PROYECTO FIN DE GRADO

TÍTULO: Sistema modular inteligente para la traducción de consultas en lenguaje natural (NLQs) a SQL mediante LLMs

AUTOR: Fidel Núñez Friera

TITULACIÓN: Grado en Ingeniería Telemática

TUTOR: Mario San Emeterio de la Parte

DEPARTAMENTO: Departamento de Ingeniería Telemática y Electrónica (DTE)

V°B° TUTOR

Miembros del Tribunal Calificador:

PRESIDENTA: Maria Larrosa Navarro

TUTOR: Mario San Emeterio de la Parte

SECRETARIO: José Fernán Martínez Ortega

Fecha de lectura:

Calificación:

El secretario

Agradecimientos

A mi familia por su apoyo incondicional, a mis gatos por hacer más llevadero este largo camino, y a los amigos de verdad que me han acompañado hasta el final. Agradezco especialmente a mi tutor, Mario, la oportunidad de realizar un proyecto tan interesante y divertido, y su disposición constante para compartir su conocimiento conmigo.

Resumen

Sistema modular inteligente para la traducción de consultas en lenguaje natural (NLQs) a SQL mediante LLMs

El presente Trabajo Fin de Grado tiene como objetivo diseñar e implementar un sistema capaz de transformar consultas en lenguaje natural (Natural Language Query (NLQ)) en sentencias Structured Query Language (SQL) válidas y ejecutables sobre una base de datos PostgreSQL. El propósito fundamental es reducir la barrera técnica entre los datos contenidos en una Base de Datos (BD) y el usuario no técnico que, pese a tener posesión de esos datos, no puede acceder a ellos dada su falta de conocimiento de lenguaje SQL. Esto democratiza el acceso a los datos y facilita la explotación de la información en entornos reales.

El proyecto se enmarca en un contexto tecnológico caracterizado por la irrupción de modelos de lenguaje de gran escala (Large Language Models (LLMs)), que han demostrado una notable capacidad para traducir instrucciones expresadas en lenguaje natural a lenguajes formales. No obstante, su aplicación práctica presenta condicionantes importantes: la ambigüedad de las consultas, el riesgo de generar sentencias no válidas o destructivas, y la necesidad de garantizar tiempos de respuesta aceptables en un entorno interactivo. Estos retos se han afrontado mediante una arquitectura modular que integra una interfaz web tipo chatbot para la interacción con el usuario, un orquestador que coordina los distintos módulos, un primer módulo basado en LLM encargado de extraer la intención semántica de la consulta en lenguaje natural, un segundo módulo basado en LLM que genera la sentencia SQL candidata, un validador basado en PostgreSQL que verifica la sintaxis y bloquea instrucciones no permitidas, garantizando que solo se ejecuten consultas de tipo `SELECT`, y finalmente un ejecutor que obtiene los datos de la BD para presentarlos al usuario.

El caso de estudio seleccionado corresponde al ámbito agrícola y ganadero, utilizando datos de collares inteligentes aplicados al ganado, que registran parámetros como posición, temperatura y anomalías [1]. Este dominio resulta especialmente representativo, ya que los ganaderos y trabajadores del sector rara vez poseen conocimientos técnicos de SQL o tecnologías de bases de datos, lo que amplifica la barrera de acceso a la información. Por ello, el sistema debe ser lo más simple e intuitivo posible, capaz de responder eficazmente incluso ante consultas formuladas con baja capacidad semántica, garantizando así su aplicabilidad en un contexto amplio y realista.

La metodología de pruebas se diseñó siguiendo estándares internacionales de verificación y validación (IEEE 1012-2016 [2] e ISO/IEC/IEEE 29119 [3] [4]), abarcando tanto pruebas funcionales como no funcionales. Asimismo, el diseño del sistema se ha guiado por principios de ingeniería de software, modularidad y reproducibilidad, asegurando que el entorno pueda desplegarse con una configuración mínima. Se ha adoptado un enfoque de seguridad por defecto, limitando la ejecución a consultas de lectura, lo que aporta un valor diferencial frente a otras aproximaciones del estado del arte que no siempre incorporan estos mecanismos de control.

En el plano económico, se ha realizado un análisis de costes que incluye amortización de equipamiento, uso de la Application Programming Interface (API) de OpenAI y mano de obra, resultando en un presupuesto total estimado de 14.890,80 €. El coste operativo generado por las consultas durante todo el proceso de pruebas fue inferior a 0,05 €, lo que confirma la

viabilidad económica de la solución. Desde una perspectiva social y ética, el sistema contribuye a la reducción de la brecha digital al permitir que usuarios sin formación técnica accedan a bases de datos, garantizando además la privacidad de la información al no incluir datos personales en los *prompts*. En cuanto al impacto ambiental, se ha verificado que el consumo energético es reducido gracias al carácter ligero del sistema y a la optimización de las peticiones a los modelos de lenguaje.

Los resultados experimentales, obtenidos a partir de más de 300 ejecuciones de consultas, muestran que el sistema alcanza un 100 % de aciertos en consultas de nivel profesional, un 87 % en formulaciones estándar y un 63 % en consultas de baja capacidad, con tiempos de respuesta dentro de los márgenes definidos. Las consultas erróneas o maliciosas fueron correctamente manejadas por el sistema, reforzando la robustez y seguridad de la solución. La validación cualitativa con el tutor, actuando como usuario experto, confirmó además la usabilidad de la interfaz y la utilidad práctica del sistema.

Más allá de los resultados inmediatos, el sistema constituye una base sólida para futuras líneas de investigación y transferencia. Entre ellas se incluyen su despliegue en entornos cloud, la extensión a consultas multibase de datos, la integración con distintos proveedores de LLM o la evaluación con usuarios finales a mayor escala. Además, el proyecto posee un valor formativo y de transferencia académica, al reunir de forma aplicada tecnologías de actualidad —procesamiento de lenguaje natural, validación en bases de datos y arquitecturas modulares— que pueden servir como modelo en contextos docentes o de investigación.

En conclusión, el sistema desarrollado cumple los requisitos funcionales y no funcionales establecidos, y demuestra ser una herramienta eficaz, segura y económicamente viable para permitir el acceso a bases de datos mediante consultas en lenguaje natural. Su arquitectura modular lo hace fácilmente adaptable a otros dominios —como sanidad, logística, educación o comercio electrónico— mediante la simple adaptación del esquema de base de datos y del *prompt*. Además, el trabajo se alinea con los Objetivos de Desarrollo Sostenible 9 (Industria, Innovación e Infraestructura), 11 (Ciudades y Comunidades Sostenibles) y 13 (Acción por el Clima), reforzando su valor social y su relevancia global.

Abstract

Intelligent Modular System for Translating Natural Language Queries (NLQs) into SQL using LLMs

The aim of this Bachelor's Thesis is to design and implement a system capable of transforming NLQs into valid and executable SQL statements over a PostgreSQL database. The fundamental purpose is to reduce the technical barrier between the data stored in a database and non-technical users, who, despite having access to the data, cannot exploit it due to their lack of SQL knowledge. This approach democratizes access to information and facilitates its use in real-world contexts.

The project is framed within a technological context characterized by the emergence of large language models (LLMs), which have shown remarkable ability to translate instructions expressed in natural language into formal languages. However, their practical application poses important challenges: ambiguity in queries, the risk of generating invalid or destructive statements, and the need to ensure acceptable response times in an interactive environment. These challenges were addressed through a modular architecture that integrates a web-based chatbot interface for user interaction, an orchestrator that coordinates the workflow, a first module based on LLM responsible for extracting the semantic intent of the query, a second module based on LLM that generates the candidate SQL statement, a PostgreSQL-based validator that checks syntax and blocks non-permitted instructions, ensuring that only `SELECT` queries are executed, and finally an executor that retrieves the requested data and presents it to the user.

The selected case study focuses on the agricultural and livestock domain, using data from smart collars applied to cattle, which record parameters such as position, temperature, and anomalies [1]. This domain is especially representative, as farmers and workers in the sector rarely possess technical knowledge of SQL or database technologies, amplifying the barrier to information access. Therefore, the system had to be simple and intuitive, capable of effectively handling even low-semantic queries, ensuring applicability in a broad and realistic context.

The testing methodology followed international standards for verification and validation (IEEE 1012-2016 [2] and ISO/IEC/IEEE 29119 [3] [4]), covering both functional and non-functional tests. Furthermore, the system design was guided by principles of software engineering, modularity, and reproducibility, ensuring that the environment can be deployed with minimal configuration. A "security by default" approach was adopted, restricting execution to read-only queries, which provides a differential value compared to other state-of-the-art approaches that do not always incorporate such control mechanisms.

