información estructurada que soporte la toma de decisiones. En su forma más extendida, el modelo BI descansa sobre una arquitectura de tres capas: una fuente de datos integrada, una capa semántica que define métricas, jerarquías y relaciones de negocio, y una capa de presentación compuesta por paneles visuales e informes prediseñados. Herramientas como Tableau, Microsoft Power BI o Looker han consolidado este paradigma durante las últimas dos décadas, convirtiéndose en el estándar de referencia para el análisis de datos en entornos empresariales [1][2].

La fortaleza del BI reside en su capacidad para responder con consistencia y velocidad a preguntas conocidas. Cuando una organización necesita monitorizar indicadores recurrentes, como el margen por línea de negocio, la evolución mensual de ingresos o la tasa de retención de clientes, el modelo de dashboards ofrece una solución madura: las métricas están definidas de forma centralizada, todos los equipos acceden a la misma versión de la verdad y el tiempo entre pregunta y respuesta se reduce a segundos. Estudios sobre adopción de BI en entornos empresariales documentan precisamente estos beneficios: estandarización de KPIs, reducción de dependencia de

equipos técnicos para consultas habituales y mejora en la consistencia de los informes de gestión [3][4]. La analítica de autoservicio (self-service analytics) extiende este modelo otorgando al usuario final cierta capacidad de exploración, aunque siempre dentro de los límites del modelo semántico preexistente.

Sin embargo, este modelo arquitectónico descansa sobre un supuesto que raramente se explicita: que las preguntas analíticamente relevantes son conocidas en el momento de diseñar la capa semántica. Es ese supuesto, y no una deficiencia de implementación, lo que define el perímetro de aplicabilidad del BI. Cuando un analista necesita explorar un dataset cuya semántica de negocio no ha sido modelada previamente, reformular hipótesis durante el análisis o formular preguntas ad hoc que ningún dashboard anticipa, el flujo de trabajo se transforma: la pregunta debe trasladarse al equipo técnico, que construye o adapta el modelo, lo que introduce un ciclo de latencia incompatible con el ritmo exploratorio del análisis. Passlick et al. [3] documentan esta fricción en entornos de autoservicio, identificando la dependencia de la capa semántica preparada por perfiles técnicos como uno de los obstáculos persistentes en la adopción real del BI por parte de usuarios de negocio.

En contextos de discovery analítico, donde el valor reside precisamente en la capacidad de iterar hipótesis con rapidez sobre datos nuevos, esta dependencia se convierte en una limitación estructural del paradigma.

El caso de uso que motiva este proyecto ilustra con precisión ese escenario. El dataset sobre el que opera el sistema desarrollado agrega información financiera transaccional de múltiples entidades bancarias para un conjunto de usuarios, con una semántica de negocio específica que no preexiste en ningún modelo BI construido. Además, en el entorno de trabajo correspondiente, las herramientas de BI no se encontraban integradas sobre estos datos, y construir una capa semántica ad hoc habría requerido una inversión técnica previa considerable, además de anticipar de forma exhaustiva las preguntas que los consultores podrían necesitar formular. Ninguna de estas condiciones era compatible con el carácter exploratorio del análisis ni con los tiempos del proyecto.

El BI, por tanto, no representa una solución insuficiente en términos absolutos, sino una solución diseñada para un problema distinto. Su arquitectura optimiza el consumo eficiente de análisis recurrente a costa de la flexibilidad ante lo desconocido, lo que hace que se descarte para el problema que este proyecto pretende abordar.

La sección siguiente examina los intentos más sistemáticos de reducir esta brecha entre la pregunta de negocio y el acceso flexible al dato mediante interfaces en lenguaje natural sobre bases de datos, una línea de trabajo con historia propia y con sus propias limitaciones no resueltas.

2.2. Interfaces en lenguaje natural sobre bases de datos

Las interfaces en lenguaje natural sobre bases de datos, conocidas en la literatura como Natural Language Interfaces to Databases (NLIDB), surgen como una respuesta directa a la distancia entre la pregunta formulada por un usuario y la consulta formal que exige una base de datos. Frente al modelo de BI descrito en la sección anterior, donde el acceso analítico depende de paneles y recorridos previamente definidos, las NLIDB persiguen que el usuario exprese una necesidad de información en lenguaje natural y que el sistema la transforme en una consulta ejecutable. Esta aproximación resulta especialmente relevante para el problema tratado en este TFM, ya que desplaza el foco desde la visualización de indicadores ya preparados hacia la traducción de intención analítica en operaciones sobre datos estructurados.

El reto técnico principal de estas interfaces se conoce como semantic parsing (análisis semántico), entendido como el proceso de convertir una expresión en lenguaje natural en una representación lógica que pueda ser interpretada por un sistema computacional. En el caso de bases de datos relacionales, esta representación suele adoptar la forma de una consulta SQL. La dificultad no reside únicamente en reconocer palabras clave, sino en alinear entidades mencionadas por el usuario con elementos del esquema, inferir filtros implícitos, identificar agregaciones y seleccionar relaciones entre tablas. Por ejemplo, una pregunta aparentemente simple puede requerir decisiones sobre qué entidad se analiza, a qué nivel de detalle, bajo qué condición temporal y con qué métrica. Por tanto, desde sus primeras formulaciones, las NLIDB plantean un problema de correspondencia entre lenguaje, estructura de datos y significado de negocio.

Los primeros sistemas de este tipo, como LUNAR o CHAT-80, mostraron que era posible consultar bases de datos mediante lenguaje natural en dominios acotados. LUNAR se diseñó para responder preguntas sobre análisis químicos de muestras lunares, mientras que CHAT-80 permitió transformar preguntas en inglés en consultas lógicas sobre información geográfica. Estos sistemas resultaron relevantes porque demostraron la viabilidad conceptual del enfoque, pero también evidenciaron una restricción que ha acompañado a esta línea de trabajo durante décadas: su rendimiento dependía de dominios cuidadosamente delimitados, esquemas conocidos y reglas lingüísticas o semánticas adaptadas al caso concreto [5]. La accesibilidad mejoraba para el usuario final, pero la flexibilidad quedaba condicionada por el esfuerzo previo de modelado y por la dificultad de trasladar la solución a nuevos contextos.

Con el desarrollo de modelos estadísticos y neuronales, el campo evolucionó ha-

cia tareas de Text-to-SQL (texto a SQL) evaluadas mediante benchmarks (bancos de prueba) estandarizados. WikiSQL introdujo un marco de evaluación a gran escala para generar consultas SQL a partir de preguntas en lenguaje natural sobre tablas individuales, lo que permitió comparar modelos de forma sistemática [6]. Posteriormente, Spider elevó la dificultad al incluir consultas más complejas, múltiples dominios y bases de datos no vistas durante el entrenamiento, con 10.181 preguntas, 5.693 consultas SQL únicas y 200 bases de datos distribuidas en 138 dominios [7]. Esta evolución fue decisiva porque obligó a evaluar no solo la capacidad de memorizar patrones, sino la generalización a esquemas nuevos.

Las métricas utilizadas en estos benchmarks ayudan a precisar qué significa que un sistema genere una consulta correcta. Exact Match (coincidencia exacta) mide si la consulta producida coincide estructuralmente con una consulta de referencia, mientras que Execution Accuracy (precisión de ejecución) evalúa si la consulta generada devuelve el resultado esperado al ejecutarse sobre la base de datos. Esta distinción es relevante para sistemas analíticos empresariales. Esto se debe a que una consulta puede no coincidir literalmente con la referencia y, aun así, producir un resultado correcto; del mismo modo, una consulta ejecutable puede devolver una cifra plausible pero no alineada con la intención de negocio si se ha elegido mal la granularidad, la población analizada o la definición de la métrica. Por ello, estas métricas aportan una base necesaria para la precisión cuantitativa, aunque no agotan los requisitos de auditabilidad y adaptación al dominio exigidos en el contexto de este proyecto. A medida que los sistemas Text-to-SQL se aproximaron a escenarios más complejos, la investigación dejó de centrarse únicamente en generar una consulta plausible y empezó a aislar subproblemas más concretos. Uno de ellos fue la relación entre la pregunta y el esquema de la base de datos. RAT-SQL abordó este punto mediante schema linking (vinculación con el esquema), representando de forma explícita las relaciones entre los términos de la pregunta, las tablas, las columnas y la estructura relacional. Sus resultados en Spider, con un 57,2 % de Exact Match y un 65,6 % al incorporar BERT, reflejan un avance relevante, pero también muestran que incluso con una modelización explícita del esquema la tarea seguía lejos de estar resuelta [8].

Otra fuente de error aparecía después, durante la generación de la consulta. Aunque el modelo identificase correctamente parte del esquema, podía producir SQL inválido o incompleto. PICARD respondió a esta limitación restringiendo el proceso de decoding (decodificación), de modo que se rechazaban tokens incompatibles con una consulta sintácticamente válida [9]. Luego apareció RESDSQL, que llevó esta separación un paso más allá al distinguir entre la selección de los elementos relevantes del esquema y la construcción del esqueleto lógico de la consulta [10].

Estos avances muestran que el problema no se reduce a disponer de una interfaz

cómoda para preguntar. A medida que los sistemas se aproximan a consultas reales, aparecen obstáculos más profundos: vincular lenguaje y esquema, controlar la validez formal de la salida, elegir la estructura correcta de la consulta y evaluar si el resultado responde realmente a la intención inicial. En un entorno de discovery financiero, esta dificultad se amplifica porque la lógica analítica no siempre está contenida explícitamente en el esquema de datos. Las preguntas pueden depender de definiciones de negocio, convenciones sobre categorías transaccionales, criterios de segmentación o supuestos sobre el nivel de agregación adecuado. Un modelo entrenado para optimizar métricas de benchmark puede capturar parte de la traducción formal, pero no garantiza por sí solo que la respuesta sea interpretable, verificable y coherente con las restricciones del dominio.

Por este motivo, las NLIDB y los modelos pre-LLM constituyen un antecedente técnico fundamental, pero no una solución suficiente para el sistema desarrollado. Su principal aportación reside en formular el problema de traducción entre lenguaje natural y consultas ejecutables, así como en introducir mecanismos de evaluación y control que siguen siendo relevantes. Sin embargo, el caso de uso de este TFM requiere un flujo más amplio que la generación aislada de SQL: se necesita interpretar la intención de negocio, apoyarse en conocimiento del dominio, producir operaciones verificables, ejecutar sobre datos reales y exponer la lógica seguida. La siguiente sección analiza cómo los Large Language Models (LLM) o grandes modelos de lenguaje amplían la flexibilidad de esta aproximación, al tiempo que introducen nuevos riesgos de fiabilidad, consistencia semántica y control.

2.3. Generación de SQL con LLMs

La evolución reciente de Text-to-SQL (texto a SQL) está marcada por la incorporación de Large Language Models (LLM), o grandes modelos de lenguaje, capaces de procesar lenguaje natural y generar código a partir de instrucciones y contexto. En la sección anterior se ha observado que las interfaces clásicas en lenguaje natural y los modelos especializados necesitaban arquitecturas explícitas para vincular la pregunta con el esquema, controlar la sintaxis y construir consultas ejecutables. Con los LLMs, parte de esa capacidad pasa a apoyarse en modelos que pueden interpretar descripciones del dominio y producir SQL sin requerir una arquitectura cerrada para cada base de datos. Esta transición resulta relevante para el presente TFM porque permite abordar preguntas de negocio más flexibles que las previstas por un modelo semántico fijo, aunque no elimina la necesidad de controlar la lógica analítica generada.

Un LLM aporta valor en este contexto porque puede combinar lenguaje natural y código dentro de una misma interacción. A diferencia de los enfoques pre-LLM, que dependían de componentes específicamente diseñados para cada subtarea, estos modelos pueden recibir una descripción del esquema, instrucciones sobre el formato esperado y restricciones sobre la consulta que debe producirse. Esta capacidad reduce parte de la rigidez de los sistemas anteriores, pero introduce un compromiso claro. Cuanto mayor es la libertad del modelo para interpretar la pregunta, mayor es también el riesgo de que genere una consulta plausible pero incorrecta desde el punto de vista del dominio o de la granularidad.

La literatura reciente ha identificado que una única llamada al modelo resulta frágil cuando la consulta requiere varias decisiones intermedias. La generación de SQL mejora cuando la tarea se divide en pasos más controlables, como comprender el esquema y descomponer la pregunta antes de producir la consulta final. DIN-SQL ejemplifica esta línea mediante decomposed in-context learning (aprendizaje en contexto descompuesto), donde el problema se estructura en fases sucesivas para reducir ambigüedad y facilitar la corrección. Sus resultados muestran una mejora relevante frente a prompting simple, con un 85,3 % de Execution Accuracy en Spider y un 55,9 % en BIRD [11]. Esta evidencia respalda una idea central para el diseño del sistema desarrollado, ya que pedir directamente una consulta a un LLM no ofrece suficiente control cuando la pregunta exige razonamiento analítico.

También se ha observado que el rendimiento no depende únicamente del modelo utilizado, sino de cómo se le presenta la información. La representación del esquema y la selección de ejemplos condicionan la capacidad del LLM para identificar qué elementos de la base de datos son pertinentes. DAIL-SQL estudia precisamente esta dimensión del problema y muestra que una construcción cuidadosa del prompt puede mejorar la generación de consultas, alcanzando un 86,6 % de Execution Accuracy en Spider con atención explícita a la eficiencia en el uso de tokens [12]. Esta conclusión es especialmente importante en un sistema empresarial, donde el contexto disponible puede ser extenso y donde introducir información irrelevante puede degradar tanto la precisión como la interpretabilidad del resultado.

Antes de adoptar Text-to-SQL como enfoque principal, debe distinguirse de Table Question Answering (Table QA), o respuesta a preguntas sobre tablas. En Table QA, el modelo intenta producir directamente una respuesta a partir de una tabla, como ocurre en líneas de trabajo representadas por TAPAS y TaPEx [13][14]. Este planteamiento puede ser adecuado para tablas pequeñas o preguntas cerradas, ya que evita exponer al usuario una consulta formal. Sin embargo, presenta una limitación estructural para el caso estudiado. El sistema desarrollado trabaja sobre cientos de miles de registros y necesita que los resultados numéricos puedan revisarse, reprodu-

cirse y depurarse. Text-to-SQL resulta más adecuado porque transforma la pregunta en una operación explícita, delega el cálculo al motor de datos y conserva una traza verificable entre la intención inicial y la respuesta obtenida.

Por todo ello, los LLMs hacen viable una interacción más flexible con datos estructurados, pero no convierten la generación de SQL en un problema resuelto. Su capacidad para interpretar preguntas abiertas amplía el alcance del análisis, mientras que su libertad generativa introduce riesgos de fiabilidad y consistencia semántica. Pueden aparecer columnas inexistentes, filtros incompletos o agregaciones incompatibles con la definición de la métrica. La conclusión que se deriva para este proyecto es que Text-to-SQL con LLMs constituye el enfoque más adecuado frente a alternativas basadas en Table QA, siempre que se integre en una arquitectura con validación, ejecución controlada y trazabilidad. La sección siguiente analiza precisamente estos requisitos de fiabilidad y evaluación, necesarios para que una respuesta generada pueda utilizarse con confianza en un contexto analítico profesional.

2.4. Fiabilidad y evaluación en asistentes analíticos

La fiabilidad de un asistente analítico no puede evaluarse con los mismos criterios que una respuesta conversacional generalista. En un sistema que opera sobre datos estructurados, una explicación fluida puede ocultar una consulta mal formulada, una métrica aplicada de forma incorrecta o una interpretación errónea del nivel de análisis. Esta diferencia es especialmente relevante en el contexto del presente TFM, donde las respuestas se apoyan en datos financieros de gran volumen y pueden influir en procesos de análisis de negocio. Por ello, la calidad del sistema no depende únicamente de que el modelo genere una respuesta comprensible, sino de que exista una cadena verificable entre la pregunta inicial, la consulta ejecutada y el resultado presentado.

Los modos de fallo más relevantes en este tipo de sistemas aparecen cuando el modelo interpreta el esquema o la intención analítica de forma incorrecta. Un primer caso es la alucinación de campos, donde se genera una consulta que hace referencia a elementos inexistentes o no disponibles en el dataset. Este error suele detectarse durante la ejecución, pero revela una dependencia excesiva de la memoria interna del modelo frente al contexto real proporcionado. Más problemáticos son los errores de granularidad, ya que una consulta puede ejecutarse correctamente y aun así responder a una pregunta distinta de la formulada. En datos financieros, confundir niveles como usuario y transacción puede alterar de forma sustancial el resultado numérico. Algo

similar ocurre con agregaciones mal planteadas, donde una media, suma o conteo puede ser técnicamente válido, pero incoherente con la definición de negocio que se pretende medir.

Modo de fallo	**Efecto analítico**	**Control necesario**
Campos inexistentes	La consulta no puede ejecutarse o parte de un esquema incorrecto	Catálogo de datos y validación sintáctica
Granularidad incorrecta	El resultado es plausible, pero responde a otro nivel de análisis	Definiciones de negocio y revisión semántica
Agregación inadecuada	La cifra obtenida no representa la métrica solicitada	Capa semántica y pruebas de ejecución
Falta de trazabilidad	El error no puede reconstruirse ni depurarse con precisión	Registro de consulta, resultado y supuestos

Cuadro 2.1: Principales modos de fallo en asistentes analíticos sobre datos estructurados

La mitigación de estos fallos exige mecanismos que limiten el espacio de decisión del modelo y permitan comprobar sus salidas. Un catálogo de datos proporciona una descripción controlada de las entidades disponibles, lo que reduce la probabilidad de que el LLM infiera campos inexistentes o relaciones no soportadas por el dataset. La capa semántica cumple una función distinta, ya que estabiliza definiciones métricas y criterios de agregación para que preguntas equivalentes no produzcan interpretaciones divergentes. A su vez, la validación de consultas permite comprobar si el SQL generado es compatible con el motor de ejecución y si puede ejecutarse sobre los datos reales. La observabilidad completa este ciclo al registrar preguntas, consultas y errores, de manera que el comportamiento del sistema pueda analizarse y mejorarse con evidencia.