From an economic perspective, a cost analysis was carried out, including equipment amortization, OpenAI API usage, and labor, resulting in a total estimated budget of €14,890.80. The operational cost generated by queries during the entire testing process was below €0.05, confirming the economic feasibility of the solution. From a social and ethical perspective, the system contributes to reducing the digital divide by enabling users without technical training to access databases, while ensuring privacy by excluding personal data from prompts. Regarding environmental impact, the system was shown to have a low energy footprint thanks to its lightweight nature and the optimization of requests to the language models.

The experimental results, obtained from more than 300 query executions, show that the system

achieved 100 % accuracy in professional-level queries, 87 % in standard formulations, and 63 % in low-capacity queries, with response times within the defined thresholds. Erroneous or malicious queries were correctly handled by the validator, reinforcing the robustness and security of the solution. The qualitative validation with the supervisor, acting as a domain expert, further confirmed the usability of the interface and the practical utility of the system.

Beyond the immediate results, the system constitutes a solid foundation for future lines of research and technology transfer. These include deployment in cloud environments, extension to multi-database queries, integration with different LLM providers, and large-scale evaluation with end users. The project also provides academic and educational value by bringing together cutting-edge technologies—natural language processing, database validation, and modular architectures—in an applied manner that can serve as a model in both teaching and research contexts.

In conclusion, the developed system meets the defined functional and non-functional requirements, and proves to be an effective, secure, and economically viable tool for enabling access to databases through natural language queries. Its modular architecture makes it easily adaptable to other domains—such as healthcare, logistics, education, or e-commerce—by simply adjusting the database schema and the prompt. Furthermore, the project aligns with Sustainable Development Goals 9 (Industry, Innovation and Infrastructure), 11 (Sustainable Cities and Communities), and 13 (Climate Action), reinforcing its social value and global relevance.

Índice de contenidos

Índice de figuras

Índice de tablas

Siglas

API	Application Programming Interface.
BBDD	Bases de Datos.
BD	Base de Datos.
ETSIST	Escuela Técnica Superior de Ingeniería y Sistemas de Telecomunicación.
GRyS	Group of Next-Generation Networks and Services.
HTTPS	Hypertext Transfer Protocol Secure.
IA	Inteligencia Artificial.
IoT	Internet of Things.
JSON	JavaScript Object Notation.
LLM	Large Language Model.
MVP	Producto Mínimo Viable.
NL2SQL	Natural Language to SQL.
NLIDB	Natural Language Interface to Databases.
NLP	Natural Language Processing.
NLQ	Natural Language Query.
ODS	Objetivos de Desarrollo Sostenible.
ONU	Organización de las Naciones Unidas.
REST	Representational State Transfer.
SQL	Structured Query Language.
TLS	Transport Layer Security.
UI	User Interface.
UPM	Universidad Politécnica de Madrid.

1. Introducción

Este capítulo presenta el contexto general, la motivación y los objetivos que han guiado la realización del presente Trabajo Fin de Grado. Asimismo, se ofrece una visión preliminar de la solución planteada y se describe la estructura del manuscrito.

1.1. Marco y motivación del proyecto

El acceso a bases de datos mediante consultas en lenguaje natural constituye un área de creciente interés en el ámbito de la ingeniería telemática y la ciencia de datos. La complejidad del lenguaje SQL supone una barrera para usuarios no especializados, lo que limita la explotación de la información almacenada en sistemas relacionales.

En este contexto, los modelos de lenguaje de gran escala (LLMs) han demostrado un gran potencial para traducir instrucciones expresadas en lenguaje natural a lenguajes formales. Sin embargo, su aplicación práctica requiere afrontar retos adicionales: ambigüedad de las consultas, riesgo de generar sentencias no válidas o destructivas, y necesidad de garantizar tiempos de respuesta aceptables.

El presente proyecto aborda esta problemática mediante el diseño e implementación de un sistema que permite a usuarios no técnicos acceder a una base de datos PostgreSQL utilizando consultas en lenguaje natural. Esto es posible gracias a una arquitectura modular que implementa un flujo de procesamiento completo, capaz de generar, validar y ejecutar consultas SQL a partir de una NLQ. El caso de estudio se centra en el dominio agrícola y ganadero, empleando datos de sensores de collares inteligentes para monitorizar ganado en base a parámetros como su posición, temperatura y detección de anomalías [1]. La motivación principal es facilitar la interacción entre usuarios finales y sistemas de información complejos, democratizando el acceso a los datos y mejorando la toma de decisiones en entornos reales.

1.2. Proyecto AFarCloud y marco de investigación

El desarrollo de este trabajo se enmarca dentro del proyecto europeo *AFarCloud* (*Aggregate Farming in the Cloud*, H2020–783221) [1], cuyo objetivo es crear una plataforma ciber–física interoperable que integre sensores, vehículos autónomos y sistemas de análisis distribuidos para optimizar los procesos agrícolas y ganaderos. El proyecto aborda la digitalización del sector agropecuario mediante el uso de tecnologías Internet of Things (IoT), análisis en la nube y automatización, fomentando una agricultura más sostenible, eficiente y conectada. Esta línea de investigación se desarrolla en el seno del Group of Next-Generation Networks and Services (GRyS) de la Escuela Técnica Superior de Ingeniería y Sistemas de Telecomunicación (ETSIST) – Universidad Politécnica de Madrid (UPM), que centra su actividad en el diseño y evaluación de arquitecturas inteligentes de comunicación y servicios avanzados basados en datos.

1.3. Contenido del manuscrito

El documento se organiza de la siguiente manera:

- El capítulo 2 presenta el *estado del arte*, revisando las soluciones previas en el campo de la traducción de lenguaje natural a SQL, así como los trabajos relacionados con el uso de LLMs en entornos de bases de datos.
- El capítulo 3 define las especificaciones y restricciones del proyecto y especifica los requisitos funcionales y no funcionales que guían el desarrollo.
- El capítulo 4 describe la propuesta de solución, detallando tanto la arquitectura global como los módulos que componen el sistema y su interacción.
- El capítulo 5 presenta la verificación y validación del sistema mediante pruebas funcionales, no funcionales y la validación con el usuario final.
- El capítulo 6 detalla el presupuesto del proyecto, considerando costes directos e indirectos.
- El capítulo 7 analiza el impacto tecnológico, económico, social, ético y ambiental del sistema, así como su contribución a los Objetivos de Desarrollo Sostenible.
- Finalmente, el capítulo 8 recoge las conclusiones generales, los logros alcanzados, las limitaciones detectadas y las posibles líneas de trabajo futuro.

2. Estado del arte

2.1. Introducción

Este capítulo describe el marco tecnológico sobre el que se fundamenta el proyecto, centrado en la conversión de NLQs a lenguaje estructurado SQL. Para ello, se analizan los enfoques históricos y las principales líneas de investigación actuales, evaluando cómo ha evolucionado esta tarea desde soluciones manuales o basadas en reglas hasta el uso de LLMs, que permiten automatizar el proceso con una alta capacidad de comprensión semántica.

En primer lugar, se realiza una revisión crítica de la literatura relevante, destacando los principales hitos y enfoques dentro del ámbito del procesamiento del lenguaje natural aplicado al acceso a bases de datos. A continuación, se presentan las tecnologías existentes más representativas, incluyendo modelos de generación de consultas, herramientas de integración y arquitecturas de sistema. Finalmente, se lleva a cabo un análisis comparativo que justifica la elección de la solución tecnológica adoptada en este proyecto, basada en LLMs accesibles por API y una arquitectura modular.

El objetivo es proporcionar al lector una visión clara y estructurada del estado actual del conocimiento en esta área, sirviendo como base para entender las decisiones técnicas que se detallarán en capítulos posteriores.

2.2. Revisión de la literatura

A continuación, se expone un análisis crítico de las contribuciones más relevantes relacionadas con la temática del proyecto.

2.2.1. Ámbito de estudio

La transformación automática de NLQs a instrucciones estructuradas en lenguaje SQL constituye una línea de trabajo en la intersección entre el Natural Language Processing (NLP), el acceso a bases de datos relacionales y la ingeniería de software orientada a la interacción hombre-máquina [5]. El objetivo principal de esta tarea es facilitar el acceso a información estructurada a través de interfaces accesibles para usuarios no técnicos, eliminando la necesidad de conocer el lenguaje SQL ni la estructura interna de la base de datos.

En este contexto, el paradigma del IoT (Internet of Things) se presenta como un marco tecnológico clave que refleja la intersección entre la integración de tecnologías avanzadas —sensores, conectividad y análisis de datos— y su uso por parte de personas sin formación técnica en entornos productivos. Este enfoque promueve la creación de sistemas inteligentes capaces de interpretar información compleja y presentarla de forma comprensible, alineándose con los objetivos de accesibilidad y automatización del presente proyecto.

Esta problemática ha cobrado relevancia en contextos donde existen grandes volúmenes de

datos disponibles, pero donde los usuarios finales carecen de formación técnica. Sectores como la agricultura, la sanidad o la administración pública presentan casos de uso claros, en los que se requiere acceder a información específica sin pasar por intermediarios técnicos ni recurrir a herramientas complejas [6].

El lenguaje natural plantea importantes desafíos técnicos. Entre ellos destacan la ambigüedad semántica, la variabilidad en las formas de expresión, la presencia de errores gramaticales y ortográficos, y la necesidad de interpretar correctamente tanto el significado de la petición como el esquema de la base de datos a la que se dirige. Estos factores dificultan el diseño de sistemas robustos, especialmente cuando se pretende que sean completamente autónomos y accesibles.

La evolución reciente de los LLMs ha abierto nuevas posibilidades para abordar esta tarea [5]. A diferencia de los sistemas tradicionales basados en reglas o en plantillas, los LLMs son capaces de comprender consultas formuladas de forma libre y generar salidas estructuradas con un alto grado de precisión, siempre que se disponga de un diseño adecuado del contexto y de técnicas de control como el *prompt engineering* o el uso de validadores automáticos [7]. Esta capacidad convierte a los LLMs en una herramienta viable para implementar interfaces conversacionales que actúen como intermediarios inteligentes entre el usuario y la base de datos, eliminando barreras técnicas y mejorando la accesibilidad a la información.

2.2.2. Trabajos previos

Las primeras aproximaciones al problema de traducir lenguaje natural a SQL surgieron a finales de los años 70, con el desarrollo de interfaces en lenguaje natural sobre bases de datos relacionales. Sistemas pioneros como *LUNAR* [8] y *ASK* [9] seguían un enfoque *rule-based*, basado en reglas léxicas y sintácticas estrictas que intentaban mapear directamente estructuras del lenguaje natural a operaciones SQL. Este enfoque permitía cierto grado de comprensión, pero su cobertura y flexibilidad eran muy limitadas: cualquier desviación gramatical o ambigüedad semántica solía provocar errores o resultados imprecisos [10].

Con la consolidación de enfoques estadísticos y el auge del NLP, comenzaron a surgir sistemas de tipo *template-based* y, posteriormente, modelos de tipo *seq2seq*, que mejoraban la cobertura gracias a su capacidad de entrenamiento sobre datos reales [11]. No obstante, estos modelos seguían siendo sensibles a errores gramaticales y requerían grandes volúmenes de datos anotados para generalizar correctamente.

La aparición de conjuntos de datos como *ATIS*, *GeoQuery* y, posteriormente, *Spider*, supuso un punto de inflexión en el campo del *Text-to-SQL* [12]. Estos *datasets* permitieron evaluar y comparar distintos enfoques de forma estandarizada, acelerando el desarrollo de modelos más sofisticados. Inicialmente, muchos sistemas seguían empleando reglas o plantillas simples, con capacidades limitadas. Sin embargo, la introducción de mecanismos de atención y, más adelante, de arquitecturas basadas en *transformers*, permitió manejar consultas significativamente más complejas, incluyendo estructuras anidadas y expresiones SQL avanzadas [13].

En los últimos años, el desarrollo de los LLMs ha supuesto un cambio de paradigma. Modelos como *GPT-3* [14], *Codex* [15] o *GPT-4* han demostrado ser capaces de resolver tareas *zero-shot* o *few-shot*, generando consultas SQL válidas incluso sin entrenamiento específico para la tarea

[5] . Este avance ha desplazado el foco del diseño de arquitecturas al diseño de *prompts* y mecanismos de validación.

Diversos estudios han categorizado los enfoques actuales en cuatro grandes estrategias: *prompt engineering*, *fine-tuning*, uso de modelos preentrenados generalistas y soluciones tipo *Agent*, que combinan planificación e iteración. Cada enfoque presenta ventajas y limitaciones. Por ejemplo, el uso directo de LLMs mediante *prompting* reduce los costes de entrenamiento, pero implica riesgos de alucinación o errores sintácticos. En cambio, los modelos adaptados mediante *fine-tuning* ofrecen mayor precisión en dominios cerrados, aunque requieren un corpus representativo y un esfuerzo de ajuste considerable [16].

Actualmente, uno de los principales retos identificados en la literatura es el control de calidad sobre las salidas generadas. La necesidad de integrar validadores semánticos y mecanismos de reintento forma parte de una tendencia reciente hacia arquitecturas más robustas, capaces de detectar errores antes de ejecutar las consultas sobre bases de datos reales.

2.2.3. Normativas y estándares

La transformación de consultas en lenguaje natural a SQL implica no solo desafíos técnicos, sino también la necesidad de seguir marcos normativos y estándares que garanticen la calidad, interoperabilidad y fiabilidad del sistema. En el contexto del desarrollo de software orientado a sistemas inteligentes e interfaces conversacionales, existen varias normas técnicas relevantes, tanto en el ámbito de la ingeniería del software como en la interacción hombre-máquina y la gestión de calidad.

ISO/IEC/IEEE 29119 Esta serie de normas internacionales define un marco completo para los procesos de prueba de software. Se compone de varias partes, entre ellas:

- **Parte 1**: Conceptos generales (*General Concepts*) [17].
- **Parte 2**: Procesos de prueba (*Test Processes*) [3].
- **Parte 3**: Documentación de prueba (*Test Documentation*) [4].
- **Parte 4**: Técnicas de prueba (*Test Techniques*) [18].

Estas normas son aplicables a cualquier tipo de sistema software, incluidos los sistemas basados en modelos de lenguaje, y permiten establecer procesos sistemáticos para validar la funcionalidad, corregir errores y asegurar el cumplimiento de requisitos.

IEEE Std 1012-2016 La norma *IEEE Standard for System, Software, and Hardware Verification and Validation* proporciona un marco para verificar y validar que un sistema cumple con los requisitos especificados y satisface las necesidades del usuario [2]. En el caso de los sistemas basados en LLMs, esta norma resulta especialmente útil para definir criterios de aceptación de las salidas generadas, mediante pruebas funcionales y no funcionales.

ISO/IEC 25010:2023 La versión más reciente de este estándar, publicada en noviembre de 2023, redefine el modelo de calidad del producto software dentro del marco SQuaRE. Establece ocho características principales —como idoneidad funcional, eficiencia de rendimiento, compatibilidad, usabilidad, fiabilidad, seguridad, mantenibilidad y portabilidad— junto con sus subcaracterísticas asociadas. Este modelo proporciona una base para evaluar propiedades críticas del sistema desarrollado, incluyendo la precisión del procesamiento semántico, la resiliencia ante fallos y la calidad de la experiencia de usuario en la interacción con la interfaz conversacional [19].

UNE-ISO 690:2024 Aunque no es un estándar técnico sobre software, esta norma es relevante para asegurar la correcta redacción y citación de fuentes bibliográficas en documentos académicos y científicos [20]. Se ha empleado en este trabajo como guía para estructurar las referencias utilizadas.

Cumplimiento ético y legal Aunque no son estándares técnicos formales, deben considerarse los principios de transparencia, explicabilidad y trazabilidad en el diseño de sistemas basados en Inteligencia Artificial (IA). Normativas como el *AI Act* de la Unión Europea [21] o las directrices éticas de la *IEEE Global Initiative on Ethics of Autonomous and Intelligent Systems* proponen recomendaciones aplicables a este tipo de herramientas. Estos principios adquieren especial importancia en dominios como la agricultura de precisión, donde las decisiones basadas en IA pueden tener un impacto económico o medioambiental.

2.3. Tecnologías existentes

Esta sección describe las tecnologías y herramientas relevantes ya existentes para el desarrollo del proyecto.

2.3.1. Tecnologías y arquitecturas

El desarrollo de un sistema de interpretación de consultas en lenguaje natural y generación automática de sentencias SQL requiere la integración de distintas tecnologías, cada una con un papel específico dentro de la arquitectura general. A continuación, se describen las más relevantes para este proyecto, señalando sus principales ventajas, limitaciones y ámbito de aplicación.

Bases de datos relacionales: PostgreSQL PostgreSQL [22] es un sistema de gestión de bases de datos relacional de código abierto, ampliamente utilizado en entornos de producción por su robustez, escalabilidad y conformidad con el estándar SQL. Soporta una amplia variedad de tipos de datos, índices avanzados y extensiones como *PostGIS* para tratamiento geoespacial. Su licencia permisiva y la gran comunidad de soporte lo convierten en una opción preferente frente a alternativas como MySQL o SQL Server. No obstante, el procesamiento de consultas complejas puede requerir optimización manual (*query tuning*) y un ajuste cuidadoso de parámetros para mantener un rendimiento óptimo.

Frameworks web backend: Flask Flask [23] es un *microframework* para Python diseñado para construir aplicaciones web ligeras y modulares. Destaca por su flexibilidad, facilidad de integración con bibliotecas externas y curva de aprendizaje reducida. En este proyecto, se utiliza para exponer una API Representational State Transfer (REST) que recibe consultas y devuelve resultados al usuario. Su carácter minimalista implica que funcionalidades avanzadas, como la autenticación o la gestión de sesiones, deban implementarse manualmente o mediante extensiones de terceros. Aunque estas características no son necesarias en la fase actual, una posible ampliación del sistema podría requerir migrar a un *framework* más completo.