La evaluación de estos sistemas también requiere estrategias específicas. Una manera razonablemente extendida es el uso de LLM-as-judge, o evaluación mediante un LLM como juez. Estos métodos pueden resultar útiles para valorar la claridad de una explicación o su alineación con una rúbrica, como se plantea en trabajos sobre evaluación de respuestas conversacionales y generación de texto [15][16]. Sin embargo, su utilidad para certificar resultados cuantitativos es limitada. Un evaluador lingüístico puede considerar satisfactoria una respuesta bien redactada aunque la cifra proceda de una consulta incorrecta. Además, se han documentado sesgos asociados al orden de presentación y a la preferencia por respuestas más extensas, lo que refuerza

la necesidad de separar la evaluación de la explicación final de la validación objetiva del cálculo.

La conclusión que se deriva para el sistema desarrollado es que la fiabilidad debe diseñarse como una propiedad de la cadena analítica completa. La respuesta final solo adquiere valor si puede vincularse a una consulta válida, ejecutada sobre los datos adecuados y coherente con las definiciones del dominio. Esta exigencia desplaza el diseño desde una interacción monolítica con el modelo hacia un flujo compuesto por pasos verificables. La sección siguiente analiza precisamente esa necesidad de orquestar interpretación, interacción con los datos y validación dentro de workflows con LLMs, donde cada fase del proceso puede controlarse de forma explícita.

2.5. Orquestación agéntica y workflows con LLMs

La sección anterior ha mostrado que la fiabilidad de un asistente analítico depende de una cadena verificable de decisiones, no de una respuesta generada de forma aislada. Si el sistema debe interpretar una pregunta, consultar contexto, generar SQL, ejecutar la consulta y revisar posibles errores, una única llamada al Large Language Model (LLM) resulta insuficiente como patrón arquitectónico. La cuestión deja de ser únicamente qué modelo se utiliza y pasa a centrarse en cómo se organiza el proceso que rodea al modelo. En este punto aparece la orquestación agéntica, entendida como la coordinación de pasos, herramientas y estados intermedios para transformar una intención analítica en una acción verificable sobre datos estructurados.

Conviene distinguir entre pipeline (flujo secuencial), agentic workflow (flujo de trabajo agéntico) y agente autónomo. Un pipeline ejecuta una secuencia fija de pasos, lo que facilita el control y la depuración, aunque limita la adaptación ante preguntas ambiguas o errores de ejecución. Un agentic workflow mantiene una estructura definida, pero permite que el modelo tome decisiones acotadas dentro del flujo, como seleccionar una herramienta o reformular una consulta fallida. Un agente autónomo opera con mayor libertad y decide su propia secuencia de acciones, lo que puede ser útil en tareas abiertas, pero resulta menos adecuado cuando se trabaja con datos financieros y se exige trazabilidad. Para el caso de uso tratado en este TFM, el punto de equilibrio se encuentra en un workflow controlado, donde el modelo participa en decisiones concretas sin perder los límites impuestos por la arquitectura.

La evolución hacia estos flujos se apoya en trabajos que han explorado cómo hacer que los LLMs razonen y actúen de forma más estructurada. Chain-of-Thought (CoT), o cadena de razonamiento, mostró que descomponer una tarea compleja en pasos intermedios puede mejorar el rendimiento del modelo en problemas que re-

quieren inferencia [17]. Sin embargo, en un asistente analítico no basta con producir razonamientos plausibles, ya que el resultado debe apoyarse en operaciones verificables. ReAct conecta mejor con esta necesidad al combinar razonamiento y acción, permitiendo que el modelo alterne entre interpretar el problema y ejecutar acciones sobre herramientas externas [18]. Esta idea resulta especialmente relevante en Text-to-SQL, donde el modelo debe pasar de una pregunta en lenguaje natural a una consulta que pueda ejecutarse sobre un motor de datos.

El concepto de tool use (uso de herramientas) formaliza esta separación entre generación lingüística y operación determinista. En lugar de pedir al modelo que calcule una cifra o simule una consulta, se le permite invocar componentes externos que realizan acciones concretas, como validar SQL o ejecutar una consulta. Toolformer mostró que los modelos pueden aprender a utilizar APIs externas para ampliar sus capacidades sin modificar directamente sus parámetros [19]. Gorilla abordó un problema complementario al mejorar la generación de llamadas a APIs y reducir errores en la selección de herramientas [20]. En sistemas empresariales, esta línea de trabajo tiene una implicación clara. El LLM debe coordinar el proceso, mientras que las operaciones críticas deben delegarse en herramientas cuya salida pueda comprobarse.

La implementación práctica de esta idea suele apoyarse en function calling (llamada a funciones), donde el modelo no devuelve únicamente texto libre, sino una invocación estructurada con argumentos definidos. Este patrón permite limitar qué acciones puede solicitar el LLM y qué información se entrega a cada componente del sistema. En el contexto del sistema desarrollado, esta separación es relevante porque el cálculo sobre datos financieros no debe depender de la memoria interna del modelo, sino de consultas ejecutadas sobre el dataset disponible. La función del LLM se aproxima así a la de un coordinador semántico que interpreta la pregunta y activa herramientas, mientras que la obtención del resultado queda asociada a una operación explícita y reproducible.

Los mecanismos de autocorrección refuerzan esta visión del sistema como proceso iterativo. Reflexion propone que un agente utilice feedback verbal de intentos previos para mejorar decisiones posteriores sin reentrenar el modelo [21]. Self-Refine plantea una dinámica similar mediante generación, crítica y refinamiento de la salida inicial [22]. En un asistente Text-to-SQL, el feedback más útil no tiene por qué proceder de una evaluación subjetiva, sino de señales observables, como un error de sintaxis o una consulta sin resultados. Esta lógica justifica que el sistema no acepte necesariamente la primera consulta generada y que incorpore ciclos de revisión cuando la ejecución indique que la respuesta no es válida o no cubre la intención analítica.

Desde el punto de vista de implementación, los frameworks de orquestación ofrecen formas distintas de organizar estos flujos. LangGraph resulta adecuado para re-

presentar workflows con estado, transiciones explícitas y nodos especializados, lo que facilita controlar qué información circula entre fases y cómo se gestionan los errores. AutoGen, por otro lado, se orienta a interacciones entre múltiples agentes conversacionales, una aproximación útil para tareas colaborativas, pero menos directa cuando se busca una traza analítica compacta [23]. MetaGPT representa otra línea, basada en roles y procedimientos, que puede ser apropiada para tareas de desarrollo más amplias [24]. En este TFM, la prioridad arquitectónica es conservar control sobre el flujo y mantener estados intermedios revisables.

Enfoque	**Ventaja principal**	**Limitación para el caso de uso**
Pipeline	Control alto y ejecución predecible	Baja adaptación ante errores o ambigüedad
Agentic workflow	Flexibilidad acotada y trazabilidad del proceso	Requiere diseñar estados y reglas de transición
Agente autónomo	Alta capacidad de exploración	Menor control sobre acciones y decisiones intermedias

Cuadro 2.2: Comparación conceptual de patrones de orquestación para asistentes analíticos

La orquestación agéntica aporta valor cuando se entiende como una arquitectura de control y no como autonomía sin restricciones. En un sistema analítico, el modelo debe interpretar la pregunta y coordinar herramientas, pero el cálculo debe quedar asociado a operaciones ejecutables y trazables. Esta separación permite combinar accesibilidad conversacional con precisión cuantitativa, manteniendo una cadena revisable entre intención, consulta y resultado. Una vez establecido que el sistema requiere un workflow controlado, la siguiente cuestión es si esta capacidad debe construirse internamente o adoptarse mediante una solución comercial existente. Esa decisión se analiza en la sección siguiente mediante el marco build vs buy.

2.6. Alternativas comerciales y decisión *build vs. buy*

Una vez establecido que el asistente analítico requiere generación controlada de SQL, ejecución verificable y trazabilidad, la decisión arquitectónica no se limita a escoger un modelo. También debe evaluarse si conviene construir una solución específica o adoptar una herramienta comercial ya existente. Esta decisión, conocida

como *build vs. buy* (construir frente a adquirir), adquiere especial relevancia en el contexto del presente TFM porque el objetivo no consiste únicamente en habilitar una conversación con los datos. El sistema debe operar sobre un dataset financiero de gran tamaño, conservar control sobre la lógica analítica y producir salidas reutilizables en un entorno profesional.

El mercado ofrece varias familias de soluciones relacionadas con este objetivo. Los asistentes empresariales generalistas, como ChatGPT Enterprise, proporcionan una experiencia conversacional madura y compromisos de privacidad orientados a organizaciones, incluyendo la no utilización de datos empresariales para entrenamiento por defecto [25]. Sin embargo, este tipo de herramienta no está diseñada de forma específica para ejecutar un flujo analítico completo sobre cientos de miles de registros ni para mantener una traza estructurada entre pregunta, SQL, resultado y visualización. Además, durante el desarrollo del proyecto no se contaba necesariamente con una disponibilidad operativa e integrada de estas capacidades dentro del flujo de trabajo real, lo que hacía poco razonable plantearlas como solución principal.

Otra familia relevante corresponde a los copilots integrados en plataformas de Business Intelligence (BI), como Copilot en Power BI. Estas soluciones mejoran la accesibilidad al permitir que usuarios de negocio interactúen con informes, modelos semánticos y activos de datos mediante lenguaje natural [26]. Su principal fortaleza aparece cuando la organización ya dispone de una capa semántica preparada y de datos integrados en el ecosistema correspondiente. En el caso estudiado, el dataset no partía de un modelo BI productizado ni de una infraestructura analítica ya configurada para este propósito. Adoptar esta vía habría desplazado el esfuerzo hacia la preparación previa del entorno, sin resolver de forma directa la necesidad de controlar la generación de consultas y la validación de resultados.

Las plataformas de datos con asistentes nativos se acercan más al problema tratado. Databricks Genie permite que usuarios de negocio formulen preguntas en lenguaje natural sobre activos creados dentro de Databricks [27]. Snowflake Cortex Analyst, por su parte, ofrece una funcionalidad basada en LLMs para responder preguntas de negocio sobre datos estructurados en Snowflake mediante modelos semánticos configurados para ese entorno [28]. Estas soluciones muestran una dirección clara del mercado hacia interfaces analíticas conversacionales, pero presuponen que el dato reside o se modela dentro de la plataforma correspondiente. Para el alcance del TFM, esa condición no estaba alineada con el flujo técnico elegido, basado en ejecución local controlada y en una arquitectura específica para el dataset disponible.

Familia de solución	**Fortaleza principal**	**Limitación para el caso de uso**
Asistentes empresariales generalistas	Alta accesibilidad y soporte para razonamiento asistido	No ejecutan de forma nativa un flujo analítico completo sobre cientos de miles de registros
Copilots de BI	Integración con informes y modelos semánticos existentes	Requieren datos ya modelados dentro del ecosistema BI
Asistentes de plataformas de datos	Mayor proximidad a Text-to-SQL y gobierno de datos	Exigen alojar o modelar el dataset dentro de la plataforma correspondiente
BI aumentado con IA	Facilita discovery sobre métricas preparadas	Menor control sobre consultas, validación y visualización específica del proyecto

Cuadro 2.3: Comparación de familias comerciales frente al caso de uso del TFM

La comparación anterior muestra que las alternativas comerciales cubren parcialmente los criterios definidos al inicio del capítulo. Suelen aportar accesibilidad y reducen fricción para el usuario, pero el caso de uso exige también precisión cuantitativa, adaptación al dominio y control completo de la cadena analítica. Esta diferencia se aprecia especialmente en dos aspectos prácticos. En primer lugar, el volumen del dataset impide tratar los datos como una tabla que se introduce completa en el contexto de un asistente generalista. En segundo lugar, el resultado esperado no es únicamente una respuesta textual, sino una salida analítica que puede incluir tablas agregadas y gráficos generados a partir de resultados ejecutados. La capacidad de adaptar estas visualizaciones al estilo corporativo forma parte del valor del sistema, ya que permite que el análisis sea reutilizable en entregables profesionales.

Adicionalmente, el marco regulatorio europeo refuerza esta necesidad de control, aunque no implica descartar automáticamente soluciones comerciales. El General Data Protection Regulation (GDPR), o Reglamento General de Protección de Datos, establece un marco exigente para el tratamiento de datos personales y aplica con independencia de la tecnología utilizada [29]. El Artificial Intelligence Act (AI Act), o Reglamento Europeo de Inteligencia Artificial, entró en vigor el 1 de agosto de 2024 y orienta el desarrollo y despliegue responsable de sistemas de IA en la Unión Europea [30]. En el ámbito financiero, PSD2 promueve innovación y seguridad en servicios de pago, mientras que DORA aplica desde el 17 de enero de 2025 para reforzar la resiliencia digital de entidades financieras y proveedores tecnológicos [31][32]. Estas normas no determinan por sí solas la arquitectura del TFM, pero sí hacen necesario

justificar dónde se procesan los datos, qué trazas se conservan y cómo se gobierna el uso de herramientas externas.

La decisión *build vs. buy* se inclina, por tanto, hacia construir una solución específica para el alcance del proyecto. Esta elección no supone negar el valor de las herramientas comerciales, que resultan adecuadas cuando los datos ya residen en sus plataformas y existe una capa semántica preparada. En este caso, construir internamente permite controlar el motor de ejecución, adaptar el catálogo al dominio financiero y conservar la traza entre pregunta, consulta y resultado. También permite integrar la generación de visualizaciones dentro del mismo flujo analítico, sin depender de una plataforma no disponible o no integrada durante el desarrollo. La revisión de estas alternativas confirma que el estado del arte ofrece componentes valiosos, pero no una solución completa ajustada a las restricciones concretas del presente problema. Esta brecha se sintetiza en la sección siguiente, donde se derivan los requisitos que debe satisfacer el sistema desarrollado.

2.7. Síntesis y brecha del estado del arte

El recorrido realizado en este capítulo muestra que el acceso analítico conversacional se ha abordado desde aproximaciones complementarias.

La brecha identificada no se encuentra en la ausencia de tecnología, sino en la dificultad de combinar esas capacidades bajo las restricciones concretas del caso de uso. El sistema debe permitir que usuarios de negocio formulen preguntas abiertas sobre datos financieros cross-bank, pero también debe producir resultados cuantitativos verificables. Debe ofrecer flexibilidad analítica, pero sin perder control sobre la consulta generada. Debe facilitar el acceso al dato, pero manteniendo trazabilidad, adaptación al dominio y gobierno de la información. Esta simultaneidad es precisamente el punto que ninguna de las familias revisadas resuelve de forma completa para el contexto del presente TFM.

De la revisión anterior se derivan los siguientes requisitos de alto nivel para el sistema desarrollado:

- permitir que usuarios de negocio formulen preguntas en lenguaje natural sobre datos financieros estructurados;
- traducir esas preguntas a consultas SQL ejecutables sobre el motor de datos definido;
- validar la consulta antes de presentar resultados al usuario;

- conservar una traza entre pregunta, consulta, resultado y explicación;
- incorporar definiciones de dominio y criterios de agregación consistentes;
- mantener control sobre el dato y sobre la información enviada al modelo;
- generar salidas reutilizables, incluyendo tablas o visualizaciones cuando el análisis lo requiera.

Estos requisitos delimitan la contribución técnica del proyecto. No se plantea sustituir las herramientas existentes en todos sus ámbitos de aplicación, sino construir una arquitectura ajustada a un problema concreto de analítica financiera exploratoria. El capítulo siguiente presenta el diseño del sistema desarrollado, describiendo cómo se organiza el flujo entre interpretación de la pregunta, generación de consultas, ejecución, validación y presentación de resultados.

Capítulo 3

Análisis del Problema y Diseño de la Solución

3.1. Descripción del problema real

3.1.1. El dataset: naturaleza y alcance

El punto de partida del presente proyecto es un dataset de naturaleza financiera agregada, proporcionado de forma periódica por una empresa española especializada en la agregación de datos bancarios. El dataset recoge información sobre los productos financieros activos de una muestra de usuarios españoles en múltiples entidades bancarias de forma simultánea, lo que le otorga una característica analíticamente singular: visibilidad transversal del comportamiento financiero de un mismo individuo a través de más de cincuenta entidades distintas. Esta perspectiva multi-entidad, habitualmente denominada visión *cross-bank*, no está disponible para ninguna entidad de forma individual, ya que cada banco únicamente tiene acceso a los productos contratados con ella.

Las características estructurales del dataset se resumen en la Tabla 3.1.

Característica	Valor
Número de registros	Más de 1,8 millones
Usuarios únicos	Aproximadamente 339.000
Entidades bancarias	Más de 50 entidades españolas
Cobertura geográfica	España
Ventana temporal	Aproximadamente 3 meses por entrega
Granularidad	Usuario × entidad × producto
Frecuencia de actualización	Periódica, mediante entregas sucesivas

Cuadro 3.1: Características estructurales del dataset

Cada registro representa el cruce de un usuario con un producto financiero concreto en una entidad específica, de modo que un mismo usuario puede aparecer múltiples veces si mantiene productos en varias entidades o varios productos en la misma. Los tipos de productos cubiertos incluyen:

- Productos de liquidez y pago: cuentas corrientes, tarjetas de crédito y tarjetas de débito.
- Productos de financiación: hipotecas, préstamos personales y líneas de crédito.
- Productos de ahorro e inversión: depósitos, fondos de inversión, planes de pensiones y productos de renta variable.
- Productos de protección: seguros vinculados a hipotecas, con información adicional sobre la compañía aseguradora asociada.

La información no refleja únicamente la actividad del periodo más reciente, sino el estado activo de cada producto en el momento de la extracción, con independencia de su antigüedad. Una hipoteca contratada varios años antes permanece visible en el dataset mientras no haya sido cancelada, lo que permite analizar tanto el comportamiento de contratación reciente como la cartera de productos consolidada de cada usuario.