API de modelo de lenguaje: OpenAI GPT-4o-mini El LLM utilizado en este proyecto se consume a través de una API externa, lo que evita la necesidad de infraestructura de cómputo local. GPT-4o-mini, ofrecido por OpenAI, destaca por su amplio contexto, bajo coste y alta capacidad de comprensión semántica. Esta versión en particular crea un balance idóneo entre capacidad, coste y velocidad de respuesta [24]. Entre sus limitaciones se incluyen el riesgo de alucinaciones, la dependencia de un proveedor externo y la variabilidad del coste en función del uso.

Arquitecturas de procesamiento NLQ $\rightarrow$ SQL Existen distintos enfoques para transformar una NLQ en una sentencia SQL. Los modelos *direct-to-SQL* generan la consulta de forma directa a partir de la entrada del usuario, mientras que las arquitecturas en dos fases separan la extracción de intención semántica de la generación de la consulta. Este proyecto adopta una arquitectura en dos fases, donde un primer componente extrae la intención y un segundo componente genera la SQL correspondiente, ambos asistidos por modelos de lenguaje. El uso de *frameworks* como LangChain [25] o LangGraph [26] facilita la orquestación de estos pasos, así como la integración de validadores automáticos.

Herramientas de validación y testing En este proyecto, la validación de las consultas generadas incluye un primer control que garantiza que la sentencia producida sea estrictamente una `SELECT`, descartando automáticamente cualquier operación potencialmente destructiva o de modificación de datos. Una vez superado este filtro, se utiliza la instrucción `EXPLAIN` para comprobar la validez sintáctica y analizar el plan de ejecución sin alterar el contenido de la base de datos. Este mecanismo actúa como un doble nivel de protección, permitiendo detectar errores antes de la ejecución real. Aunque no se dispone de un entorno *sandbox* ni de un validador semántico completo, esta comprobación resulta suficiente para evitar la mayoría de fallos sintácticos y operaciones no intencionadas.

2.3.2. Comparativa

A continuación se presentan tablas comparando las soluciones/herramientas utilizadas en este proyecto y sus competidores más relevantes.

Enfoque / Solución	**Tipo de entrada soportada**	**Requerimientos de entrenamiento**	**Adaptabilidad a nuevos dominios**	**Limitaciones principales**	**Ejemplos representativos**
Rule-based	Lenguaje natural controlado	Ninguno (reglas predefinidas)	Muy baja	Rigidez ante variaciones y ambigüedades	LUNAR [8], ASK [9]
Template-based	Lenguaje natural simple con patrones fijos	Bajo	Baja	Cobertura limitada, difícil mantener escalabilidad	SQL Templates, ATIS inicial [27]
Seq2seq	Lenguaje natural moderadamente complejo	Alto (dataset anotado)	Media	Sensible a errores gramaticales, requiere mucho entrenamiento	ATIS avanzado, GeoQuery [28]
Transformers	Lenguaje natural complejo, consultas anidadas	Medio-Alto	Alta	Necesidad de ajuste fino para dominio específico	BERT-to-SQL [29], T5-to-SQL [30]
LLM + prompting	Lenguaje natural libre, incluso consultas ambiguas	Ninguno (uso directo)	Muy alta	Alucinaciones, dependencia de proveedor	GPT-4o mini, Codex [28]

Tabla 2.1: Comparativa funcional de enfoques para la traducción NLQ → SQL

Framework	**Facilidad de uso**	**Eficiencia**	**Coste despliegue**	**Completitud**
Flask [23]	Alta (mínima curva)	Buena (poco *overhead*)	Bajo (configuración simple)	Bajo–Medio (requiere extensiones)
Django [31]	Media (muchos conceptos iniciales)	Buena (*overhead* moderado)	Medio (más piezas a configurar)	Alto (incluye ORM, auth, etc.)
FastAPI [32]	Alta (declarativa, tipado)	Muy buena (ASGI, *async*)	Bajo (despliegue simple)	Medio–Alto (soporte nativo a API complejas)

Tabla 2.2: Comparativa funcional resumida de *frameworks* web backend

Como se explicó anteriormente, para el tamaño actual del proyecto, *Flask* es el *framework* más sencillo y manejable. No obstante, se valoraría migrar a *Django* o *FastAPI* si se requiriese un ecosistema integrado o capacidades avanzadas de concurrencia y escalabilidad.

Modelo	**Capacidad semántica**	**Coste**	**Tiempo de respuesta**	**Adecuación al proyecto**
GPT-4o-mini [24]	Alta (contexto amplio)	Muy bajo	Muy rápido	Muy alta (equilibrio coste/eficiencia)
GPT-4o [24]	Muy alta	Bajo–Medio	Rápido	Alta (más preciso, coste algo mayor)
GPT-5 [33]	Excelente (máxima precisión)	Alto	Lento	Media (no usado por latencia y coste)
LLaMA 3 70B [34]	Alta	Variable (adquisición de infraestructura)	Variable (depende de HW)	Media (requiere infraestructura propia)
Claude 3.5 Sonnet [35]	Muy alta (especialmente en razonamiento)	Medio	Rápido	Alta (buena alternativa a GPT-4o)

Tabla 2.3: Comparativa funcional resumida de modelos de lenguaje

En la fase actual, *GPT-4o-mini* es el modelo más adecuado por su bajo coste, alta capacidad semántica y rapidez de respuesta. Modelos como *GPT-5* podrían ofrecer mayor precisión, pero no se emplean debido a su mayor latencia y coste. Las opciones *open-source* como *LLaMA 3* aportan independencia de proveedor, aunque requieren infraestructura de cómputo propia.

SGBD	Rendimiento	Escalabilidad	Coste	Adecuación al proyecto
PostgreSQL [22]	Muy alto (consultas complejas optimizadas)	Alta (particionado, réplicas)	Nulo (*open-source*)	Muy alta (soporte completo SQL, extensiones como PostGIS)
MySQL [36] / *MariaDB* [37]	Alto (rápido en lecturas simples)	Alta (réplicas; sharding limitado)	Nulo	Media (menos eficiente en SQL complejo)
SQLite [38]	Medio (bajo volumen de datos)	Muy baja (sin concurrencia masiva)	Nulo	Baja (solo útil en prototipos locales)
SQL Server [39]	Alto	Alta	Alto (licencia propietaria)	Media–Baja (coste y sobrecapacidad)
MongoDB [40]	Alto (JSON flexible)	Alta	Bajo–Medio	Baja (NoSQL; no optimizado para SQL)

Tabla 2.4: Comparativa funcional de sistemas de gestión de bases de datos

Como se explicó anteriormente, para el tamaño y requisitos actuales del proyecto, *PostgreSQL* es la opción más adecuada por su soporte completo del estándar SQL, robustez y extensibilidad. Alternativas como *MySQL* o *SQLite* ofrecen simplicidad, pero carecen de la misma capacidad para manejar consultas complejas y tipos de datos avanzados, mientras que opciones propietarias como *SQL Server* incrementan costes sin aportar beneficios significativos en este contexto.

2.3.3. Casos de uso

En esta sección se muestran casos de uso que validan el correcto funcionamiento de las tecnologías utilizadas.

Modelos de lenguaje para NLQ → SQL Aplicaciones como *Microsoft Copilot* y *OpenAI Codex*-integrated IDEs han demostrado que los modelos de lenguaje pueden generar consultas SQL precisas a partir de lenguaje natural en entornos de producción, reduciendo la carga de trabajo de analistas y desarrolladores [41]. Plataformas como *Dataherald* emplean LLMs para que usuarios no técnicos exploren bases de datos empresariales mediante consultas en lenguaje natural, combinando validación sintáctica y semántica [42].

Frameworks web backend Flask ha sido ampliamente adoptado para el desarrollo de servicios web ligeros y API REST en entornos productivos. Un ejemplo destacado es su uso en Netflix, donde la plataforma interna *Scriptflask* (“Automation as a Service”) se construyó con

Flask para orquestar automatizaciones, y el propio equipo confirma que Flask es empleado en múltiples aplicaciones Python dentro de la compañía [43]. Además, Netflix documenta productos internos como *Winston* y *Bolt* desarrollados con Gunicorn + Flask + Flask-RESTPlus, evidenciando su uso operativo en entornos reales [44].

Bases de datos relacionales PostgreSQL es el motor elegido por plataformas como *Reddit* e *Instagram* por su robustez, extensiones avanzadas y escalabilidad [45] [46]. En entornos con alta carga transaccional y requisitos de integridad estrictos, empresas como *Fujitsu* emplean PostgreSQL en sistemas críticos, aprovechando su escalabilidad, robustez, compatibilidad con el estándar SQL y capacidades mejoradas, como cifrado transparente e in-memory column store [47].

2.4. Análisis crítico y justificación de la elección tecnológica

La solución propuesta se selecciona por su capacidad para reducir de forma efectiva la barrera entre usuarios no técnicos y las bases de datos. A continuación se justifican las decisiones en relación con los objetivos del proyecto y con criterios técnicos objetivos.

Rendimiento y eficiencia esperados El uso de un LLM de tamaño moderado permite tiempos de respuesta adecuados para interacción conversacional, manteniendo suficiente capacidad semántica para manejar consultas con errores ortográficos, frases desestructuradas o ambigüedad. La inclusión del esquema de la base de datos y de instrucciones claras en el *prompt* ayuda a guiar la generación y a limitar errores. La validación previa mediante `EXPLAIN` evita ejecutar consultas incorrectas, reduciendo tiempos muertos y posibles bloqueos.

Costes de desarrollo e implantación Consumir el LLM vía API elimina la necesidad de infraestructura de cómputo específica y de *MLOps* (entrenamiento, despliegue y escalado del modelo), reduciendo el coste inicial y el *time-to-market*. La arquitectura modular (servicio web ligero, orquestación con *LangGraph*, adaptadores de base de datos) acota el esfuerzo de desarrollo a componentes de valor (orquestación, validación, manejo de errores), evitando costes de *fine-tuning* y de curación de datos. El coste operativo se asocia exclusivamente al consumo de tokens: el modelo GPT-4o mini utilizado tiene un precio de 0,15$ por millón de tokens de entrada y 0,60$ por millón de tokens de salida [48]. Con la magnitud de tokens generada en las pruebas unitarias o en usos aislados, el gasto real se limita a pocos céntimos en la totalidad del proyecto, manteniendo la viabilidad económica incluso en fases de validación prolongada.

Compatibilidad con sistemas existentes El sistema expone una API REST que desacopla la interfaz conversacional del motor de datos, lo que facilita su integración con aplicaciones ya desplegadas. La generación SQL se ajusta al estándar y opera sobre *PostgreSQL* sin requerir extensiones propietarias. Los conectores hacia la base de datos y los *drivers* de cliente se mantienen estándar; el esquema (tablas, columnas, tipos) se inyecta como contexto,

permitiendo adaptar el sistema a distintas bases con cambios mínimos en configuración (no en código).

Facilidad de integración y mantenimiento El diseño por componentes (servicio web, orquestación LLM, validadores, ejecutor SQL) simplifica el mantenimiento: cada módulo es sustituible y testeable de forma independiente. La orquestación declarativa con *LangGraph* facilita introducir pasos de control (detección de intención, desambiguación, validación `EXPLAIN`, *retry/backoff*) sin afectar a las demás capas. El registro estructurado de *prompts*, respuestas y planes de ejecución permite pruebas de regresión y diagnóstico de errores. No se gestionan *runtimes* de inferencia ni GPUs, reduciendo la superficie operativa.

Cumplimiento de normativas y estándares El sistema transmite todas las peticiones al LLM y las respuestas a través de conexiones cifradas mediante Hypertext Transfer Protocol Secure (HTTPS) (Transport Layer Security (TLS)), garantizando la confidencialidad e integridad de los datos en tránsito. Aunque la base de datos utilizada no contiene información personal, esta práctica se considera un estándar de seguridad para prevenir accesos no autorizados y asegurar la autenticidad del servidor de destino. Además, la API de OpenAI opera sobre una infraestructura que registra las solicitudes y genera métricas internas, lo que permite monitorizar patrones de uso, detectar anomalías y contribuir a la trazabilidad del sistema. El diseño modular facilita documentar y auditar el flujo de datos y las decisiones del sistema, favoreciendo la alineación con buenas prácticas de transparencia y control en el uso de IA.

Síntesis frente a los objetivos del proyecto

- **Robustez ante errores y ambigüedad**: el LLM maneja variaciones lingüísticas; la orquestación añade desambiguación y validación `EXPLAIN` antes de ejecutar.
- **Tiempo de respuesta**: selección de modelo moderado y contexto acotado permiten interacción fluida.
- **Interfaz accesible**: API REST y capa conversacional que oculta SQL y el esquema al usuario final. Por tanto el usuario solo percibe una interfaz simple tipo chatbot que responde a sus peticiones.
- **Validación y métricas**: *pipeline* con validadores para asegurar que no haya fallos en el SQL generado y gestión de logs para medir cada respuesta proporcionada por el LLM.

En conclusión, el enfoque *middleware* con LLM satisface los requisitos de accesibilidad, robustez y operatividad con un coste y complejidad de implantación contenidos, manteniendo compatibilidad con sistemas existentes y un marco de control adecuado para su explotación en entornos reales.

3. Especificaciones técnicas y restricciones de diseño

3.1. Introducción

Este capítulo fija las especificaciones y las restricciones que condicionan el diseño de la solución. En primer lugar se enuncian las especificaciones del diseño (Sección 3.2), a continuación las restricciones de diseño (Sección 3.3) y, por último, la trazabilidad entre requisitos y componentes (Sección 3.4). Con ello se establece la base para las decisiones de arquitectura y la validación posterior.

3.2. Especificaciones del diseño

En esta sección se formulan los requisitos que debe cumplir la solución. Para facilitar su análisis y trazabilidad, se organizan en tres categorías: funcionales (Tabla 3.1), no funcionales (Tabla 3.2) y operativos (Tabla 3.3). Cada requisito se redacta de forma verificable y se identifica con un código único (RF–#, RNF–#, RO–#), manteniendo su trazabilidad con las pruebas y la matriz definida en la Sección 3.4.

Requisito funcional	**ID**
Recepción de consultas en lenguaje natural a través de la interfaz conversacional (chat).	RF-01
Traducción de la consulta a una sentencia SQL válida y coherente con el esquema activo.	RF-02
Validación previa no destructiva de la sentencia mediante `EXPLAIN`.	RF-03
Rechazo de operaciones de modificación de datos (`INSERT`, `UPDATE`, `DELETE`, etc.); sólo lectura.	RF-04
Ejecución de la consulta sobre *PostgreSQL*.	RF-05
Presentación del resultado y del estado de la operación en la interfaz conversacional en texto plano.	RF-06
Registro mínimo de actividad para evaluación (NLQ, SQL generada, estado de validación, marca temporal).	RF-07

Tabla 3.1: Requisitos funcionales del sistema

Requisito no funcional	**ID**
Latencia media por petición $\leq$ 5,0 s y p95 $\leq$ 10,0 s.	RNF-01
Precisión sintáctica: $\geq$ 90 % de las sentencias generadas son aceptadas por PostgreSQL sin error.	RNF-02
Seguridad de ejecución: 0 consultas destructivas (DML/DDL) ejecutadas en la batería de pruebas.	RNF-03
Mensajes de error comprensibles (causa y acción sugerida) en la interfaz conversacional.	RNF-04
Mantenibilidad: estructura modular en capas (API, orquestación LLM, validador, ejecutor SQL) con documentación breve.	RNF-05
Portabilidad y despliegue: instalación y ejecución reproducibles en un entorno con Python 3.11+.	RNF-06
Observabilidad: registro local por petición (NLQ, SQL, resultado de `EXPLAIN`, latencia, marca temporal).	RNF-07
Privacidad y seguridad: no incluir datos personales en los *prompts*; llamadas al proveedor LLM por HTTPS (TLS).	RNF-08
Coste: coste por petición en pruebas $<$ 0,01 € dadas las tarifas (0,15 $/M tokens entrada; 0,60 $/M tokens salida).	RNF-09

Tabla 3.2: Requisitos no funcionales del sistema

Requisito operativo	**ID**
Entorno con Python 3.11+ y acceso a una instancia de *PostgreSQL* alcanzable por red (local o remota).	RO-01
Conectividad saliente por HTTPS (TLS) hacia el proveedor del LLM.	RO-02
Configuración mediante variables de entorno o fichero `.env` (p. ej., clave de API, cadena de conexión, modelo, *timeouts*).	RO-03

Tabla 3.3: Requisitos operativos del sistema

3.3. Restricciones de diseño

Las restricciones de diseño son limitaciones que condicionan las decisiones técnicas, económicas o normativas del proyecto. En la Tabla 3.4 se presentan las restricciones identificadas, clasificadas según su naturaleza.

Restricción	Tipo
El consumo mensual en llamadas a la API de OpenAI no debe superar los 2 € en la fase de pruebas.	Económica
La solución debe funcionar en entornos Windows y ser compatible con Python 3.11 o superior.	Técnica
Todas las comunicaciones con el proveedor LLM deben realizarse mediante HTTPS (TLS).	Normativa
No se deben procesar datos personales en los prompts enviados al LLM.	Normativa
El sistema requiere conexión a Internet estable para la comunicación con la API del LLM.	Técnica

Tabla 3.4: Restricciones de diseño del sistema

3.4. Trazabilidad de requisitos

La trazabilidad establece la correspondencia entre cada requisito y el módulo o componente del sistema donde se implementa, garantizando la coherencia entre objetivos, especificaciones y diseño.