Los datos de usuario están anonimizados mediante agrupación en tramos. Variables como la edad, el nivel de renta o la localización geográfica no se expresan con valores exactos sino mediante rangos predefinidos, lo que preserva la privacidad individual sin eliminar la capacidad de segmentación analítica. Esta estructura impone condiciones específicas sobre cómo deben formularse las consultas analíticas:

los cálculos de distribución o tendencia central deben tener en cuenta la naturaleza discreta de las variables segmentadas, lo que introduce una capa de complejidad semántica que cualquier sistema de acceso al dato debe ser capaz de gestionar correctamente.

3.1.2. Perfiles de usuario y contexto de uso

El sistema desarrollado está orientado a perfiles de usuario internos de una consultora estratégica de primer nivel. Los destinatarios principales son directivos de la firma responsables de la formulación de propuestas comerciales a entidades financieras. En ese contexto, el acceso a análisis cuantitativos sobre el comportamiento financiero de los usuarios españoles representa un activo diferencial: permite sustentar hipótesis de negocio con evidencia empírica, identificar oportunidades de mercado y dimensionar el alcance de posibles iniciativas estratégicas para los clientes. Con frecuencia, estos análisis deben incorporarse a propuestas que se elaboran en plazos de días, no de semanas.

Un segundo grupo de usuarios lo conforman consultores que participan activamente en proyectos con entidades financieras, para quienes el dataset constituye un insumo analítico recurrente durante el desarrollo del encargo. Ambos perfiles comparten tres características que condicionan de forma determinante el diseño del sistema:

- Ausencia de formación técnica para interactuar directamente con los datos en formato *raw*, incluyendo la escritura de consultas SQL u otros lenguajes de análisis.
- Imposibilidad práctica de dedicar tiempo a aprender un lenguaje de consulta o a operar herramientas de análisis de datos.
- Exigencia de resultados en plazos incompatibles con los ciclos habituales de análisis técnico, condicionados por los ritmos propios de la actividad de consultoría estratégica.

La combinación de estas tres restricciones, alta demanda analítica, baja capacidad técnica y plazos reducidos, define el espacio del problema con precisión y descarta de entrada las soluciones que requieren formación técnica previa o tiempos de configuración prolongados.

3.1.3. El proceso de acceso al dato antes del sistema

Antes del desarrollo del sistema descrito en este trabajo, el acceso a los datos requería la intervención de un perfil técnico de la firma, habitualmente dedicado a otros proyectos con cliente. El proceso seguía una secuencia que, lejos de ser excepcional, constituía la norma operativa habitual, y cuyas etapas se representan a continuación:

1. El consultor o directivo elaboraba una descripción de los resultados esperados, frecuentemente en formato de presentación, detallando qué indicadores, gráficos o cálculos se requerían.

2. Esta descripción se trasladaba al perfil técnico, quien debía interpretar la intención analítica, identificar si los datos necesarios estaban disponibles y aclarar ambigüedades con el solicitante.

3. El perfil técnico desarrollaba el código de extracción y análisis en entornos de trabajo aislados, sin versionado ni documentación sistemática.

4. Los resultados se devolvían al solicitante para revisión, con frecuencia desencadenando nuevas iteraciones por discrepancias entre lo solicitado y lo entregado.

5. El ciclo se repetía hasta obtener resultados validados, con un número variable de iteraciones según la complejidad y la claridad inicial de la solicitud.

El coste de este proceso es cuantificable en varios planos. La Tabla 3.2 recoge una estimación conservadora basada en parámetros habituales en el sector.

Parámetro	**Estimación baja**	**Estimación alta**
Tiempo total por petición	2 días	5 días
Horas de perfil técnico	8 horas	15 horas
Coste hora perfil técnico	80 €	120 €
Coste directo por petición	640 €	1.800 €

Cuadro 3.2: Estimación del coste por petición analítica bajo el proceso manual

A estas cifras debe añadirse el coste de oportunidad derivado de desviar un perfil técnico de proyectos facturables, cuyo impacto económico real supera con frecuencia al coste directo. A escala de varias peticiones mensuales y múltiples equipos activos simultáneamente, el impacto agregado resulta considerable.

Más allá del coste directo, el proceso presentaba un problema estructural de trazabilidad. Los análisis se desarrollaban sin versionado ni documentación sistemática, de modo que reproducir un cálculo realizado semanas antes o responder a preguntas sobre la metodología empleada resultaba frecuentemente inviable. Esta opacidad introducía riesgo en contextos donde la credibilidad del análisis es tan importante como su contenido. El conjunto de estas fricciones no solo encarecía el acceso al dato, sino que desincentivaba la exploración: formular una hipótesis nueva implicaba iniciar de nuevo un ciclo costoso, lo que en la práctica limitaba el uso del dataset a un subconjunto reducido de análisis recurrentes, infrautilizando una fuente con un potencial analítico considerablemente mayor.

3.1.4. Formulación del problema

El problema que motiva este proyecto no es, en sentido estricto, un problema de acceso a datos: el dataset existe, está disponible y contiene información de alto valor analítico. El problema es de acceso analítico autónomo bajo condiciones reales de uso: usuarios sin formación técnica, plazos reducidos, necesidad de trazabilidad y datos que no pueden abandonar el entorno controlado de la organización. La solución no puede depender de intermediarios técnicos que no escalan, ni de herramientas genéricas que no respetan las restricciones de confidencialidad aplicables. De la caracterización de este problema se derivan directamente los requisitos funcionales y no funcionales del sistema, que se detallan en la sección siguiente.

3.2. Requisitos funcionales y no funcionales del sistema

Los requisitos que se recogen a continuación constituyen la traducción formal de cada fricción identificada: la dependencia de intermediarios técnicos, la ausencia de trazabilidad analítica, el carácter sensible del dato y la necesidad de un sistema que pueda evolucionar con el proyecto. Se distingue entre requisitos funcionales, que describen las capacidades del sistema, y requisitos no funcionales, que establecen las condiciones bajo las cuales dichas capacidades deben ejercerse.

3.2.1. Requisitos funcionales

Los requisitos funcionales (RF) se organizan en dos niveles de prioridad. Los requisitos críticos definen las capacidades sin las cuales el sistema no resuelve el pro-

blema central; los requisitos de calidad amplían la utilidad del sistema y determinan la experiencia de uso, pero no alteran su lógica nuclear.

ID	Prioridad	Descripción
RF1	Crítico	El sistema interpreta preguntas formuladas en lenguaje natural y las traduce a consultas SQL ejecutables, apoyándose en una capa semántica que codifica las definiciones de dominio y los criterios de agregación del dataset.
RF2	Crítico	Tras la ejecución de cada consulta, el sistema devuelve una respuesta en lenguaje natural que incluye los resultados numéricos y una explicación del procedimiento seguido para obtenerlos, garantizando la trazabilidad entre pregunta, consulta y resultado.
RF3	Crítico	El sistema identifica y comunica explícitamente las preguntas que se encuentran fuera del alcance del dataset, evitando que respuestas incompletas o incorrectas se presenten como válidas.
RF4	Calidad	Cuando el usuario lo solicite o el sistema lo considere pertinente, se generan visualizaciones interactivas de los resultados, siguiendo el estilo visual corporativo definido. El usuario puede alternar entre representación gráfica y tabular, y exportar los datos resultantes en formato compatible con hojas de cálculo.
RF5	Calidad	Los resultados obtenidos admiten operaciones de filtrado, ordenación ascendente y descendente, y cambio de unidades, sin necesidad de reformular la consulta original.
RF6	Calidad	El sistema incorpora un mecanismo de retroalimentación que permite al usuario valorar la calidad de cada respuesta mediante una valoración binaria y un comentario opcional, registrando estos eventos para informar iteraciones de mejora del sistema.

Cuadro 3.3: Requisitos funcionales del sistema

3.2.2. Requisitos no funcionales

Los requisitos no funcionales (RNF) definen las condiciones bajo las cuales el sistema debe operar, derivadas del entorno real de uso. A diferencia de los requisitos

funcionales, cuyo incumplimiento hace el sistema incorrecto, el incumplimiento de un requisito no funcional lo hace inviable o inadecuado para el contexto al que se destina.

ID	Categoría	Descripción
RNF1	Seguridad	El acceso al sistema está restringido mediante autenticación por contraseña; solo usuarios autorizados pueden interactuar con él.
RNF2	Seguridad	Las consultas ejecutadas sobre el dataset son exclusivamente de lectura. Cualquier operación de escritura, modificación o eliminación queda bloqueada programáticamente, con independencia de la instrucción recibida.
RNF3	Seguridad	El volumen de filas devuelto por cualquier consulta está limitado a un máximo de 52 registros, impidiendo la extracción masiva del dataset a través de la interfaz conversacional.
RNF4	Seguridad	Los datos no abandonan el entorno controlado de la organización en ningún punto del flujo de ejecución.
RNF5	Fiabilidad	El flujo de ejecución no se interrumpe ante respuestas inesperadas del modelo. Todos los outputs intermedios se validan mediante esquemas estructurados antes de pasar al nodo siguiente.
RNF6	Fiabilidad	Las visualizaciones se renderizan siempre sin errores visibles para el usuario. El sistema instancia esquemas predefinidos cuyos campos el modelo rellena, asegurando que la compilación sea siempre válida.
RNF7	Mantenibilidad	El catálogo semántico del dataset puede actualizarse de forma independiente al código del sistema, permitiendo incorporar nuevas entregas periódicas sin intervención en la lógica de orquestación.
RNF8	Extensibilidad	La arquitectura permite incorporar nuevos nodos al flujo de procesamiento sin modificar los existentes, siguiendo un principio de composición modular que facilita la evolución incremental del sistema.
RNF9	Configuración	Todos los parámetros de configuración, incluyendo el modelo de lenguaje utilizado, se centralizan en un único fichero, desacoplando las decisiones de configuración de la lógica de negocio.
RNF10	Latencia	El tiempo de respuesta medio debe mantenerse por debajo de 60 segundos, para conservar una buena experiencia de usuario y conversación fluida, y resultar en una mejora sustancial respecto al proceso manual anterior, cuya latencia se medía en días.

Cuadro 3.4: Requisitos no funcionales del sistema

El conjunto de requisitos presentado define el contrato de diseño al que debe responder la arquitectura del sistema. Cada decisión tecnológica adoptada en las secciones siguientes puede rastrearse hasta uno o varios de estos requisitos: la elección del motor de ejecución SQL, el framework de orquestación, el mecanismo de validación de outputs y la estructura del catálogo semántico responden a compromisos específicos establecidos aquí. Conocidos los requisitos, la pregunta que sigue es cómo satisfacerlos, qué alternativas se evaluaron y por qué se tomaron las decisiones que configuran el diseño final del sistema.

3.3. Decisiones tecnológicas clave y alternativas consideradas

Las decisiones de diseño que configuran el sistema parten de los requisitos establecidos en la sección anterior. Para cada componente principal se evaluaron alternativas concretas, cuyos tradeoffs se analizan a continuación en orden decreciente de alcance arquitectónico: desde el paradigma de acceso al dato, que condiciona el resto de elecciones, hasta los componentes más periféricos del flujo.

3.3.1. Paradigma de acceso al dato

La primera decisión de diseño determina cómo el sistema transforma una pregunta en lenguaje natural en un resultado numérico verificable. Se evaluaron tres paradigmas distintos antes de adoptar Text-to-SQL como enfoque principal.

- **Table Question Answering (Table QA):** el modelo razona directamente sobre los datos sin generar una consulta formal. Este enfoque resulta inviable para el caso de uso por dos razones estructurales: el dataset supera 1,8 millones de registros, un volumen incompatible con cualquier ventana de contexto actual, y la ausencia de una consulta intermedia elimina la posibilidad de auditar el cálculo realizado, incumpliendo `RF2` y `RNF5`.
- **Generación de código Python:** el modelo produce código ejecutable en lugar de SQL. Aunque amplía el espacio de transformaciones posibles, introduce una superficie de error mayor, dificulta la restricción programática a operaciones de solo lectura exigida por `RNF2` y complica la validación estructural de outputs intermedios que requiere `RNF5`. La predictibilidad del SQL sobre un motor relacional resulta más adecuada para un sistema donde la fiabilidad del resultado es prioritaria.

- **Text-to-SQL sobre motor local:** el modelo genera una consulta SQL que se ejecuta sobre un motor de datos controlado. Este paradigma satisface simultáneamente la trazabilidad entre pregunta y resultado (`RF2`), la restricción a operaciones de lectura (`RNF2`) y la validación determinista antes de presentar el resultado al usuario (`RNF5`).

3.3.2. Motor de ejecución SQL

Establecido el paradigma Text-to-SQL, la siguiente decisión afecta al motor sobre el que se ejecutan las consultas generadas. Las alternativas evaluadas fueron Pandas, SQLite y soluciones de base de datos externas.

Pandas permite operar sobre archivos CSV desde Python, pero no expone una interfaz SQL nativa completa: las consultas generadas por el LLM deben traducirse a operaciones sobre DataFrames, lo que introduce una capa de conversión innecesaria y reduce la compatibilidad con SQL estándar. SQLite ofrece una interfaz SQL completa, pero su motor está optimizado para cargas transaccionales (OLTP), no para las agregaciones analíticas intensivas que caracterizan el caso de uso. Las bases de datos externas como PostgreSQL añaden dependencias de infraestructura incompatibles con el requisito de que los datos no abandonen el entorno controlado (`RNF4`).

DuckDB resuelve estas limitaciones como motor OLAP (Online Analytical Processing, o procesamiento analítico en línea) embebido, sin servidor y con integración nativa en Python. Ejecuta consultas SQL directamente sobre archivos CSV sin requerir carga previa completa en memoria, con un rendimiento muy superior a Pandas en operaciones analíticas sobre grandes volúmenes. Su soporte de SQL estándar completo, incluyendo funciones de ventana y agregaciones complejas, garantiza compatibilidad con el SQL generado por el LLM sin transformaciones intermedias. Esta combinación de características satisface `RNF4`, `RNF5` y `RF1`.

3.3.3. Framework de orquestación

El flujo del sistema requiere coordinar interpretación de la pregunta, consulta del catálogo semántico, generación de SQL, ejecución, validación y generación de la respuesta. Una cadena secuencial simple resulta insuficiente para gestionar estados intermedios, manejar errores por nodo e implementar lógica condicional entre pasos. Se evaluaron tres alternativas antes de adoptar LangGraph.

AutoGen y CrewAI se orientan a interacciones entre múltiples agentes conversacionales con mayor autonomía en la toma de decisiones. Esta filosofía resulta menos adecuada cuando se exige un flujo analítico predecible con trazabilidad explícita:

la autonomía no estructurada incrementa el riesgo de derivaciones no controladas en un sistema donde cada paso debe poder auditarse. LangGraph, por el contrario, representa el workflow como un grafo de estados con transiciones explícitas, estado tipado que fluye entre nodos y lógica condicional definida en el propio grafo. Esta arquitectura satisface directamente `RNF5` y `RNF8`: la validación puede realizarse en cada nodo de forma independiente, y añadir nuevos pasos al flujo no requiere modificar los existentes.

Adicionalmente, LangGraph cuenta con una adopción extensa en la comunidad de desarrollo de sistemas con LLMs, lo que facilita la comprensión del código por parte de otros miembros del equipo y reduce la curva de incorporación ante futuros cambios. La modularidad del framework resultó especialmente relevante durante el desarrollo, ya que el sistema evolucionó de forma incremental: comenzó con un número reducido de nodos y fue ampliándose a medida que las evaluaciones identificaba oportunidades de mejora, validando en la práctica la extensibilidad que `RNF8` exigía desde el diseño.

3.3.4. Modelo de lenguaje, capa semántica y visualización

La elección de GPT-4o como modelo resultó de las restricciones operativas del entorno: GPT-4o es el estándar de la organización y su integración con la infraestructura Azure disponible estaba ya establecida, lo que garantiza además que los datos se procesan dentro del entorno controlado exigido por `RNF4`. Desde el punto de vista de evolución futura, la arquitectura modular de LangGraph permitiría asignar modelos distintos a nodos distintos en función de la complejidad de cada paso, optimizando latencia y coste sin reescribir la lógica del sistema. Esta posibilidad no está implementada en la versión actual, pero forma parte del espacio de mejoras que la arquitectura hace accesible.

Para la representación del catálogo de datos, se evaluaron texto libre en el prompt, ficheros YAML sin tipado y soluciones tipo capa semántica externa. Pydantic se adopta porque ofrece validación automática de tipos en tiempo de ejecución, serialización directa a JSON para su inclusión en el contexto del LLM, integración nativa con el ecosistema Python y LangGraph, y actualización independiente al código del sistema, satisfaciendo `RNF7`. Esta última propiedad resulta especialmente relevante dado que el dataset se actualiza periódicamente: modificar el catálogo no requiere intervención en la lógica de orquestación.

La elección de Altair con especificaciones Vega-Lite para la generación de visualizaciones responde a una limitación técnica concreta. Las librerías basadas en código imperativo, como Matplotlib o Plotly, exponen al LLM a una superficie de generación donde cualquier error sintáctico produce un fallo de ejecución visible para el usua-

rio. Vega-Lite adopta una especificación declarativa en JSON donde el LLM rellena campos de un esquema predefinido en lugar de generar código libre, garantizando que la compilación sea siempre válida y satisfaciendo `RNF6`. Su interactividad nativa y su integración con Streamlit completan las razones de esta elección.

Las decisiones analizadas no son independientes entre sí. Como se ha visto en esta sección, cada elección refuerza las demás, de modo que el sistema resultante no es la suma de sus componentes sino una arquitectura donde la fiabilidad se construye en cada capa. El capítulo siguiente describe cómo estos componentes se implementan y se integran en el sistema final.

Capítulo 4

Implementación del Sistema

4.1. Visión general de la implementación

El sistema se implementa como un grafo de siete nodos especializados orquestados mediante LangGraph, donde cada nodo encapsula una responsabilidad funcional delimitada y se comunica con el resto exclusivamente a través de un estado compartido y tipado. Esta organización responde a cuatro principios de diseño que atraviesan toda la implementación.

- **Modularidad:** cada nodo es una función independiente, testeable y reemplazable sin afectar al resto del grafo.
- **Tipado fuerte:** toda la información que circula entre nodos pasa por el `AgentState`, un contrato Pydantic que valida estructura y tipos en tiempo de ejecución.
- **Separación de responsabilidades:** generación de SQL, ejecución, validación, análisis y visualización son pasos independientes con interfaces definidas, nunca lógica entremezclada.
- **Compilación única:** el grafo se compila una sola vez al arranque del sistema y se reutiliza en todas las invocaciones, lo que elimina overhead de inicialización por consulta.

Nodos del grafo y flujo de control

La Figura 4.1 presenta el grafo completo del sistema. El flujo parte de `sql_generator_node` y avanza a través de siete nodos especializados hasta `__end__`, con una estructura predominantemente lineal que se describe en detalle en la sección siguiente.

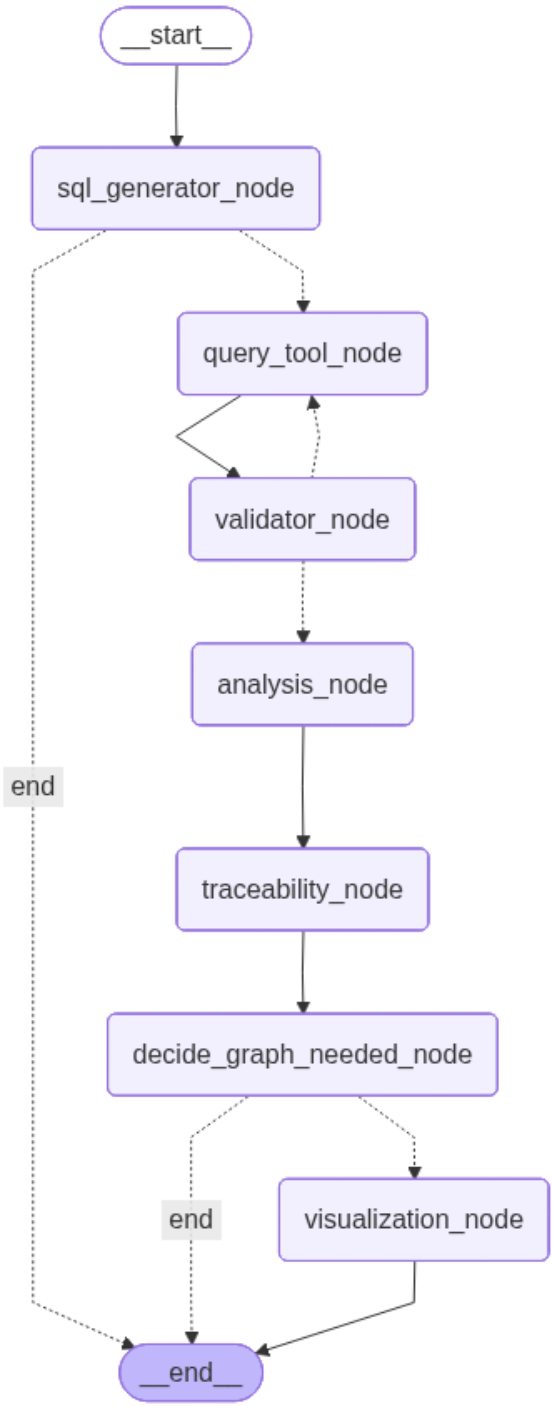


Figura 4.1: Grafo de ejecución del sistema implementado en LangGraph

La función de cada nodo se resume en el Cuadro 4.1.

Nodo	**Responsabilidad**
`sql_generator_node`	Interpreta la pregunta del usuario y genera la consulta SQL correspondiente, apoyándose en el catálogo semántico del dataset.
`query_tool_node`	Ejecuta la consulta sobre DuckDB y devuelve el resultado al estado compartido.
`validator_node`	Evalúa si el resultado es válido y coherente con la intención analítica de la pregunta. En caso de error, corrige la query y la envia a `query_tool_node` para ejecutarla nuevamente.
`analysis_node`	Transforma el resultado numérico en una respuesta interpretativa en lenguaje natural para los stakeholders.
`traceability_node`	Genera la explicación del procedimiento seguido para obtener el resultado con el fin de facilitar auditabilidad.
`decide_graph_need_node`	Determina si la respuesta requiere una visualización gráfica, en función de si el usuario la solicita o si sería de utilidad.
`visualization_node`	Construye la especificación Vega-Lite del gráfico cuando el nodo anterior lo requiere.

Cuadro 4.1: Nodos del grafo y su responsabilidad funcional

Cabe destacar que la función `build_agent_graph()` que construye el grafo está decorada con `@lru _cache(maxsize=1)`, lo que garantiza que la compilación se realiza una sola vez al arranque del sistema y el grafo resultante se reutiliza en todas las invocaciones posteriores, lo que minimiza latencia. Sin embargo, esto implica que la topología del grafo no puede modificarse en tiempo de ejecución, pero esa restricción no tiene relevancia práctica dado que la estructura del flujo es estable.

Puntos de decisión y naturaleza agéntica del flujo

El flujo descrito no ejecuta siempre los mismos pasos en el mismo orden. El flujo de ejecución parte de `sql_generator_node` y avanza de forma predominantemente

lineal, pero en tres puntos el LLM razona sobre el contexto disponible y determina qué ocurre a continuación. El primero ocurre en el propio `sql_generator_node`, donde el modelo evalúa cómo debe responderse la pregunta con el contexto disponible. El segundo es un ciclo de reintento entre `query_tool_node` y `validator_node`, activado cuando el resultado de la ejecución no supera la validación. El tercero es una bifurcación en `decide_graph_need_node`, que determina si la ejecución continúa hacia `visualization_node` o concluye directamente.

Estas bifurcaciones son las que convierten el sistema en un *agentic workflow* (flujo de trabajo agéntico), en el sentido introducido en el Capítulo 2: el modelo participa en decisiones de routing, pero dentro de un espacio de opciones definido por la arquitectura del grafo. El Cuadro 4.2 detalla cada punto de decisión.

Nodo	Decisión	Opciones	Criterio de decisión
`sql_generator_node`	¿Cómo debe responderse la pregunta con el contexto disponible?	01- Responde directamente usando el catálogo semántico, sin ejecutar consulta. 02- Genera una consulta SQL y la pasa a `query_tool_node`. 03- Declara la pregunta fuera de alcance y termina.	El LLM evalúa si la pregunta puede responderse con el conocimiento del catálogo (por ejemplo, preguntas sobre la estructura del dataset), si requiere ejecutar una consulta sobre los datos, o si está fuera del alcance del sistema.
`validator_node`	¿El resultado de la ejecución es válido y coherente con la pregunta?	01- Acepta el resultado y continúa hacia `analysis_node`. 02- Rechaza y reenvía a `query_tool_node` para corrección.	El LLM compara el resultado obtenido con la intención analítica original. Un error de ejecución o un resultado vacío activa el ciclo de corrección.
`decide_graph_need_node`	¿El tipo de resultado justifica una visualización?	01- Continúa a `visualization_node`. 02- Termina en `__end__`.	El LLM evalúa si la estructura del resultado (series, comparativas, distribuciones) se comunica mejor con un gráfico que con texto.

Cuadro 4.2: Puntos de decisión del grafo y criterios de routing

Los nodos especializados, el estado tipado y los tres puntos de decisión descritos producen un sistema que combina dos atributos que en un diseño menos cuidadoso resultan difíciles de compatibilizar: flexibilidad en la respuesta y verificabilidad del proceso. La flexibilidad proviene de las bifurcaciones agénticas, que permiten al sistema adaptarse a la naturaleza de cada pregunta. La verificabilidad proviene de

que cada transición deja traza en el `AgentState`, de modo que el recorrido completo de cualquier ejecución puede reconstruirse. La sección siguiente recorre ese flujo del sistema en una ejecución real, para entender como se conecta cada componente.

4.2. Flujo de ejecución end-to-end

Esta sección recorre el flujo de ejecución completo del sistema, desde que el usuario formula una pregunta hasta que recibe la respuesta. El flujo no es lineal ni fijo: en función de la naturaleza de la entrada, el sistema determina qué nodos se activan en cada ejecución. Una pregunta simple puede resolverse en un único nodo, mientras que una pregunta compleja puede recorrer los siete nodos e incluir hasta dos reintentos del ciclo de autocorrección. A continuación se describen los tres caminos de ejecución posibles, según el caso de uso.

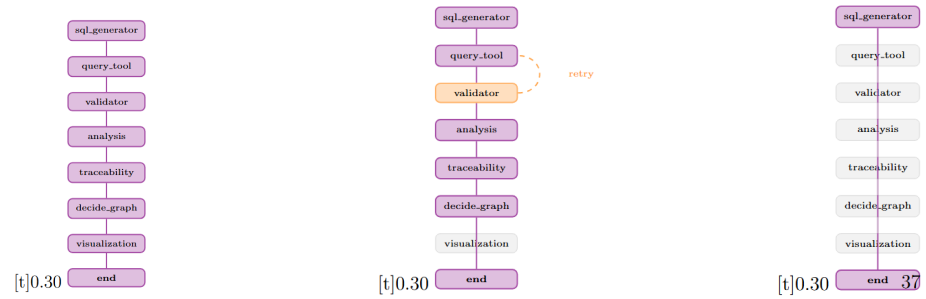
Antes de recorrer los caminos posibles de ejecución, conviene precisar qué recibe el sistema y qué produce. El Cuadro 4.3 recoge los campos del input y del output global.

Dirección	Campo	Descripción
Input	`messages`	Lista de mensajes de la conversación. Contiene el mensaje del usuario como `HumanMessage`.
Input	`question`	Texto de la pregunta formulada por el usuario, en lenguaje natural.
Output	`answer`	Respuesta en lenguaje natural generada por `analysis_node`.
Output	`sql_executed`	Consulta SQL generada y ejecutada sobre DuckDB.
Output	`traceability`	Explicación del procedimiento seguido para obtener el resultado.
Output	`chart_spec`	Especificación Vega-Lite del gráfico. Solo se popula si `decide_graph_need_node` activa `visualization_node`.
Output	`validation_retries`	Número de reintentos realizados por el ciclo de autocorrección.
Output	`error`	Mensaje de error en caso de fallo no recuperable.

Cuadro 4.3: Campos de entrada y salida del sistema

Caminos de ejecución

El grafo admite múltiples caminos de ejecución posibles, resultado de la combinación de las tres decisiones de routing descritas en la sección anterior. A continuación se presentan tres casos representativos que cubren los escenarios de uso más habituales; otros caminos intermedios son posibles según la combinación concreta de decisiones que tome el sistema en cada ejecución.



Caso A. Ejecución completa con visualización

Caso B. Ejecución con reintento y sin gráfico

Caso C. Respuesta directa sin consulta SQL

Figura 4.2: Tres caminos de ejecución representativos según la naturaleza de la pregunta. Los nodos en violeta se activan; los nodos en gris no participan en esa ejecución. La flecha naranja discontinua indica el ciclo de reintento. La combinación de las tres decisiones de routing genera otros caminos posibles no ilustrados aquí.

Caso A: ejecución analítica completa. El usuario formula una pregunta que requiere consultar el dataset. `sql_generator_node` genera el SQL, `query_tool_node` lo ejecuta sobre DuckDB y `validator_node` confirma que el resultado es coherente con la intención analítica. `analysis_node` transforma el resultado en lenguaje natural, `traceability_node` genera la explicación del cálculo y `decide_graph_need_node` determina que la estructura del resultado justifica una visualización. Los siete nodos se activan en secuencia.

Caso B: ejecución con autocorrección. El flujo recorre los mismos nodos iniciales, pero `validator_node` detecta que el resultado no satisface la pregunta original y reenvía la ejecución a `query_tool_node`. El ciclo se repite hasta que el resultado supera la validación o se alcanza el límite configurable de reintentos (`max_query_retries=2`). En este caso, `decide_graph_need_node` determina que el resultado no requiere visualización y el flujo termina sin activar `visualization_node`.

Caso C: respuesta sin consulta SQL. `sql_generator_node` evalúa que la pregunta puede responderse con el conocimiento del catálogo semántico, como una consulta sobre qué información contiene el dataset, o que está fuera del alcance del sistema. En ambos subcasos, el nodo produce una respuesta directa y el flujo termina sin activar ningún nodo posterior.

El objeto `Output` mencionado anteriormente en el Cuadro 4.3 está diseñado para cubrir el conjunto de caminos de ejecución posibles. Por ejemplo, los campos `chart_spec` y `validation_retries` son opcionales y se populan únicamente cuando los nodos correspondientes se activan. De esta manera, la interfaz de usuario recibe

siempre un objeto con forma conocida, con independencia del camino seguido.

En la siguiente sección, se profundizará sobre el estado interno que transporta esta información entre nodos a lo largo del grafo.

4.3. Estado del agente y contratos de datos

Los nodos del grafo no se comunican entre sí de forma directa. Toda la información que circula a lo largo de una ejecución reside en un objeto compartido denominado `AgentState`, un `TypedDict` tipado que actúa como memoria de turno del sistema. Cada nodo lee los campos que necesita para ejecutar su lógica y escribe únicamente los campos que le corresponden por responsabilidad funcional, sin acceso ni conocimiento del estado interno de los demás nodos. Esta separación por contrato hace que el flujo sea predecible, puesto que el comportamiento de cualquier nodo queda completamente determinado por los campos del estado que recibe, con independencia de cómo se haya llegado a ese punto del grafo.

El `AgentState` se define de manera que cada nodo popula los campos que le corresponden y deja el resto intacto. Así, el estado se construye de forma acumulativa a medida que avanza la ejecución.

La definición completa del estado es la siguiente:

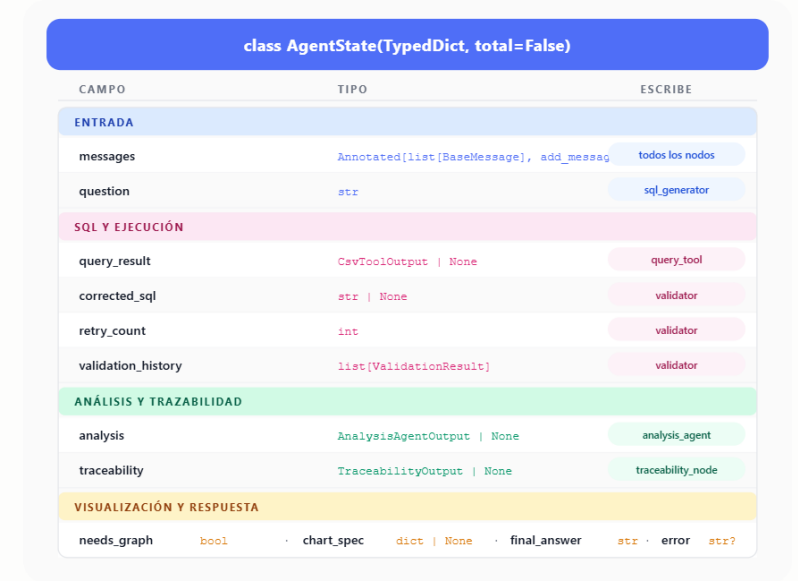


Figura 4.3: Estado del Sistema creado

Cada nodo del sistema cuenta con un modelo Pydantic asociado que actúa como contrato estructurado que define exactamente qué forma debe tener el valor que ese nodo escribe en el estado compartido. Esta decisión desplaza la validación al momento de la escritura, de modo que un valor malformado no puede propagarse a nodos posteriores sin ser detectado.

Además, para garantizar que cada LLM responde conforme al modelo especificado, se utiliza el patrón `with_structured_output`. Este patrón fuerza al LLM a producir directamente un objeto tipado con los campos esperados, en lugar de generar texto libre que deba parsearse a posteriori. Si el LLM no puede producir un output que satisfaga el contrato, el sistema detecta el fallo en ese nodo y activa los mecanismos de recuperación definidos en el grafo.

El Cuadro 4.4 recoge los siete modelos y su correspondencia con los campos del estado compartido.

Nodo	Modelo Pydantic	**Campos del estado que popula**
`sql_generator_node`	`SqlGeneratorOutput`	`messages`, `final_answer`
`query_tool_node`	`CsvToolOutput`	`query_result`
`validator_node`	`ValidationResult`	`retry_count`, `validation_history`, `corrected_sql`
`analysis_node`	`AnalysisAgentOutput`	`analysis`, `final_answer`
`traceability_node`	`TraceabilityOutput`	`traceability`
`decide_graph_need_node`	`GraphDecisionOutput`	`needs_graph`, `graph_type`
`visualization_node`	`VisualizationOutput`	`chart_spec`

Cuadro 4.4: Modelos Pydantic por nodo y campos del `AgentState` que cada uno popula

4.4. Capa semántica y representación del dominio

El LLM que genera SQL en el sistema nunca accede directamente al dataset. Sin una representación explícita del dominio, el modelo tiende a producir consultas con columnas inexistentes, nombres de entidades financieras inventados o valores

categóricos que no corresponden a los datos reales. El catálogo semántico resuelve ese problema: es un objeto de metadata estática, definido por el desarrollador, que describe qué columnas existen, qué significan y qué valores pueden tomar, y que se inyecta directamente en el prompt de los nodos que requieren comprensión del dominio. El resultado es que el LLM genera SQL anclado en la estructura real del dataset, con filtros y agregaciones coherentes con las definiciones de negocio.

El catálogo creado se organiza en tres niveles jerárquicos, recogidos en el Cuadro 4.5.