La trazabilidad se presenta en tres tablas: requisitos funcionales (Tabla 3.5), requisitos no funcionales (Tabla 3.6) y requisitos operativos (Tabla 3.7).

Requisito	ID	Módulo asociado
Recepción de consultas en lenguaje natural a través de la interfaz conversacional (chat).	RF-01	Interfaz conversacional
Traducción de la consulta a una sentencia SQL válida y coherente con el esquema activo.	RF-02	Generador SQL
Validación previa no destructiva de la sentencia mediante `EXPLAIN`.	RF-03	Validador SQL
Rechazo de operaciones de modificación de datos (`INSERT, UPDATE, DELETE`, etc); sólo lectura.	RF-04	Validador SQL
Ejecución de la consulta sobre *PostgreSQL* en modo lectura.	RF-05	Ejecutor SQL
Presentación del resultado y del estado de la operación en la interfaz conversacional en texto plano.	RF-06	Interfaz conversacional
Registro mínimo de actividad para evaluación (NLQ, SQL generada, estado de validación, marca temporal).	RF-07	Módulo de registro

Tabla 3.5: Trazabilidad de requisitos funcionales

Requisito	ID	Módulo asociado
Latencia media por petición $\leq$ 5,0 s y p95 $\leq$ 10,0 s.	RNF-01	API Flask + conexión API LLM
Precisión sintáctica: ≥ 90 % de las sentencias generadas son aceptadas por PostgreSQL sin error.	RNF-02	Generador SQL
Seguridad de ejecución: 0 consultas destructivas (DML/DDL) ejecutadas en la batería de pruebas.	RNF-03	Validador SQL
Mensajes de error comprensibles (causa y acción sugerida) en la interfaz conversacional.	RNF-04	Interfaz conversacional
Mantenibilidad: estructura modular en capas (API, orquestación LLM, validador, ejecutor SQL) con documentación breve.	RNF-05	Arquitectura modular
Portabilidad y despliegue: instalación y ejecución reproducibles en un entorno con Python 3.11+.	RNF-06	Infraestructura de ejecución
Observabilidad: registro local por petición (NLQ, SQL, resultado de `EXPLAIN`, latencia, marca temporal).	RNF-07	Módulo de registro
Privacidad y seguridad: no incluir datos personales en los prompts; llamadas al proveedor LLM por HTTPS (TLS).	RNF-08	API Flask + conexión API LLM
Coste: coste por petición en pruebas $<$ 0,01 € dadas las tarifas (0,15 $/M tokens entrada; 0,60 $/M tokens salida).	RNF-09	API Flask + conexión API LLM

Tabla 3.6: Trazabilidad de requisitos no funcionales

Requisito	ID	Módulo asociado
Entorno con Python 3.11+ y acceso a una instancia de *PostgreSQL* alcanzable por red (local o remota).	RO-01	Infraestructura de ejecución
Conectividad saliente por HTTPS (TLS) hacia el proveedor del LLM.	RO-02	Infraestructura de red
Configuración mediante variables de entorno o fichero .env (p. ej., clave de API, cadena de conexión, modelo, *timeouts*).	RO-03	Módulo de configuración

Tabla 3.7: Trazabilidad de requisitos operativos

3.5. Resumen del capítulo

En este capítulo se han definido las especificaciones que guían el desarrollo del proyecto y se han identificado las principales restricciones que condicionan las decisiones de diseño.

Estas consideraciones establecen el marco técnico dentro del cual se desarrollará la solución propuesta, garantizando su viabilidad y alineación con los objetivos definidos.

4. Descripción de la propuesta

4.1. Introducción

Este capítulo describe en detalle la solución técnica diseñada para transformar consultas en lenguaje natural en sentencias SQL ejecutables sobre *PostgreSQL*. La arquitectura se organiza como un grafo de orquestación construido con *LangGraph* [26] —que encadena los pasos de interpretación, generación, validación y ejecución— y se expone mediante una API ligera implementada con *Flask*.

En primer lugar se presenta una visión general de la propuesta y la justificación de las decisiones clave. A continuación, se detallan los componentes principales (interfaz conversacional, API, orquestación con *LangGraph*, validador `EXPLAIN`, ejecutor SQL y registro de actividad) y su integración. Por último, se incluyen los diagramas y algoritmos necesarios para comprender el flujo interno y se identifican las aportaciones originales frente a los elementos reutilizados.

4.2. Visión general de la solución

La solución propuesta consiste en un sistema *middleware* que permite a usuarios sin conocimientos técnicos interactuar con una base de datos sin necesidad de dominar el lenguaje de Bases de Datos (BBDD). El sistema se organiza en módulos independientes y escalables —implementados principalmente con Flask y LangGraph—, varios de ellos apoyados en LLMs, que transforman las consultas en lenguaje natural en sentencias SQL válidas. Antes de su ejecución, cada sentencia pasa por un proceso de validación para garantizar su seguridad y coherencia con el esquema activo. Todo el procesamiento se realiza en el *backend*, mientras que el usuario interactúa únicamente a través de una User Interface (UI) tipo *chatbot*, donde formula su consulta y recibe la respuesta en texto plano. La interfaz está diseñada para simular una conversación —el usuario pregunta y el sistema responde—, ofreciendo una experiencia amigable, cómoda y segura.

El desarrollo se apoya en un contexto real procedente del proyecto *AFarCloud* (*Aggregate Farming in the Cloud*, H2020–783221) [1], coordinado por el GRyS de la ETSIST–UPM [49]. *AFarCloud* desarrolló una plataforma distribuida para la agricultura autónoma, integrando sistemas ciber–físicos (sensores, drones, vehículos agrícolas) en tiempo real con software de gestión agrícola, con el objetivo de mejorar la productividad, la calidad de los productos y la eficiencia en el uso de recursos.

En este trabajo se emplea un subconjunto de los datos del proyecto, correspondiente a los registros obtenidos de collares inteligentes portados por animales. Este conjunto, denominado *Collar*, contiene 785 692 mediciones procedentes de 73 collares distintos, cada uno asociado a una serie temporal de valores que incluyen posición geográfica, temperatura y diferentes indicadores de anomalía. Cada registro se identifica de manera única mediante la combinación de un identificador de collar (`entityName`) y una marca temporal (`time`), permitiendo analizar

la actividad y las condiciones de cada animal a lo largo del tiempo.

La Tabla 4.1 resume la estructura del conjunto de datos empleado.

Collar	
Number of Entities	73 collar-type devices
Number of Points	785,692
Time	Timestamp (ISO 8601)
Dimensions	
Geoposition	Latitude, Longitude (Float), Geohash (String)
Semantic	Semantic information regarding data nature, identifiers, and service provider
Primary Key	`entityName` + `time`
Fields	19 attributes (Spatial and Semantic)
Integer	1
Float	7
Boolean	6
String	5

Tabla 4.1: Estructura del conjunto de datos *Collar*.

Este dataset no se analiza en profundidad, ya que su finalidad en este trabajo es servir como base realista para validar el sistema propuesto. Su estructura temporal y semántica resulta especialmente adecuada para evaluar la conversión de lenguaje natural a SQL, al representar un dominio técnico —la monitorización ganadera— en el que las consultas suelen ser complejas para usuarios sin conocimientos técnicos. Esta elección permite ilustrar el valor práctico del sistema, al facilitar que un ganadero pueda, por ejemplo, preguntar en lenguaje natural *«¿Qué collar tiene alarmas?»* sin necesidad de comprender ni escribir una sentencia SQL.

Dado que el proyecto *AFarCloud* tiene alcance europeo, los datos y la documentación asociada se encuentran principalmente en inglés, lo que además refuerza la interoperabilidad y la capacidad del sistema para adaptarse a distintos contextos multilingües.

Desde el punto de vista metodológico, el desarrollo sigue un enfoque modular y distribuido, alineado con la arquitectura cliente–servidor. Esta estructura facilita la escalabilidad, el mantenimiento y la integración de nuevos módulos, a la vez que permite la evolución del sistema sin interrumpir su funcionamiento. Además, este diseño restringe el uso de los LLMs a tareas muy específicas —como la extracción de intención o la conversión a SQL—, minimizando así la generación de alucinaciones y errores semánticos característicos de modelos de contexto amplio. Esto implica que, en caso de querer escalar el proyecto con nuevas funcionalidades (e.g., gestión de usuarios), solo sería necesario implementar nuevos módulos y añadirlos al sistema. En la Figura 4.1 se presenta un diagrama de flujo del sistema completo por módulos.