Nivel	`Modelo`	**Campos**
Raíz	`DataCatalog`	`datasets: List[Dataset]`
Dataset	`Dataset`	`name`, `description`, `columns: List[Column]`
Columna	`Column`	`name`, `dtype`, `description`, `examples` (opcional)

Cuadro 4.5: Estructura jerárquica del catálogo semántico

El campo `examples` merece atención específica. Para columnas categóricas, como el tipo de producto financiero o el segmento de edad del usuario, proporcionar valores representativos en el catálogo es lo que permite al LLM construir filtros y condiciones con los literales correctos. Sin ese anclaje, el modelo genera valores plausibles pero incorrectos que producen consultas sin resultados o, peor, con resultados silenciosamente erróneos.

La estrategia de inyección del catálogo en el prompt fue objeto de evaluación explícita durante el desarrollo. Con 26 columnas y aproximadamente 8.200 caracteres, el catálogo completo cabe holgadamente en la ventana de contexto del modelo. Desde las primeras iteraciones se optó por inyección directa, reservando la migración a un enfoque de Retrieval-Augmented Generation (RAG, o generación aumentada por recuperación) para el caso de que el sistema mostrara degradación por saturación de contexto. A medida que avanzó la implementación quedó claro que la inyección directa era suficiente: el modelo interpretaba el catálogo con precisión y los resultados del benchmarking no evidenciaron errores atribuibles a saturación. Añadir RAG habría introducido latencia adicional y un coste de implementación desproporcionado respecto al beneficio esperado en ese punto. Dicho esto, RAG permanece identificado como línea de evolución para iteraciones futuras si el volumen del catálogo crece de forma significativa.

4.5. Generación de SQL y estrategia de prompting

El `sql_generator_node` es el nodo donde el sistema transforma una pregunta en lenguaje natural en una operación sobre datos. El modelo utiliza el mecanismo de *tool calling* (llamada a herramienta), que fuerza al LLM a estructurar sus argumentos en un esquema JSON validado por la API antes de que lleguen al motor de ejecución. Extraer SQL de texto libre mediante parseo habría introducido una fuente de error difícil de controlar; con tool calling, cualquier discrepancia entre el output del modelo y el esquema esperado se detecta en el momento de la invocación. El mecanismo requiere un modelo con soporte nativo de function calling, condición que GPT-4o cumple en la infraestructura Azure disponible.

El mismo mecanismo resuelve el caso en que la pregunta del usuario no requiere datos. Si el LLM determina que puede responder con el conocimiento del catálogo semántico, produce una respuesta directa sin invocar la herramienta. La función de routing `should_continue` detecta la ausencia de tool calls en el estado y dirige la ejecución a `__end__`, evitando que preguntas simples recorran el pipeline completo de validación y análisis.

El conocimiento que guía esa decisión, junto con todas las instrucciones de generación SQL, llega al modelo a través del prompt. Su estructura se organiza en dos capas: un system message con el contexto de dominio, las reglas de generación y el catálogo semántico completo, y un human message con la pregunta del usuario.

Los prompts del sistema se han tratado como artefactos de ingeniería propios, con la misma disciplina aplicada al resto del código. Cada prompt reside en un módulo dedicado, se versiona junto al sistema y se itera en función de los resultados del pipeline de evaluación. Cuando el benchmarking identifica un patrón de error recurrente en algún nodo, la respuesta habitual es una modificación del prompt correspondiente, añadiendo una instrucción que reduzca la frecuencia de ese comportamiento. Este ciclo de evaluación y refinamiento ha sido el principal mecanismo de mejora del sistema a lo largo del desarrollo.

La redacción de los prompts sigue criterios de prompt engineering contrastados en la literatura y aplicados al contexto concreto del sistema.

- **Instrucciones positivas y específicas.** Las instrucciones describen el comportamiento esperado de forma afirmativa y con ejemplos concretos del dominio, en lugar de enumerar lo que el modelo debe evitar. Esto reduce la ambigüedad interpretativa y mejora la consistencia entre ejecuciones.

- **Separación de contexto y tarea.** El system message contiene el conocimiento estático del dominio (catálogo, reglas de negocio, convenciones del motor

SQL), mientras que el human message se limita a la pregunta del turno actual. Esta separación facilita la actualización del catálogo sin modificar la lógica de la tarea.

- **Compatibilidad con `with_structured_output`.** Los prompts de los nodos que producen outputs tipados se redactan de forma que el modelo recibe instrucciones alineadas con el esquema Pydantic esperado, lo que reduce la tasa de fallos de conformidad cuando se fuerza la salida estructurada.

El prompt se escribe en español porque el dataset, las columnas y el dominio son españoles. Mantener el mismo idioma en las instrucciones y en los nombres de columna reduce la ambigüedad léxica que aparece cuando el modelo debe alinear términos en dos lenguas distintas.

El conjunto de estas decisiones sobre prompting son importantes porque determinan directamente la calidad del SQL generado. La sección siguiente describe qué ocurre con ese SQL una vez que llega al motor de ejecución.

4.6. Ejecución de consultas con DuckDB

Que el sistema acepte SQL generado por un LLM sobre datos financieros sensibles exige garantías que no dependan del comportamiento del modelo. Por bien instruido que esté, el LLM puede producir queries que extraigan volúmenes masivos de datos o, en casos extremos, intenten operaciones de escritura sobre el sistema de ficheros. La herramienta de ejecución resuelve esto de forma estructural, con controles que actúan antes de que cualquier query llegue al motor.

Antes de llegar al motor, toda query pasa por dos capas de validación recogidas en el Cuadro 4.6.

#	Mecanismo	Qué bloquea	Cuándo actúa
1	`_validate_read_only_sql`	`INSERT`, `UPDATE`, `DELETE`, `DROP`, `ALTER`, `CREATE`, `COPY`, `ATTACH`, `DETACH`, `TRUNCATE`, `CALL`	Antes de ejecutar
2	`Verificación de prefijo`	Cualquier SQL que no comience por `SELECT` o `WITH`	Antes de ejecutar
3	`_enforce_limit`	Resultados sin límite o con `LIMIT` arbitrario generado por el LLM	Antes de ejecutar
4	`duckdb.connect()` + `finally: close()`	Estado residual entre queries	En toda ejecución

Cuadro 4.6: Capas de seguridad en la ejecución de consultas SQL

El límite de filas se fija en 52, valor que responde a la granularidad máxima del dominio geográfico del dataset: 50 provincias españolas más dos ciudades autónomas. Cualquier agregación territorial válida cabe dentro de ese límite. El valor es configurable por entorno sin modificar el código, lo que permite ajustarlo si el dominio de análisis cambia en iteraciones futuras.

Además, cabe destacar que cuando la ejecución falla, el error se captura y se propaga como campo `error` del objeto `CsvToolOutput`, que llega al `validator_node` con la información necesaria para activar el ciclo de autocorrección. El flujo nunca se interrumpe por una excepción no controlada: el sistema siempre produce un objeto de salida con forma conocida, independientemente de si la query fue válida. La sección siguiente describe cómo el `validator_node` procesa ese resultado y decide si el ciclo de ejecución debe repetirse.

4.7. Validación y autocorrección

Una query generada por un LLM puede fallar de formas muy distintas. Un error de sintaxis es detectable sin coste adicional; una query que devuelve cero filas porque el filtro aplica una condición semánticamente incorrecta requiere un nivel de juicio

diferente. El `validator_node` responde a esa heterogeneidad con un mecanismo de dos fases que actúan en secuencia sobre el resultado de cada ejecución.

La primera fase es determinista. Si `query_tool_node` devuelve un error de ejecución o un resultado vacío, el nodo marca la query para reintento de forma inmediata, sin invocar al LLM. Solo cuando el resultado pasa ese filtro inicial se activa la segunda fase, donde el modelo evalúa la coherencia semántica entre la pregunta original hecha por el usuario y los datos obtenidos, detectando situaciones como granularidad incorrecta o agregaciones incoherentes con la intención analítica. Los hallazgos de ambas fases se fusionan en un único objeto `ValidationResult`, cuya severidad determina el camino que sigue el flujo. El Cuadro 4.7 recoge las tres severidades posibles y su efecto sobre la ejecución.

`Severidad`	**Significado**	**Acción**	**Ejemplo**
`pass`	Resultado correcto y coherente	Continúa a `analysis_node`	Query ejecutada sin incidencias
`warning`	Resultado aceptable con reservas	Continúa con aviso al usuario	Filtro parcialmente aplicado
`retry`	Resultado incorrecto	Reintento con SQL corregida	Error de ejecución o discrepancia entre la pregunta de usuario y lo que se ha calculado

Cuadro 4.7: Severidades del `validator_node` y su efecto sobre el flujo de ejecución

Cuando la severidad es `retry`, el validador genera una `corrected_sql` y la escribe en el estado compartido. `query_tool_node` la consume directamente en el siguiente ciclo, sin volver a pasar por `sql_generator_node`. Para evitar que el LLM proponga la misma corrección fallida en intentos sucesivos, el historial acumulativo de validaciones anteriores se inyecta en el prompt de cada reintento, de modo que el modelo tiene visibilidad sobre qué se ha intentado y por qué ha fallado. El límite de reintentos se fija en dos, valor que equilibra la posibilidad de corrección con el coste de latencia: cada ciclo adicional introduce entre tres y cinco segundos de espera. La Figura 4.4 ilustra el flujo completo del mecanismo.

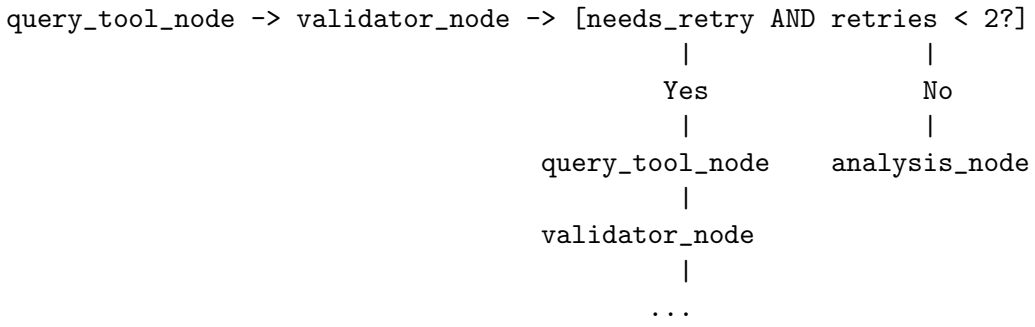


Figura 4.4: Ciclo de validación y autocorrección

Si el propio validador falla por un error inesperado en la llamada al LLM, el sistema registra la incidencia y trata el resultado como válido, permitiendo que el flujo continúe hacia `analysis_node`. La respuesta resultante puede ser imperfecta, pero el usuario recibe siempre una salida con forma conocida. Esta degradación controlada es una decisión de diseño deliberada. ya que en un sistema de uso profesional, una respuesta conservadora supera en utilidad a un error fatal que interrumpa la sesión. Una vez superada la validación, el resultado pasa a `analysis_node`, donde los datos obtenidos se transforman en una respuesta interpretativa en lenguaje natural.

4.8. Análisis y generación de respuesta

Los datos que llegan a `analysis_node` han superado el ciclo de validación y son cuantitativamente correctos. Sin embargo, un resultado tabular con filas y columnas no es directamente utilizable por un consultor que necesita incorporarlo a una presentación o tomar una decisión en minutos. Este nodo cierra esa distancia: recibe el resultado validado y produce la respuesta en lenguaje natural que el usuario finalmente lee.

La decisión de implementarlo como nodo independiente del `sql_generator_node` responde a una consideración de mantenibilidad. Ambas tareas, generar SQL correcto e interpretar sus resultados con criterio analítico, evolucionan con lógicas distintas. Iterar el prompt de análisis para mejorar la calidad interpretativa de las respuestas no introduce ningún riesgo sobre la generación de queries, y viceversa. El tradeoff es una llamada LLM adicional por turno, coste que se considera aceptable dado que el prompt de este nodo opera sobre datos ya estructurados y es considerablemente más simple que el de generación SQL.

El output del nodo se produce mediante `with_structured_output`, lo que garantiza que el campo `analysis` del estado siempre está presente con forma conocida,

con independencia de la complejidad del resultado recibido. Este campo constituye la respuesta principal que el usuario recibe en la interfaz.

La sección siguiente describe cómo esa respuesta se complementa con la explicación del procedimiento seguido para obtenerla, dotando al sistema de la trazabilidad que hace los resultados auditables.

4.9. Trazabilidad y explicación del cálculo

Cuando un sistema genera una respuesta numérica a partir de una pregunta formulada en lenguaje natural, la primera reacción de cualquier usuario con criterio profesional es preguntarse si el sistema interpretó correctamente lo que se le pedía. En un entorno de consultoría donde los análisis se incorporan a propuestas comerciales presentadas a clientes, esa pregunta tiene consecuencias concretas: una cifra obtenida con un filtro incorrecto o bajo un supuesto no comunicado puede inducir decisiones erróneas. `traceability_node` existe precisamente para hacer esa verificación posible, generando una explicación estructurada de qué entendió el sistema, qué operaciones realizó y qué supuestos tomó cuando la pregunta era ambigua.

El nodo produce un objeto `TraceabilityOutput` con nueve campos tipados que cubren distintas dimensiones de la explicación. El Cuadro 4.8 detalla cada campo y su propósito.

Campo	Tipo	Propósito
`interpretation`	str	Descripción en lenguaje natural de cómo el sistema interpretó la pregunta del usuario.
`steps`	list[str]	Secuencia de pasos del cálculo expresados en lenguaje llano, sin sintaxis técnica.
`filters_applied`	list[str]	Condiciones de filtrado que se aplicaron sobre el dataset para obtener el resultado.
`filters_not_applied`	str — None	Dimensiones disponibles en el dataset que no se filtraron, relevantes para contextualizar el alcance del resultado.
`assumptions`	list[str] — None	Supuestos adoptados cuando la pregunta presentaba ambigüedad semántica.
`data_source`	str	Descripción del origen de los datos utilizados en el cálculo.
`grouping`	str — None	Dimensión o conjunto de dimensiones por las que se agrupó el resultado.
`aggregations`	str — None	Función o funciones de agregación aplicadas sobre el resultado.
`limit_applied`	bool	Indica si el resultado fue truncado por el techo máximo de filas del sistema.

Cuadro 4.8: Campos del objeto `TraceabilityOutput` y su propósito

El output estructurado con campos tipados tiene una implicación directa sobre la interfaz. Cada sección de la explicación, la interpretación, los pasos, los supuestos y los filtros, puede renderizarse de forma diferenciada sin lógica de parseo adicional. Un texto libre habría transferido esa complejidad a la capa de presentación. Para perfiles con formación técnica, la sentencia SQL ejecutada se expone en un desplegable independiente dentro de la interfaz, de modo que la verificación puede realizarse tanto en lenguaje natural como inspeccionando directamente la operación sobre los datos.

Una decisión tomada fue la de que el nodo operase de forma non-blocking. Si

la generación de la trazabilidad falla por cualquier motivo, el sistema registra la incidencia y continúa con `traceability` establecido a `None` en el estado. De esta manera, la trazabilidad añade valor cuando está disponible, pero su ausencia ocasional no compromete la utilidad del sistema. Con la respuesta generada y su explicación disponible, el flujo pasa a `decide_graph_need_node`, que evalúa si la estructura del resultado justifica complementarlo con una representación visual.

4.10. Generación de visualizaciones

Pedir al LLM que genere código de visualización directamente produce errores con una frecuencia que resulta inaceptable en producción. Un JSON Vega-Lite malformado llega al frontend como un fallo visible, y en un contexto donde los análisis se incorporan a presentaciones profesionales, ese tipo de incidencia compromete la credibilidad del sistema completo. La arquitectura implementada parte de esta observación para separar dos responsabilidades que tienen naturalezas distintas.

El flujo de generación se divide en dos nodos secuenciales. `decide_graph_need_node` evalúa si el resultado justifica una representación visual: un valor escalar único no requiere gráfico; una distribución por categorías sí. Cuando la decisión es afirmativa, `visualization_node` determina el tipo de gráfico y los campos a representar, produciendo un objeto `VisualizationOutput` tipado. La función `build_chart_spec_from_intent` toma ese objeto y construye el JSON Vega-Lite mediante lógica completamente determinista. La Figura 4.5 ilustra el flujo.

```
[LLM] --> VisualizationOutput (intent semántico)
                    |
     [build_chart_spec_from_intent] (determinista)
                    |
          chart_spec (Vega-Lite JSON)
                    |
         [st.vega_lite_chart] (renderizado)
```

Figura 4.5: Separación entre intent semántico y rendering determinista en la generación de visualizaciones

El modelo actúa como clasificador semántico. Dado el resultado y la pregunta, determina si el gráfico apropiado es de barras, líneas, área o dispersión, e identifica qué columnas mapear a cada eje. El builder recibe ese intent y construye el JSON con la sintaxis correcta, aplicando además la configuración visual corporativa de forma independiente al modelo. La compilación es siempre válida. El único vector de error

restante es un intent semánticamente incoherente, situación que el `model_validator` de Pydantic intercepta antes de que llegue al builder, aplicando correcciones heurísticas sobre situaciones frecuentes detectadas en los primeros benchmarks del sistema, como la conversión de tipo nominal a ordinal en gráficos de tendencia o la eliminación del campo de color cuando duplica la dimensión del eje X.

La configuración visual, colores, tipografía y estilo corporativo, se define como parámetro de configuración y el builder la aplica sistemáticamente, con independencia de lo que el modelo haya clasificado. Esto garantiza consistencia visual a lo largo de todas las ejecuciones sin añadir esa responsabilidad al prompt del nodo.

Con la visualización generada, el sistema dispone de todos los componentes del output: respuesta analítica, trazabilidad del cálculo y representación gráfica opcional. La sección siguiente describe cómo estos elementos se integran en la interfaz de usuario.

4.11. Interfaz de usuario e integración con el dashboard

El chatbot se desarrolló inicialmente como herramienta autónoma. Con el tiempo, desde el área de negocio se identificó la oportunidad de integrarlo en un panel analítico preexistente que presentaba análisis predefinidos sobre el mismo conjunto de datos financieros, permitiendo al usuario alternar entre consulta exploratoria y análisis fijo dentro de un único entorno. El panel estaba construido en Streamlit, lo que eliminó la fricción técnica de integración y consolidó la elección del marco de trabajo para el chatbot. La interfaz resultante sigue la identidad visual corporativa de la firma.

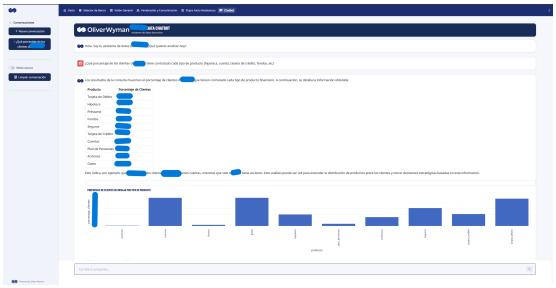
La experiencia de usuario se diseñó con decisiones técnicas concretas detrás de cada elemento. Al acceder a la pestaña del chatbot, el usuario encuentra un conjunto de preguntas predefinidas que, con un solo clic, lanzan consultas frecuentes o preguntas de exploración sobre las capacidades y el alcance del sistema. Esta funcionalidad reduce la barrera de adopción para perfiles que no tienen claro cómo empezar a interactuar con una interfaz conversacional sobre datos.

El tiempo de carga se gestionó mediante carga diferida con la función `_get_runner`: la importación de los módulos pesados del sistema se pospone hasta el primer mensaje del usuario, de modo que la página se carga de forma instantánea. En esa primera invocación, se muestra un mensaje informativo que aporta transparencia al usuario sobre por qué esa primera respuesta tarda algo más que las siguientes.

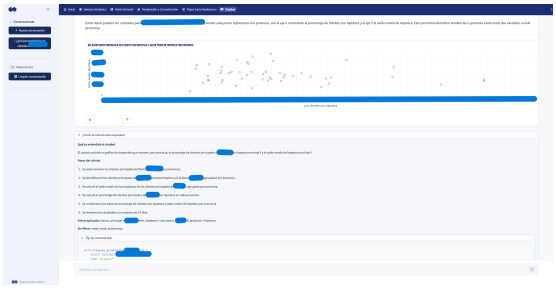
Los resultados que el sistema produce tienen un destino claro en el flujo de tra-

bajo de los equipos de consultoría: presentaciones y documentos de trabajo. Con ese uso en mente, se habilitaron botones de exportación de tablas en formato de hoja de cálculo y descarga de visualizaciones. En los gráficos generados, un botón de visualización tabular permite alternar entre la representación gráfica y la tabla numérica subyacente. Esta tabla admite filtrado por columna, ordenación ascendente y descendente, ocultación de columnas y cambio de unidad, funcionalidades que permiten al usuario adaptar el resultado a su necesidad sin reformular la consulta original.

- **Preguntas predefinidas:** acceso con un clic a consultas frecuentes y preguntas de exploración del sistema.
- **Carga diferida:** inicio instantáneo de la página con mensaje informativo en la primera consulta.
- **Exportación:** descarga de tablas en formato de hoja de cálculo y visualizaciones para uso en presentaciones.
- **Vista tabular:** alternancia entre gráfico y tabla numérica, con filtrado, ordenación y cambio de unidad.



(a) Vista de la Interfaz Gráfica - anonimizada



(b) Vista de la Interfaz Gráfica - anonimizada

Figura 4.6: Interfaz del chatbot integrado en el panel analítico.

El sistema de retroalimentación permite valorar cada respuesta mediante indicadores positivo y negativo, con la opción de añadir un comentario. El evento registra la valoración junto al contexto completo del turno: la consulta SQL ejecutada, los errores detectados y el número de reintentos realizados. Esta información convierte la retroalimentación en un mecanismo de diagnóstico accionable para iteraciones posteriores del sistema. Para evitar que el envío bloquee la interfaz, la telemetría se

despacha de forma asíncrona en un hilo secundario de tipo demonio, eliminando una latencia perceptible de entre uno y dos segundos. Los eventos se registran en Azure Application Insights, desde donde pueden consultarse para orientar las decisiones de mejora del sistema.

Con la interfaz operativa y la retroalimentación estructurada en marcha, el sistema está en condiciones de desplegarse en la infraestructura de la organización. La sección siguiente describe ese proceso y las consideraciones técnicas relevantes para su uso en producción.

4.12. Despliegue y consideraciones de producción

El sistema se despliega en Azure App Service con contenerización Docker. Dicha elección se debe a que en un entorno de consultoría con plazos ajustados, la prioridad era llegar a producción con la menor fricción operativa posible para empezar a recibir retroalimentación real de los usuarios. Azure App Service ofrece un servicio gestionado que elimina la administración de infraestructura, y un único contenedor que concentra todas las dependencias del sistema sin requerir almacenamiento externo.

Durante todo el proyecto, se realizó una separación clara entre entornos de desarrollo y producción. para gestionar variables de entorno en local, se utilizó el fichero `.env`; mientras que para el entorno de producción, se empleó *Key Vault* de Azure. El Cuadro 4.9 recoge las variables de configuración principales.

`Variable`	**Propósito**
`AOAI_ENDPOINT`, `AOAI_API_KEY`	Credenciales del servicio Azure OpenAI
`AOAI_DEPLOYMENT`	Identificador del modelo desplegado
`DATASET_PATH`	Ruta al dataset dentro del contenedor
`DEFAULT_MAX_ROWS`	Límite máximo de filas por consulta
`MAX_QUERY_RETRIES`	Número máximo de reintentos del validador
`APPINSIGHTS_CONNECTION_STRING`	Cadena de conexión para telemetría

Cuadro 4.9: Variables de entorno de configuración del sistema

Para acelerar el ciclo de iteración, se configuró un flujo de integración y despliegue continuos vinculado a la rama de producción del repositorio. Cada incorporación de cambios a esa rama ejecuta automáticamente el despliegue en Azure mediante un flujo de trabajo con los secretos de acceso configurados en GitHub. Esto elimina el despliegue manual y reduce a minutos el tiempo entre una mejora del sistema y su disponibilidad para los usuarios.

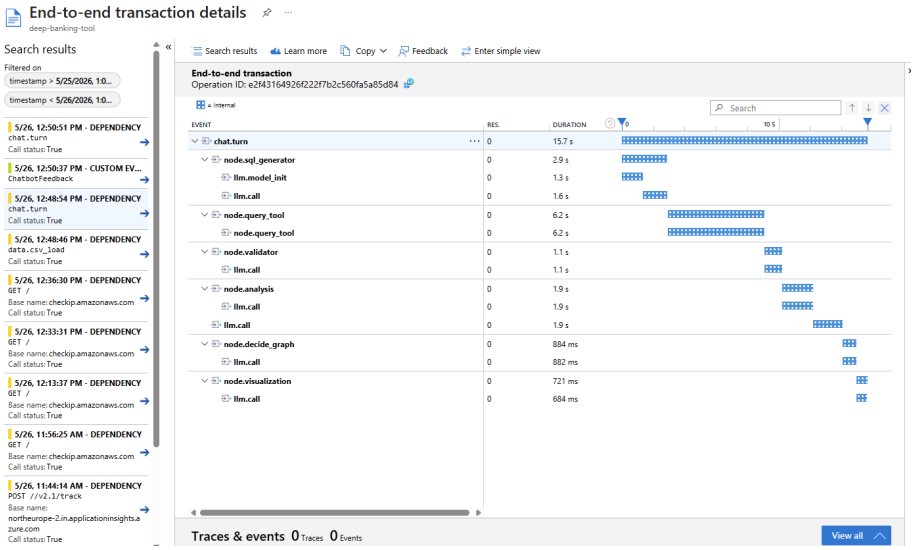


Figura 4.7: Ejemplo de observabilidad del sistema mediante Application Insights en Azure

La observabilidad en producción se instrumenta mediante Azure Application Insights, conectado a un repositorio de registros compartido de la organización. Desde Application Insights se visualizan en tiempo real todos los eventos relacionados con la interacción de usuarios con el sistema, como las preguntas realizadas, el desglose de la ejecución del flujo del sistema para cada pregunta, y el posible feedback enviado por los usuarios al recibir respestas. Esto permite construir paneles que muestran la proporción de valoraciones positivas y negativas sobre las respuestas del sistema, los comentarios recibidos y los patrones de uso a lo largo del tiempo. Esta infraestructura cierra el ciclo entre el sistema en producción y las decisiones de mejora en iteraciones sucesivas, cuyo proceso se describe en la sección siguiente.

4.13. Evolución iterativa de la implementación

La arquitectura final del sistema es el resultado de un proceso de mejora continua guiado por evidencia. Cada componente incorporado a lo largo del desarrollo respondió a un gap concreto, detectado mediante evaluaciones automatizadas o retroalimentación directa de los usuarios finales. Esta lógica determina también lo que no se implementó: ningún componente se incorporó por anticipación especulativa. La metodología de evaluación que permitió detectar estos gaps se describe en el Capítulo 5; esta sección recoge únicamente cómo el proceso iterativo guió las decisiones de construcción.

La Figura 4.8 presenta las siete iteraciones del desarrollo como una cadena de sprints, donde el gap detectado en cada etapa motivó directamente la siguiente.

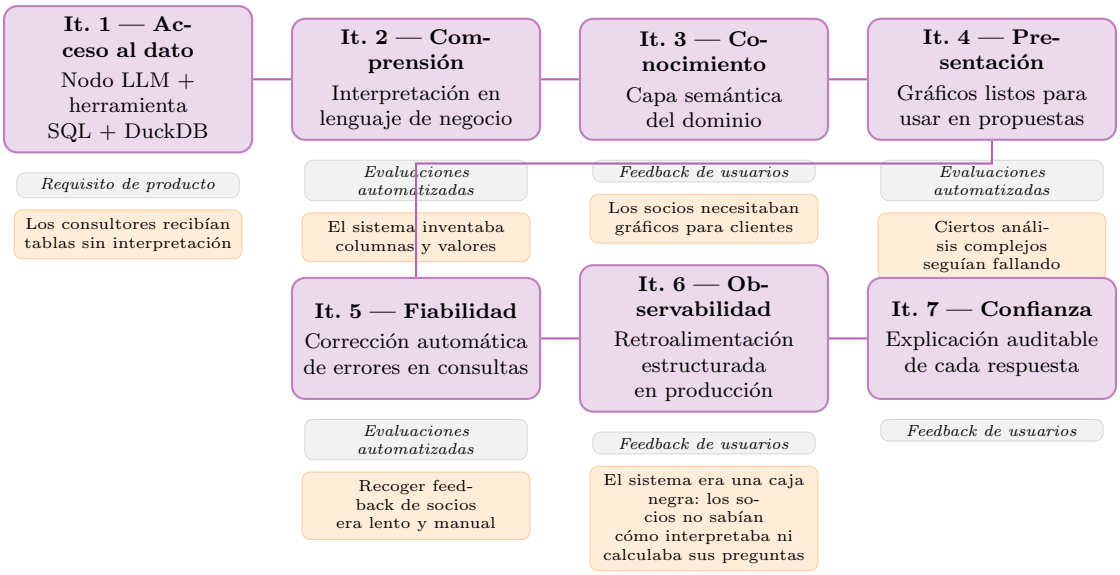


Figura 4.8: Evolución iterativa del sistema. Cada iteración muestra el componente implementado, la fuente de detección (en gris) y el problema que motivó la siguiente iteración (en naranja). Las flechas indican la secuencia de desarrollo.

Las siete iteraciones siguen tres patrones de evolución que reflejan cómo fue madurando el sistema. Las tres primeras consolidaron el núcleo funcional. El MVP demostró la viabilidad técnica del acceso conversacional a datos, pero los resultados eran tablas crudas que los usuarios de negocio no les era sencillo interpretar directamente. El nodo de análisis resolvió esa brecha transformando los datos en respuestas en lenguaje natural.

El catálogo semántico fue la mejora con mayor impacto unitario. Antes de su incorporación, el modelo generaba consultas con nombres de columna incorrectos o afirmaba que ciertos datos no existían en el dataset cuando sí estaban disponibles, al carecer de una descripción explícita del esquema. Proporcionarle esa descripción, con nombres exactos de columna, tipos y valores representativos de campos categóricos, eliminó esa categoría de error por completo.

#	**Fase**	**Iteraciones**	**Evolución del sistema**
1	Consolidar el núcleo funcional	MVP conversacional, interpretación en lenguaje natural, catálogo semántico	El sistema pasa de ejecutar consultas a comprender el dominio financiero.
2	Ampliar capacidades	Visualización automática, validación con autocorrección	El sistema pasa de responder correctamente a responder con fiabilidad y producir resultados directamente utilizables.
3	Cerrar el ciclo de confianza	Retroalimentación estructurada, trazabilidad del cálculo	El sistema pasa de ser funcionalmente correcto a ser adoptable por perfiles de decisión senior.

Cuadro 4.10: Agrupación de las siete iteraciones por patrón de evolución

Las iteraciones cuarta y quinta ampliaron las capacidades del sistema hacia la utilidad real. La visualización respondió a una necesidad concreta de los socios: incorporar gráficos a propuestas sin trabajo manual adicional. La validación con autocorrección respondió a un problema diferente: múltiples rondas de mejora del prompt habían llegado a un plateau sin resolver ciertos patrones de error en preguntas complejas. La respuesta fue cambiar de estrategia: en lugar de seguir enriqueciendo el contexto del modelo, se implementó un mecanismo de corrección en tiempo de ejecución. Las evaluaciones posteriores confirmaron que esta iteración rompió el plateau observado.

Las dos últimas iteraciones cerraron el ciclo de confianza. El sistema técnicamente correcto seguía teniendo una barrera de adopción en perfiles senior que necesitaban poder verificar que el chatbot había interpretado bien su pregunta. La trazabilidad

resolvió esa barrera con una explicación estructurada del razonamiento seguido. El mecanismo de retroalimentación por parte de los usuarios finales cerró el ciclo de mejora continua en producción, ya que son los usuarios reales quienes revelan patrones de uso, formas de preguntar y necesidades concretas que ningún banco de pruebas automatizado anticipa completamente.

Las siete iteraciones comparten un principio de diseño común. Cada componente se implementó con el alcance mínimo necesario para resolver el problema que lo motivaba, y se descartó cualquier mejora cuya necesidad no estuviese respaldada por evidencia. Estas decisiones de sobriedad se llevaron a cabo como elecciones deliberadas que mantienen el sistema predecible y evolutivo.

El siguiente Capítulo 5 presenta la evaluación cuantitativa del impacto de estas iteraciones sobre el rendimiento del sistema.

Capítulo 5

Evaluación y Resultados

5.1. Metodología de evaluación

Desarrollar un chatbot analítico sobre datos financieros reales impone una exigencia que no se satisface con pruebas manuales ocasionales: cada modificación del sistema, ya sea en el prompt de generación de SQL, en las reglas del catálogo semántico o en la lógica de validación, puede mejorar el comportamiento en un subconjunto de preguntas mientras lo degrada silenciosamente en otro. Sin un mecanismo de medida reproducible, el desarrollo avanza sin saber si los cambios introducidos tienen el efecto esperado. Por esta razón, antes de abordar la optimización del sistema en sí, se diseñó un *pipeline* (cadena de procesamiento automatizada) de evaluación que pudiera ejecutarse de forma continua a lo largo del desarrollo.

El diseño de este pipeline de evaluación respondió a tres requisitos concretos del contexto del proyecto:

- **Evaluación continua sin fricción:** con más de 25 iteraciones en seis semanas de desarrollo, el proceso completo debía poder lanzarse en segundo plano en menos de diez minutos para un conjunto de preguntas cerradas.
- **Output interpretable por perfiles no técnicos:** las reuniones semanales con socios y dirección requerían una lectura ejecutiva centrada en el *score* global del sistema, las categorías de error predominantes y las recomendaciones para el siguiente sprint.
- **Modularidad:** el conjunto de preguntas, los criterios de evaluación y el formato del informe debían poder modificarse de forma independiente a medida que el alcance evolucionaba, para adaptarse a necesidades cambiantes.

Teniendo estos requisitos en cuenta, el pipeline resultante se organizó en tres fases secuenciales representadas en la Figura 5.1.

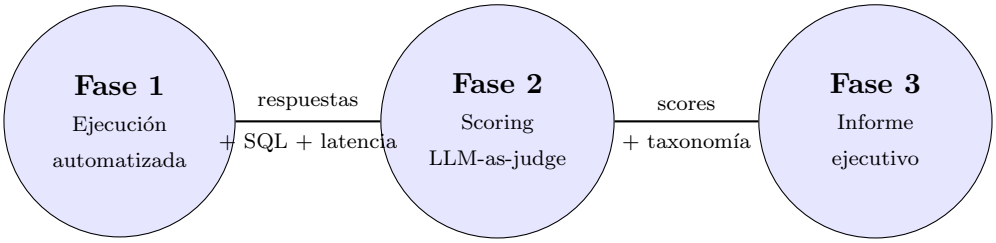


Figura 5.1: Pipeline de evaluación automatizada: tres fases secuenciales.

El pipeline descrito opera siempre sobre el sistema completo, desde la pregunta en lenguaje natural hasta la respuesta final generada por el nodo de análisis, recorriendo los mismos nodos que atraviesa una consulta real de usuario. Esta decisión implica que los resultados del pipeline reflejan el comportamiento observable en producción, con todos los efectos de integración entre componentes incluidos, o cual resulta un comportamiento deseable.