4.2.1. Comunicación con el agente LLM

La interacción con el agente LLM se realiza mediante *prompts* de dos tipos bien diferenciados. El **system prompt** fija el comportamiento global del modelo (tarea a realizar), el formato de salida requerido y las restricciones de seguridad —por ejemplo, producir únicamente una sentencia `SELECT` y no generar operaciones *DML/DDL*—, favoreciendo respuestas consistentes y reduciendo alucinaciones. El **user prompt** aporta el *contexto variable* de cada petición: la NLQ del usuario y el *esquema* de la base de datos; en la segunda fase incluye además la *intención* previamente extraída. Con esta separación, el sistema limita el uso del LLM a tareas acotadas (interpretación y generación) y mantiene controlado el flujo de información. Los detalles operativos de esta interacción se desarrollan en las secciones siguientes.

4.3. Componentes principales del sistema

El sistema propuesto se organiza en una serie de módulos claramente diferenciados, cuya interacción permite transformar una consulta en lenguaje natural en una sentencia SQL válida y devolver al usuario únicamente el dato solicitado, presentado en una interfaz cómoda. A continuación se describen los principales componentes y su integración dentro de la arquitectura. En la Figura 4.2 se muestra la secuencia completa de las interacciones entre los distintos módulos del sistema.

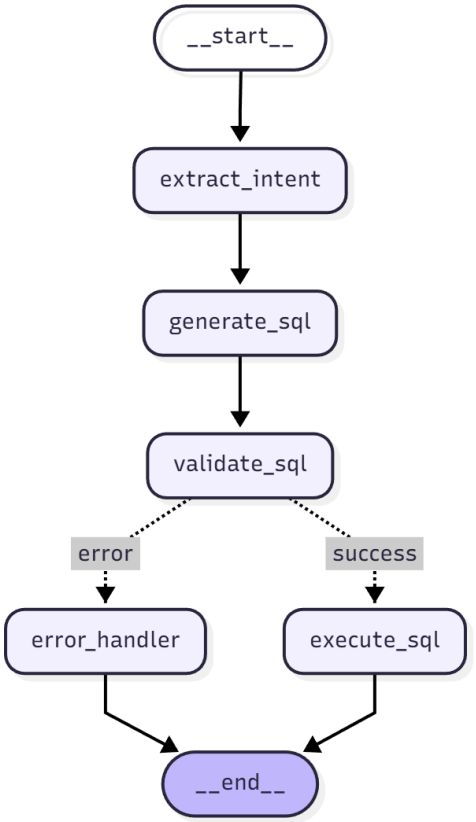

Figura 4.1: Diagrama de flujo del sistema por módulos.

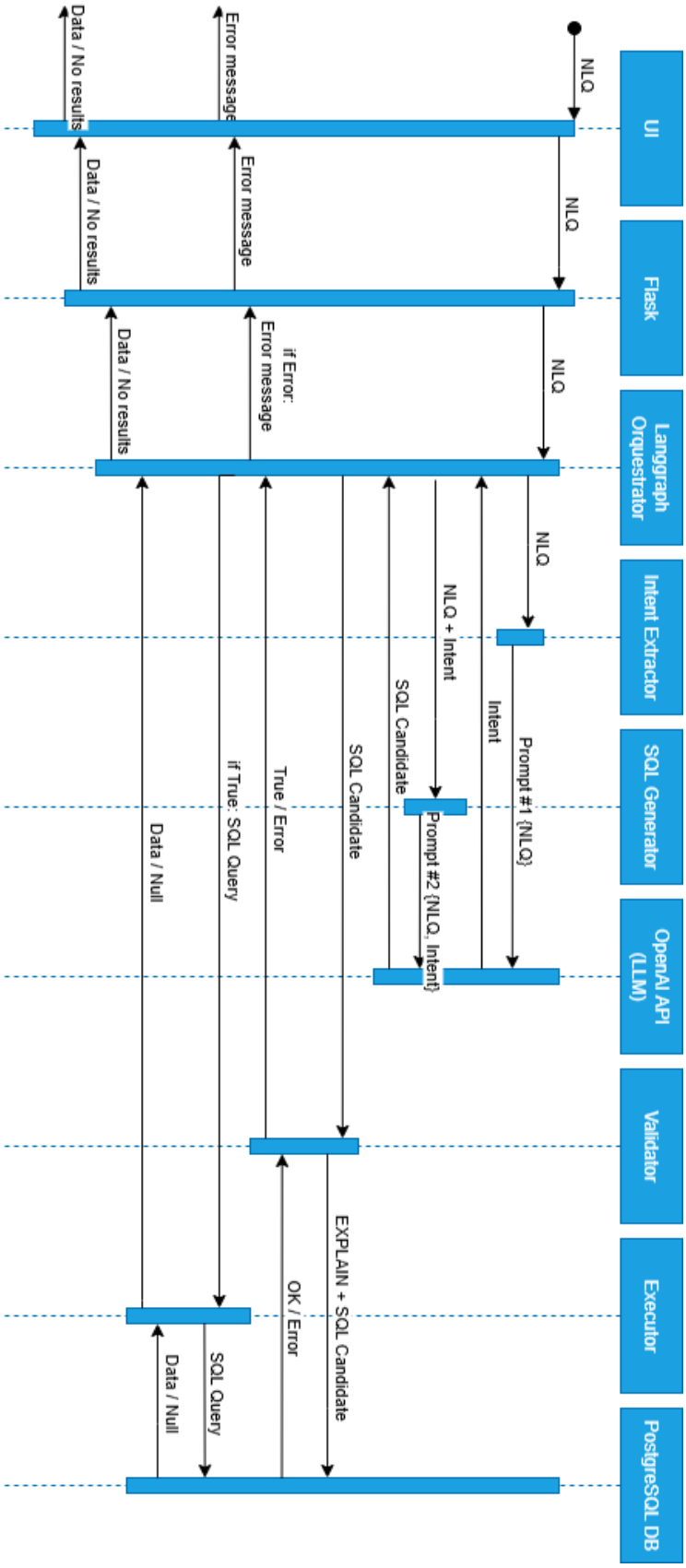

Figura 4.2: Diagrama secuencial por módulos.

4.3.1. Interfaz de usuario

La interacción con el sistema se realiza mediante una interfaz web estilo *chatbot*, desarrollada con HTML, CSS y JavaScript, que conecta con el *backend* a través de la API REST de Flask. Esta interfaz permite que el usuario formule su consulta en lenguaje natural a través de un campo de texto mediante escritura por teclado; el sistema recibe esa consulta y realiza los procesos necesarios para obtener el resultado. Posteriormente, el usuario recibe como respuesta el valor extraído de la base de datos, presentado en formato limpio a partir del texto recibido. La elección de un diseño conversacional se basa en otros modelos conversacionales como ChatGPT y es acorde al proyecto por su accesibilidad y facilidad de uso frente a interfaces más complejas. A continuación se pueden ver dos imágenes de cómo se muestra la interfaz al usuario en las Figuras 4.3 y 4.4.

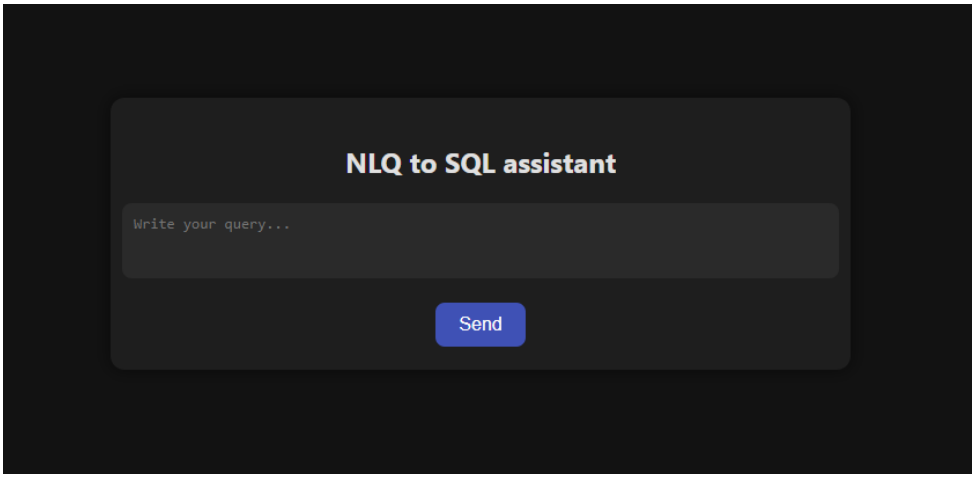

Figura 4.3: Interfaz antes de recibir la primera consulta.

4.3.2. Módulo de extracción de intención

El objetivo de este componente es interpretar la consulta en lenguaje natural introducida por el usuario y transformarla en una representación semántica explícita denominada *intent*. Esta intención describe, de manera estructurada, la acción que el usuario desea realizar sobre la base de datos (por ejemplo, «Retrieve the location data (latitude and longitude) for the collar with the entity name 'AH277'.» o «Count the cows with activity anomalies.»).