Cuadro 5.1: Fases del pipeline de evaluación

Fase	**Proceso**	**Salida**
1. Ejecución	Invocación end-to-end del chatbot para cada pregunta del golden dataset	Respuestas generadas, SQL ejecutada, latencia, metadatos de trazabilidad
2. Scoring	LLM-as-judge + verificación heurística numérica	Score por pregunta (`pass / partial / fail / not_evaluable`) + categoría de error según taxonomía cerrada
3. Informe	Agregación de métricas y generación de report Markdown	Informe ejecutivo con resumen, errores principales, accuracy por complejidad y recomendaciones; clasificación semafórica (verde / ámbar / rojo)

Las preguntas del conjunto de evaluación se clasifican en cuatro niveles de complejidad que reflejan la dificultad analítica de la consulta subyacente. El nivel C1

corresponde a preguntas con un único eje de análisis y sin cálculos intermedios, como la tasa de penetración de una entidad financiera sobre su universo de clientes. El nivel C2 introduce uno o dos ejes de análisis con un cálculo agregado, como la distribución de volumen por tipo de producto. El nivel C3 añade comparativas entre dos entidades o combinaciones de dos ejes con un cálculo derivado. El nivel C4 agrupa las preguntas de máxima complejidad analítica: tres o más ejes, cálculos estadísticos sobre segmentos cruzados como percentiles de saldo filtrados por perfil sociodemográfico, o composiciones de métricas que requieren múltiples CTEs (expresiones de tabla comunes) en la consulta generada. Esta clasificación se llevó a cabo porque permite identificar en qué punto de la escala de complejidad se produce la mayor degradación del sistema y priorizar las mejoras donde el impacto es más relevante para los casos de uso reales.

Cuadro 5.2: Niveles de complejidad del conjunto de evaluación

Nivel	**Descripción**	**Ejemplo genérico**
C1	Un eje de análisis, sin cálculos intermedios	Tasa de penetración de clientes de una entidad
C2	Uno o dos ejes con un cálculo agregado	Distribución de volumen por tipo de producto
C3	Comparativa entre entidades o dos ejes con cálculo derivado	Volumen dentro y fuera de cartera en dos entidades
C4	Tres o más ejes, cálculos estadísticos complejos, múltiples CTEs	Percentiles de saldo de un segmento de clientes con filtros cruzados

La sección siguiente describe el conjunto de preguntas y respuestas esperadas sobre el que se ejecuta este pipeline, también referido como "golden dataset", y explica cómo se construyó para garantizar que las métricas obtenidas sean representativas de los casos de uso reales del sistema.

5.2. Conjunto de datos de evaluación

La calidad de cualquier sistema de evaluación depende de la representatividad de las preguntas que lo componen. Una alternativa habitual en la literatura es generar preguntas sintéticas a partir de la estructura del esquema de datos [7, 33]; este enfoque produce conjuntos grandes y variados, pero introduce un sesgo estructural:

las preguntas se diseñan para cubrir construcciones SQL, no para reflejar lo que los usuarios reales necesitan saber. Para un sistema cuyo objetivo es responder preguntas de negocio concretas, esa desconexión presenta una limitación práctica relevante. Por este motivo, el conjunto de evaluación del presente trabajo se construyó íntegramente a partir de casos de uso reales documentados.

Las preguntas proceden de tres fuentes verificables:

- **Propuestas a cliente previas:** casos en los que ya se habían calculado métricas financieras agregadas empleando el mismo dataset financiero, para propuestas de proyectos a clientes financieros de la firma. Los insights numéricos documentados en esas propuestas se convirtieron en preguntas evaluables con respuesta esperada conocida.
- **Cálculos validados manualmente:** métricas calculadas por perfiles técnicos mediante herramientas convencionales, que actuaron como ground truth independiente del sistema evaluado.
- **Dashboard analítico:** fuente principal del conjunto. Desarrollado en paralelo al chatbot sobre el mismo dataset subyacente, sus 38 insights se incorporaron directamente al conjunto de evaluación una vez validadas sus métricas.

Cada entrada del conjunto se almacena como un registro estructurado en formato JSON con los campos recogidos en la Tabla 5.3. El campo de fórmula explícita en la respuesta esperada merece atención particular: en un dominio financiero con múltiples definiciones posibles para una misma métrica (por ejemplo, penetración calculada sobre el universo total de clientes o solo sobre los clientes activos), la respuesta esperada no puede limitarse a un valor numérico. Documentar la fórmula que produce ese valor es lo que permite distinguir un error de cálculo de una interpretación alternativa igualmente válida.

Aunque a lo largo del desarrollo del sistema se construyeron múltiples versiones de un "golden dataset", el conjunto final comprende 38 insights que cubren múltiples familias de métricas financieras, como por ejemplo: ratios de penetración y cuota de mercado, distribuciones por segmentos demográficos y de producto, métricas de concentración y percentiles de saldo, perfiles descriptivos de cliente, y comparativas entre entidades financieras genéricas. Los registros abarcan los cuatro niveles de complejidad y se distribuyen sobre un universo de doce entidades comparables. Por razones de confidencialidad, la tabla completa de preguntas no se reproduce en este documento; la descripción cuantitativa anterior caracteriza suficientemente la cobertura y el alcance del conjunto.

Cuadro 5.3: Campos del registro de evaluación (*golden record*)

Campo	Descripción
`id`	Identificador único de la pregunta
`question`	Pregunta en lenguaje natural
`expected.value`	Valor numérico esperado
`expected.formula`	Fórmula explícita que produce el valor esperado
`tolerance`	Margen de error aceptable para evaluación heurística
`complexity`	Nivel C1–C4
`source`	Origen del caso de uso (propuesta, cálculo manual, dashboard)

Este diseño presenta limitaciones que conviene explicitar. El conjunto está anclado a un período temporal fijo, por lo que no evalúa la capacidad del sistema para responder preguntas con dimensión temporal o de evolución. Las preguntas cubren métricas con respuesta numérica verificable, lo que deja fuera la calidad narrativa de las respuestas y la pertinencia de las visualizaciones generadas. Adicionalmente, al derivar mayoritariamente del dashboard validado, el conjunto podría infraestimar el rendimiento del sistema en preguntas exploratorias que un usuario formularía sin referencia previa a métricas conocidas. Estas limitaciones se retoman en la discusión de resultados de la Sección 5.6.

Con el conjunto de evaluación definido y sus propiedades caracterizadas, la sección siguiente establece las métricas con las que se mide el rendimiento del sistema sobre ese conjunto, y justifica por qué cada dimensión de medida aporta información que las demás no capturan.

5.3. Métricas utilizadas

Un sistema analítico conversacional no puede caracterizarse con una única cifra de rendimiento. La accuracy global indica si el sistema produce respuestas correctas, pero no dónde falla ni si su comportamiento es predecible entre ejecuciones. Por esta razón se definieron cuatro dimensiones de medida complementarias, cada una orientada a capturar un aspecto distinto del comportamiento del sistema.

Accuracy. Se distinguen dos variantes. La *strict accuracy* mide la proporción de preguntas cuya respuesta recibe clasificación `pass` sobre el total de preguntas evaluables, según la expresión

$$\text{Accuracy}_{\text{strict}} = \frac{\left|\{q \mid \text{score}(q) = \texttt{pass}\}\right|}{\left|\{q \mid \text{score}(q) \neq \texttt{not\_evaluable}\}\right|} \tag{5.1}$$

La *relaxed accuracy* amplía el numerador para incluir también las respuestas clasificadas como `partial`, es decir, aquellas que contienen la información correcta pero con alguna imprecisión en forma o escala. La distinción entre ambas variantes es útil en la práctica: una brecha amplia entre strict y relaxed indica que el sistema produce resultados parcialmente correctos con frecuencia, lo cual sugiere problemas de formulación o escala antes que errores de lógica analítica.

Accuracy por nivel de complejidad. Se calcula la strict accuracy de forma independiente para cada nivel C1–C4. Esta desagregación permite identificar el umbral a partir del cual el rendimiento se degrada de forma significativa y priorizar las mejoras en los niveles donde el impacto sobre los casos de uso reales es mayor.

Consistencia. Mide la proporción de ejecuciones en las que la misma pregunta produce el mismo resultado cuando se lanza en múltiples ocasiones bajo condiciones idénticas. La variabilidad estocástica propia de los modelos de lenguaje hace que esta dimensión sea relevante para cualquier sistema analítico destinado a la toma de decisiones: una respuesta que cambia entre ejecuciones no es fiable con independencia de su accuracy media.

Latencia. Se registran el percentil 50 y el percentil 95 del tiempo de respuesta. Durante las primeras iteraciones del desarrollo, la latencia media se situó en torno a 80,6 segundos, un valor aceptable frente al proceso manual que sustituye pero insuficiente para una interacción conversacional fluida. La migración del dataset al formato nativo del motor analítico utilizado, que carga en memoria únicamente las columnas necesarias para cada consulta y está optimizado para operaciones OLAP (*Online Analytical Processing*, procesamiento analítico en línea), redujo la latencia media a aproximadamente 23 segundos. Este ajuste no modificó la lógica del sistema ni afectó a los resultados de accuracy, lo que lo convierte en una mejora con un perfil de riesgo favorable.

Distribución de errores. Complementa las métricas de accuracy con información diagnóstica. Para cada respuesta clasificada como `fail` o `partial`, el juez asigna una categoría de la taxonomía cerrada de once tipos definida para el proyecto. El seguimiento de esta distribución entre iteraciones permite cuantificar el impacto de cada intervención sobre patrones de error específicos, y es la métrica que con mayor frecuencia orientó las decisiones de modificación del prompt durante el desarrollo.

La Tabla 5.4 recoge las cinco dimensiones con su tipo, objetivo de referencia y prioridad relativa en el análisis de resultados.

Cuadro 5.4: Métricas de evaluación del sistema

Métrica	**Tipo**	**Objetivo**	**Prioridad**
Strict accuracy	Ratio	Maximizar	Principal
Relaxed accuracy	Ratio	Maximizar	Principal
Accuracy C1–C4	Ratio × 4	Identificar degradación	Principal
Consistencia	Ratio	$> 90\,\%$	Complementaria
Latencia p50 / p95	Tiempo (s)	$< 30\,\text{s}$ p95	Complementaria
Distribución de errores	Distribución	Diagnóstico	Principal

Con el marco de medida establecido, la sección siguiente aplica estas métricas sobre el conjunto de evaluación descrito en la Sección 5.2 y presenta los resultados obtenidos a lo largo del proceso de desarrollo, desde las primeras iteraciones hasta la evaluación final del sistema.

5.4. Resultados cuantitativos

Las primeras evaluaciones de la primera iteración del sistema sobre preguntas de negocio reales arrojaron una strict accuracy inferior al 5 %. El resultado no fue una sorpresa: el sistema en su versión inicial carecía de catálogo semántico, y el modelo generaba consultas con columnas inexistentes, valores categóricos inventados y fórmulas que no correspondían a las definiciones de negocio del dominio. Sin el pipeline de evaluación construido en la Sección 5.1, ese comportamiento habría sido difícil de cuantificar y más difícil aún de corregir de forma sistemática. Con él, cada sprint de desarrollo produjo un diagnóstico concreto: qué categoría de error dominaba, qué la causaba y qué intervención podía reducirla.

La Figura 5.2 recoge la evolución de la strict accuracy a lo largo de las cuatro fases principales del desarrollo. A diferencia de una evaluación sobre un conjunto de preguntas fijo, cada fase se midió sobre un golden dataset progresivamente más amplio y representativo; por ello, las cifras reflejan tanto la maduración del sistema como el incremento de exigencia del conjunto de evaluación.

La incorporación del catálogo semántico fue la intervención con mayor impacto individual: eliminó el error dominante de las primeras semanas, que consistía en que

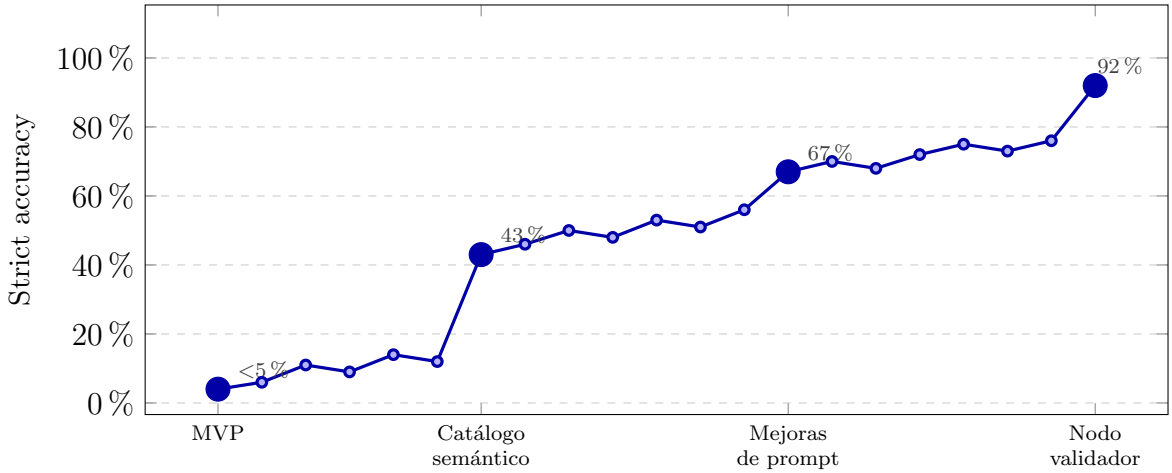


Figura 5.2: Evolución de la strict accuracy a lo largo del desarrollo. Los puntos grandes señalan las cuatro fases con mayor impacto; los puntos pequeños corresponden a iteraciones intermedias en las que se exploraron variaciones de prompt, catálogo y lógica de validación, con resultados no siempre positivos.

el sistema afirmaba la ausencia de datos cuando estos sí existían. Simultáneamente, el alcance del proyecto se amplió al escenario multi-entidad, con doce entidades comparables, y el conjunto de evaluación se sustituyó por un golden dataset más amplio, representativo y exigente. Medida sobre ese conjunto, la strict accuracy se situó en el 43 %. Este valor no refleja una regresión del sistema respecto a versiones anteriores, sino el incremento de dificultad del benchmark: el nuevo conjunto incluía consultas comparativas entre entidades, métricas de cuota calculadas sobre universos distintos y preguntas con múltiples ejes de análisis que el dataset de partida no contemplaba.

Las iteraciones siguientes introdujeron reglas explícitas de formulación multi-entidad, fórmulas de KPIs en el prompt y una regla obligatoria de agregación por usuario antes de calcular estadísticos, llevando la strict accuracy al 67 %. La adición del nodo validador, que detecta resultados incoherentes con la intención analítica y reintenta la consulta de forma automática, junto con la reformulación de las preguntas del golden dataset con especificación explícita de la población, la métrica, el denominador, los filtros aplicados y el grano de agregación, elevó la strict accuracy hasta el 92 % final.

Los casos no resueltos correspondieron a preguntas en las que el sistema devolvió una respuesta de fuera de alcance por una interpretación incorrecta de la intención analítica, a pesar de que los datos necesarios estaban disponibles. Este patrón apunta

a una limitación que las mejoras técnicas acumuladas no eliminaron por completo: con ciertos tipos de formulación, el modelo interpreta la pregunta de forma diferente a lo que el usuario pretendía, y genera una respuesta válida para esa interpretación alternativa. Las implicaciones de este comportamiento se examinan en la Sección 5.6.

5.5. Feedback cualitativo de usuarios

El pipeline de evaluación descrito en la Sección 5.1 opera sobre un conjunto de preguntas predefinidas en un entorno controlado. Este diseño garantiza reproducibilidad, pero tiene un límite estructural: solo puede detectar problemas que el evaluador anticipó al construir el golden dataset. Las preguntas que los usuarios reales formulan en producción, la forma en que las articulan y los aspectos de la respuesta que consideran insatisfactorios son información que ningún benchmark puede capturar por definición. Por esta razón, el sistema incorpora un mecanismo de retroalimentación directa de usuarios, satisfaciendo el requisito funcional RF6 definido en el Capítulo 3.

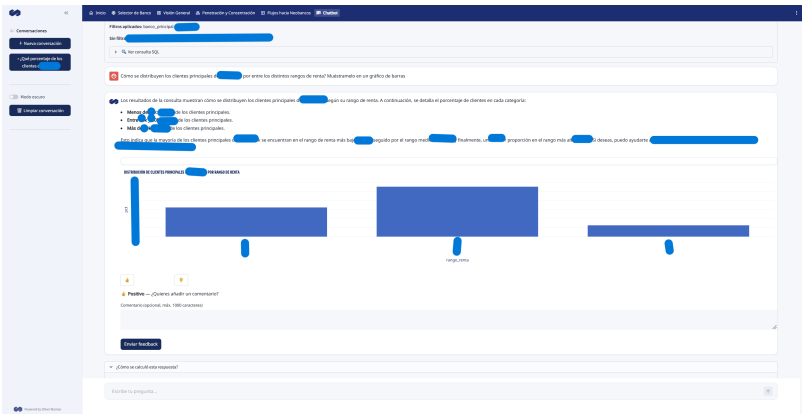


Figura 5.3: Interfaz gráfica del mecanismo de feedback de usuarios

La implementación consiste en una valoración binaria (like/dislike) accesible tras cada respuesta del chatbot, acompañada de un campo de comentario opcional. Desde el punto de vista técnico, cada interacción genera un evento estructurado de 14 campos que incluye los siguientes metadatos:

- Valoración binaria del usuario (like/dislike) y comentario opcional
- Timestamp del evento e identificador de la conversación a la que pertenece

- Consulta SQL ejecutada por el sistema
- Número de reintentos realizados por el nodo validador
- Errores registrados durante la ejecución
- Campos de trazabilidad del estado del agente

El evento se envía a un Application Insight en Azure, la plataforma de telemetría corporativa, que permite analizar los eventos de feedback en tiempo real, y generar análisis sobre el histórico de feedback. Además, los evíos de eventos se realizan de manera asíncrona, sin bloquear la interfaz, y se gestiona como operación de tipo *fire-and-forget*: un fallo en el envío no interrumpe la experiencia del usuario.

La Tabla 5.5 contrasta este mecanismo con la evaluación offline para clarificar qué aporta cada uno y por qué son complementarios.

Cuadro 5.5: Evaluación offline frente a feedback online

	Evaluación offline	**Feedback online**
Cuándo	Durante el desarrollo, antes del despliegue	En producción, con usuarios reales
Quién evalúa	LLM judge + heurístico	Usuario final
Preguntas cubiertas	Golden dataset predefinido	Cualquier pregunta real
Qué mide	Corrección numérica verificable	Utilidad percibida
Ventaja	Reproducible y automatizado	Detecta gaps no anticipados
Limitación	Solo cubre preguntas previstas	Volumen inicial reducido; sesgo de participación

El feedback recibido de socios y consultores durante el período de desarrollo fue mayoritariamente positivo: el 86 % de las valoraciones registradas correspondieron a likes. Los casos negativos, aunque minoritarios, resultaron especialmente valiosos porque señalaron dos problemas que el pipeline de evaluación no había detectado.

De los casos negativos emergieron dos problemas accionables. El primero fue la ausencia de explicación del procedimiento de cálculo: un socio indicó que no podía verificar si la respuesta era correcta porque el sistema no mostraba cómo había llegado a ella. Esto derivó en la incorporación de un nodo de trazabilidad dedicado, descrito en la Sección 4.9. El segundo fue la sensibilidad del sistema ante preguntas

con ambigüedad analítica, que producían respuestas internamente consistentes pero desalineadas con la intención del usuario. Como respuesta, se elaboró una guía de buenas prácticas para socios y consultores con cuatro bloques:

- **Checklist de formulación:** población objetivo, métrica, denominador, filtros y grano de agregación.
- **Ejemplos contrastados:** preguntas ambiguas frente a su versión analíticamente especificada.
- **Patrones a evitar:** formulaciones que omiten el universo de referencia o mezclan denominadores sin distinguirlos.
- **Convenciones del dominio:** conceptos financieros con múltiples interpretaciones posibles en el dataset.

El volumen actual de feedback no permite extraer conclusiones estadísticas robustas, y no se pretende hacerlo, ya que el valor real del mecanismo de feedback reside en haber funcionado como canal de detección de problemas cualitativos que el entorno controlado de evaluación no podía anticipar, y en haber dejado una infraestructura de observabilidad operativa para iteraciones futuras.

5.6. Discusión de resultados

La combinación de evaluación continua mediante benchmarks cerrados y retroalimentación directa de usuarios permitió orientar cada sprint hacia la causa raíz del problema dominante en ese momento. El resultado acumulado de ese proceso se refleja en la Tabla 5.6, que recoge la correspondencia entre las decisiones de diseño del Capítulo 3 y los efectos observados en la evaluación.

Alcanzado el 92 % de accuracy en la evaluación final, el factor limitante ya no reside en la capacidad técnica del sistema para generar SQL correcto. Reside en la habilidad del usuario para especificar sin ambigüedad lo que quiere calcular. Cuando la pregunta explicita la población, la métrica, el denominador, los filtros y el grano de agregación, el sistema interpreta correctamente la intención analítica. Cuando alguno de esos elementos queda implícito, el modelo selecciona una interpretación plausible que puede diferir de la esperada.

Fortalezas. Los resultados demuestran que el sistema resuelve correctamente una amplia variedad de cálculos analíticos sobre datos financieros agregados, entre ellos:

Cuadro 5.6: Decisiones de diseño y su efecto sobre el rendimiento observado

Decisión de diseño	**Efecto observado**
Catálogo semántico con ejemplos de valores	Eliminación del error de ausencia de datos; accuracy de <5 % a 85 %
Nodo validador con reintento automático	Autocorrección sin intervención humana; +6 puntos porcentuales en benchmark multi-entidad
Sandbox de solo lectura y límite de filas	Cero incidentes de seguridad en pruebas y producción
Nodo de trazabilidad	Respuestas auditables por el usuario; reducción del feedback negativo por opacidad
Telemetría asíncrona	Observabilidad en producción sin impacto en latencia

- Ratios de penetración y cuota de mercado sobre universos de clientes definidos
- Distribuciones demográficas por segmentos de edad, renta y perfil de producto
- Percentiles de saldo y métricas de concentración sobre subpoblaciones filtradas
- Comparativas simultáneas entre múltiples entidades financieras
- Preguntas exploratorias con agrupaciones, filtros cruzados y cálculos derivados en una sola consulta

Limitaciones. La sensibilidad a la formulación de la pregunta es la debilidad más relevante del sistema en su estado actual: una pregunta subespeficificada puede producir una respuesta ejecutable pero funcionalmente incorrecta sin que el sistema lo detecte como error. La segunda limitación es estructural al dataset: al corresponder a un snapshot trimestral de productos financieros activos, no permite análisis de evolución temporal ni comparativas entre períodos. Ambas limitaciones delimitan el espacio de mejora para iteraciones futuras, que se desarrolla en el Capítulo 6.

Capítulo 6

Conclusiones y Trabajo Futuro

6.1. Consecución de objetivos

El objetivo central del proyecto era permitir que socios y consultores sin perfil técnico pudieran obtener inteligencia de negocio sobre datos financieros agregados mediante lenguaje natural, sin depender de un analista intermediario ni formular consultas SQL. Los resultados de la evaluación respaldan ese objetivo: el sistema alcanza una accuracy del 92 % sobre 38 insights reales del dominio financiero, fue desplegado en producción y recibió un 86 % de valoraciones positivas de los usuarios finales.

Los objetivos específicos definidos en el Capítulo 1 se evalúan a continuación:

- **O1 — Habilitar el autoservicio analítico para perfiles no técnicos.** Cumplido. El sistema permite a socios y consultores obtener indicadores financieros calculados y verificados mediante lenguaje natural, sin intervención técnica. El despliegue en producción con usuarios reales y el feedback recibido confirman que el sistema resuelve el problema en condiciones de uso real. Para maximizar los resultados, se elaboró una guía de buenas prácticas que incluye un checklist de formulación, ejemplos contrastados de preguntas y convenciones del dominio financiero, de modo que cualquier usuario pueda aprender a especificar su intención analítica con precisión.

- **O2 — Garantizar que el sistema opera sobre las definiciones de métricas validadas por la firma.** Cumplido. El catálogo semántico codifica las definiciones de dominio, las fórmulas de KPIs y los valores posibles de las variables financieras, asegurando que los cálculos producidos sean coherentes con

las reglas de negocio establecidas. Esta intervención fue la de mayor impacto individual sobre la accuracy, elevándola del 5 % al 85 % en entidad única.

- **O3 — Dotar al sistema de mecanismos de validación y autocorrección autónomos.** Cumplido. El nodo validador detecta resultados incoherentes con la intención analítica de la pregunta y reintenta la consulta de forma automática, sin intervención del usuario. Su incorporación aportó un incremento de seis puntos porcentuales sobre el benchmark multi-entidad.
- **O4 — Hacer cada respuesta auditable y reproducible.** Cumplido. El nodo de trazabilidad genera para cada turno una explicación en lenguaje natural del procedimiento seguido, incluyendo la métrica aplicada, los filtros utilizados y la consulta ejecutada. Este objetivo surgió directamente del feedback de usuarios en producción, que señalaron la opacidad del sistema como la principal barrera para confiar en los resultados.
- **O5 — Presentar los resultados en formatos directamente utilizables.** Cumplido. El sistema genera visualizaciones interactivas cuando la naturaleza del resultado lo justifica, y expone los datos tabulares en un formato exportable compatible con hojas de cálculo, permitiendo que el output del chatbot se integre directamente en presentaciones y análisis de cliente.

6.2. Implicaciones para el negocio

El sistema desarrollado reduce a segundos una operación que anteriormente requería horas o días de trabajo técnico: obtener un indicador financiero calculado y verificado sobre datos agregados de clientes. Para el contexto de consultoría en el que se enmarca el proyecto, esta reducción tiene implicaciones concretas sobre cómo se preparan y presentan los análisis.

Un socio o consultor puede explorar el dataset directamente durante una reunión con cliente, formular hipótesis en tiempo real y verificarlas sin depender de un ciclo de petición y entrega con el equipo técnico. Eso desplaza el valor del analista desde la ejecución de consultas predefinidas hacia la interpretación y el diseño de preguntas de mayor profundidad.

La trazabilidad integrada en cada respuesta añade una dimensión especialmente relevante en el contexto financiero: el usuario puede verificar qué interpretó el sistema, qué filtros aplicó y qué fórmula utilizó antes de confiar en el resultado. En un entorno donde la trazabilidad de los cálculos es un requisito operativo, esta propiedad forma parte del valor entregado, no es un añadido opcional.

La replicabilidad del enfoque es otra implicación a considerar. La arquitectura modular, el catálogo semántico versionado y el pipeline de evaluación automatizada constituyen un patrón técnico transferible a otros dominios de datos dentro de la firma, con un esfuerzo de adaptación acotado principalmente a la redefinición del catálogo y el golden dataset.

6.3. Trabajo futuro

Las líneas de mejora más relevantes se derivan directamente de las limitaciones identificadas durante el desarrollo y la evaluación.

- **Clarificación activa ante preguntas ambiguas.** El mecanismo más directo para reducir los casos de interpretación incorrecta es incorporar un nodo de clarificación que, ante preguntas con parámetros implícitos, solicite al usuario los elementos que faltan antes de generar la consulta. Una alternativa complementaria son las preguntas sugeridas derivadas del catálogo semántico, que guíen al usuario hacia formulaciones con mayor probabilidad de producir resultados correctos sin interrumpir el flujo conversacional.
- **Optimización del modelo por nodo.** La arquitectura modular permite asignar modelos de distinto tamaño y capacidad a cada nodo según sus requisitos de razonamiento. Los nodos de generación de SQL y validación requieren alta capacidad de razonamiento; los nodos de trazabilidad y decisión de visualización podrían operar con modelos más ligeros sin pérdida apreciable de calidad, reduciendo latencia y coste de inferencia sin modificar la lógica del sistema.
- **Escalado del catálogo semántico mediante RAG.** A medida que el número de datasets crece, la inyección directa del catálogo completo en el prompt puede saturar la ventana de contexto. La migración hacia un enfoque de Retrieval-Augmented Generation (RAG, o generación aumentada por recuperación) permitiría recuperar únicamente las secciones relevantes para cada pregunta, manteniendo la precisión semántica con un footprint de contexto menor.

Bibliografía

[1] Claudia Imhoff y Colin White. *Self-Service Business Intelligence: Empowering Users to Generate Insights*. Best Practices Report. Q3 2011. TDWI Research, 2011.

[2] Paul Alpar y Mario Schulz. "Self-Service Business Intelligence". En: *Business & Information Systems Engineering* 58.2 (2016), págs. 151-155. DOI: 10.1007/s12599-016-0424-6.

[3] Jens Passlick, Benedikt Lebek y Michael H. Breitner. "User-Related Challenges of Self-Service Business Intelligence". En: *Information Systems Management* (2020). DOI: 10.1080/10580530.2020.1814458.

[4] Christian Lennerholt, Joeri van Laere y Eva Söderström. "Implementation Challenges of Self-Service Business Intelligence: A Literature Review". En: *Proceedings of the 51st Hawaii International Conference on System Sciences (HICSS)*. 2018, págs. 5055-5063.

[5] I. Androutsopoulos, G. D. Ritchie y P. Thanisch. *Natural Language Interfaces to Databases - An Introduction*. 1995. arXiv: cmp-lg/9503016 [cmp-lg]. URL: https://arxiv.org/abs/cmp-lg/9503016.

[6] Victor Zhong, Caiming Xiong y Richard Socher. *Seq2SQL: Generating Structured Queries from Natural Language using Reinforcement Learning*. 2017. arXiv: 1709.00103. URL: https://arxiv.org/abs/1709.00103.

[7] Tao Yu et al. "Spider: A Large-Scale Human-Labeled Dataset for Complex and Cross-Domain Semantic Parsing and Text-to-SQL Task". En: *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 2018, págs. 3911-3921.

[8] Bailin Wang et al. "RAT-SQL: Relation-Aware Schema Encoding and Linking for Text-to-SQL Parsers". En: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, 2020, págs. 7567-7578.

[9] Torsten Scholak, Nathan Schucher y Dzmitry Bahdanau. "PICARD: Parsing Incrementally for Constrained Auto-Regressive Decoding from Language Models". En: *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 2021, págs. 9895-9901.

[10] Haoyang Li et al. "RESDSQL: Decoupling Schema Linking and Skeleton Parsing for Text-to-SQL". En: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 37. 11. 2023, págs. 13067-13075.

[11] Mohammadreza Pourreza y Davood Rafiei. "DIN-SQL: Decomposed In-Context Learning of Text-to-SQL with Self-Correction". En: *Advances in Neural Information Processing Systems*. Vol. 36. 2023, págs. 36339-36348. URL: https://arxiv.org/abs/2304.11015.

[12] Dawei Gao et al. "Text-to-SQL Empowered by Large Language Models: A Benchmark Evaluation". En: *Proceedings of the VLDB Endowment* 17.5 (2024), págs. 1132-1145. DOI: 10.14778/3641204.3641221. URL: https://www.vldb.org/pvldb/vol17/p1132-gao.pdf.

[13] Jonathan Herzig et al. "TaPas: Weakly Supervised Table Parsing via Pre-training". En: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Online: Association for Computational Linguistics, 2020, págs. 4320-4333. DOI: 10.18653/v1/2020.acl-main.398. URL: https://aclanthology.org/2020.acl-main.398/.

[14] Qian Liu et al. "TAPEX: Table Pre-training via Learning a Neural SQL Executor". En: *International Conference on Learning Representations*. 2022. URL: https://openreview.net/forum?id=O50443AsCP.

[15] Lianmin Zheng et al. "Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena". En: *Advances in Neural Information Processing Systems*. Vol. 36. 2023. URL: https://proceedings.neurips.cc/paper_files/paper/2023/hash/91f18a1287b398d378ef22505bf41832-Abstract-Datasets_and_Benchmarks.html.

[16] Yang Liu et al. "G-Eval: NLG Evaluation using GPT-4 with Better Human Alignment". En: *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 2023, págs. 2511-2522. DOI: 10.18653/v1/2023.emnlp-main.153. URL: https://aclanthology.org/2023.emnlp-main.153/.

[17] Jason Wei et al. "Chain-of-Thought Prompting Elicits Reasoning in Large Language Models". En: *Advances in Neural Information Processing Systems*. Vol. 35. 2022, págs. 24824-24837. URL: https://proceedings.neurips.cc/paper_files/paper/2022/hash/9d5609613524ecf4f15af0f7b31abca4-Abstract-Conference.html.

[18] Shunyu Yao et al. "ReAct: Synergizing Reasoning and Acting in Language Models". En: *International Conference on Learning Representations*. 2023. URL: https://openreview.net/forum?id=WE_vluYUL-X.

[19] Timo Schick et al. "Toolformer: Language Models Can Teach Themselves to Use Tools". En: *Advances in Neural Information Processing Systems*. Vol. 36. 2023. URL: https://proceedings.neurips.cc/paper_files/paper/2023/hash/d842425e4bf79ba039352da0f658a906-Abstract-Conference.html.

[20] Shishir G. Patil et al. "Gorilla: Large Language Model Connected with Massive APIs". En: *arXiv preprint arXiv:2305.15334* (2023). URL: https://arxiv.org/abs/2305.15334.

[21] Noah Shinn et al. "Reflexion: Language Agents with Verbal Reinforcement Learning". En: *Advances in Neural Information Processing Systems*. Vol. 36. 2023. URL: https://proceedings.neurips.cc/paper_files/paper/2023/hash/1b44b878bb782e6954cd888628510e90-Abstract-Conference.html.

[22] Aman Madaan et al. "Self-Refine: Iterative Refinement with Self-Feedback". En: *Advances in Neural Information Processing Systems*. Vol. 36. 2023. URL: https://proceedings.neurips.cc/paper_files/paper/2023/hash/91edff07232fb1b55a505a9e9f6c0ff3-Abstract-Conference.html.

[23] Qingyun Wu et al. "AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversation". En: *arXiv preprint arXiv:2308.08155* (2023). URL: https://arxiv.org/abs/2308.08155.

[24] Sirui Hong et al. "MetaGPT: Meta Programming for A Multi-Agent Collaborative Framework". En: *International Conference on Learning Representations*. 2024. URL: https://openreview.net/forum?id=VtmBAGCN7o.

[25] OpenAI. *Enterprise Privacy at OpenAI*. https://openai.com/enterprise-privacy/. Accessed: 2026-05-12. 2026.

[26] Microsoft. *Overview of Copilot for Power BI*. https://learn.microsoft.com/en-us/power-bi/create-reports/copilot-introduction. Accessed: 2026-05-12. 2026.

[27] Databricks. *Use the Genie Interface*. https://docs.databricks.com/aws/en/workspace/genie. Accessed: 2026-05-12. 2026.

[28] Snowflake. *Cortex Analyst*. https://docs.snowflake.com/en/user-guide/snowflake-cortex/cortex-analyst. Accessed: 2026-05-12. 2026.

[29] European Commission. *Data Protection Explained*. https://commission.europa.eu/law/law-topic/data-protection/data-protection-explained_en. Accessed: 2026-05-12. 2026.

[30] European Commission. *AI Act Enters into Force*. https://commission.europa.eu/news-and-media/news/ai-act-enters-force-2024-08-01_en. Accessed: 2026-05-12. 2024.

[31] European Central Bank. *The Revised Payment Services Directive*. https://www.ecb.europa.eu/press/intro/mip-online/2018/html/1803_revisedpsd.en.html. Accessed: 2026-05-12. 2018.

[32] European Insurance and Occupational Pensions Authority. *Digital Operational Resilience Act*. https://www.eiopa.europa.eu/digital-operational-resilience-act-dora_en. Accessed: 2026-05-12. 2026.

[33] Jinyang Li et al. "Can LLM Already Serve as A Database Interface? A BIg Bench for Large-Scale Database Grounded Text-to-SQLs". En: *Advances in Neural Information Processing Systems*. Vol. 36. 2023. URL: https://proceedings.neurips.cc/paper_files/paper/2023/hash/83fc8fab1710363050bbd1d4b8cc0021-Abstract-Datasets_and_Benchmarks.html.