Técnicamente, el módulo se implementa mediante un LLM, al que se le proporciona un *prompt* (véase I.1) diseñado para guiar la extracción de información relevante sin generar directamente una sentencia SQL. Dicho *prompt* incluye: el esquema de la base de datos, la consulta en lenguaje natural del usuario y las instrucciones. El resultado se devuelve en formato de cadena, conteniendo únicamente el texto de la intención.

Este diseño aporta una capa intermedia que reduce la ambigüedad inherente al lenguaje natural y facilita la posterior generación de consultas SQL consistentes por parte del LLM. Se valoraron alternativas como la traducción directa de NLQs a SQL, pero se descartaron por su mayor propensión a errores de sintaxis, menor control sobre la interpretación semántica y mayor posibilidad de fallos por ambigüedad. En cambio, introducir un paso explícito de obtención del *intent* mejora la trazabilidad y la capacidad de validación del flujo completo.

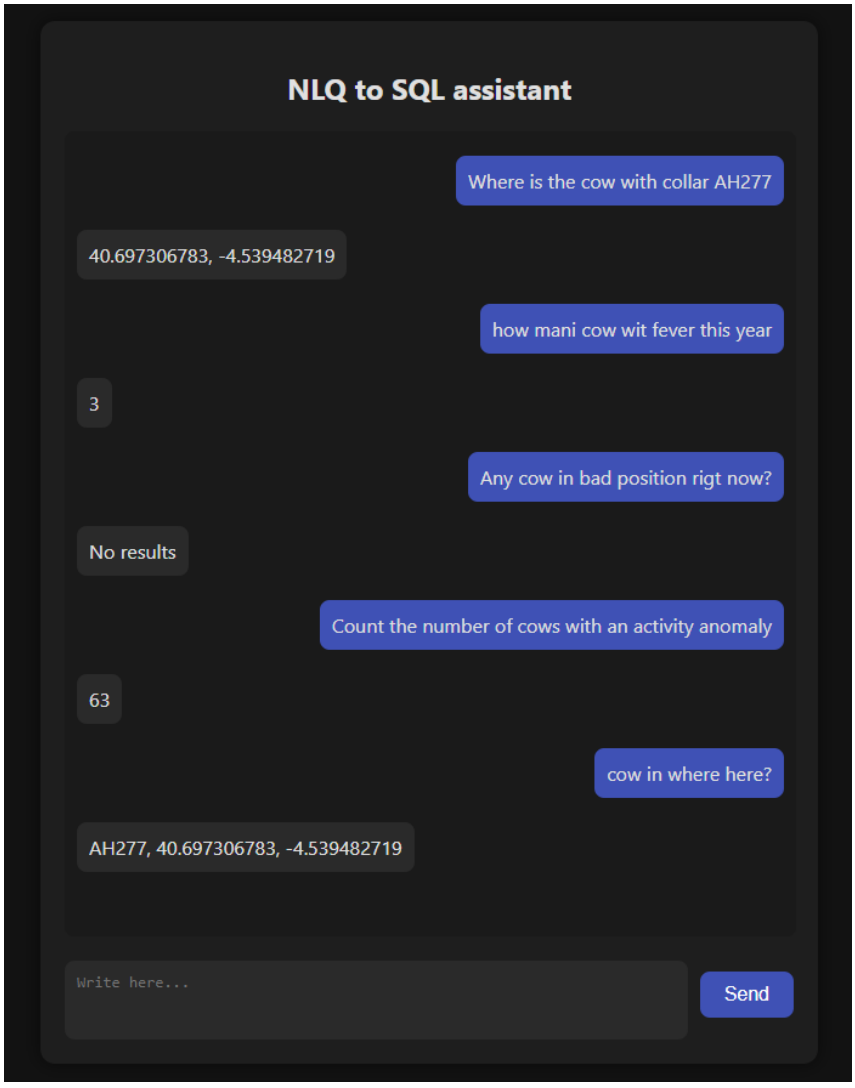

Figura 4.4: Interfaz después de recibir varias consultas.

A continuación, en la Tabla 4.2 se presentan ejemplos reales de la intención obtenida a partir de *queries* en distintos grados de complejidad:

Consulta NLQ recibida	**Intent obtenido**
Where is the cow with collar AH277	Find the location of the cow with collar ID AH277.
how mani cow wit fever this year	Count the number of cows with fever this year.
Any cow in bad position rigt now?	Retrieve records of cows currently experiencing a position anomaly.
Count the number of cows with an activity anomaly	Count the cows with activity anomalies.
cow in where here?	Retrieve the location data of the cows.

Tabla 4.2: Ejemplo de correspondencia entre consulta en lenguaje natural e intención extraída.

Como se puede observar en los resultados obtenidos, la extracción de la intención se logra de forma fidedigna al dedicar el LLM exclusivamente a esta tarea. Los resultados muestran un alto grado de precisión incluso en consultas con faltas de ortografía o formulaciones ambiguas, donde el modelo infiere la intención más probable de manera consistente.

4.3.3. Módulo de generación de SQL

Este módulo es similar en cuanto a funcionamiento al de extracción de intención, aunque en este caso la tarea consiste en obtener una sentencia SQL candidata a partir de un *prompt* procesado por el LLM mediante LangChain. Dicho *prompt* (véase I.2) incluye el esquema de la base de datos, la consulta en lenguaje natural, el *intent* previamente extraído y unas instrucciones explícitas que obligan a devolver únicamente la sentencia SQL, evitando así la necesidad de un posprocesado adicional.

La precisión en este paso es crucial: una sentencia que solicite datos que no son los que han sido requeridos podría conducir a resultados erróneos para el usuario. Para minimizar este riesgo, el *prompt* está diseñado de forma muy restrictiva, enfatizando el esquema y las reglas de generación, lo que reduce la ambigüedad y mejora la coherencia de la consulta producida.

En la Tabla 4.3 se muestran ejemplos de consultas en SQL generadas a partir de distintas NLQs, incluyendo casos con errores ortográficos o formulaciones ambiguas. Se utilizan las mismas NLQs que en la Tabla 4.2.

Como se observa, el sistema logra producir sentencias SQL válidas incluso ante entradas con errores ortográficos o formulaciones incompletas. Sin embargo, en casos muy ambiguos (como la última consulta de la tabla) el modelo puede proponer una interpretación razonable pero no necesariamente correcta, lo que justifica la necesidad de un módulo adicional de validación y verificación antes de la ejecución sobre la BD.

Consulta NLQ recibida	**SQL generado**
Where is the cow with collar AH277	SELECT ”latitude”, ”longitude” FROM collar WHERE ”entityName” = ’AH277’ AND ”time” = (SELECT MAX(”time”) FROM collar WHERE ”entityName” = ’AH277’ AND ”time”::date = ’2021-07-28’);
how mani cow wit fever this year	SELECT COUNT(DISTINCT ”entityName”) FROM collar WHERE ”time” >= ’2021-01-01’ AND ”time” <’2022-01-01’ AND ”temperatureAnomaly” = TRUE;
Any cow in bad position rigt now?	SELECT DISTINCT ”entityName” FROM collar WHERE ”positionAnomaly” = TRUE AND ”time” = (SELECT MAX(”time”) FROM collar WHERE ”time”::DATE = ’2021-07-28’ AND ”type” = ’collar’);
Count the number of cows with an activity anomaly	SELECT COUNT(DISTINCT ”entityName”) FROM collar WHERE ”activityAnomaly” = TRUE;
cow in where here?	SELECT ”entityName”, latitude, longitude FROM collar WHERE ”time” = (SELECT MAX(”time”) FROM collar WHERE ”time” <’2021-07-28T23:59:59Z’);

Tabla 4.3: Ejemplo de correspondencia entre consulta en lenguaje natural y SQL generado.

4.3.4. Módulo de validación

El objetivo de este componente es garantizar que la sentencia SQL candidata sea segura y coherente antes de ejecutarla sobre la BD. El proceso aplica dos controles secuenciales: (i) una política de *sólo lectura*, y (ii) una validación sintáctica mediante `EXPLAIN`. Únicamente si ambos pasos se superan, la consulta se pasa al módulo de ejecución.

La política de solo lectura restringe explícitamente la ejecución a sentencias `SELECT`. Como filtro práctico, el validador comprueba que la consulta no contenga palabras clave de *DML/DDL* (`INSERT`, `UPDATE`, `DELETE`, `ALTER`, `DROP`, `CREATE`, `TRUNCATE`, `GRANT`, `REVOKE`, etc.). Si se detecta cualquiera de estas, la petición se rechaza y se informa al usuario.

Para evitar errores de ejecución, la sentencia candidata se somete primero a `EXPLAIN`. Si `EXPLAIN` devuelve un error del motor, se considera consulta inválida y se comunica el motivo en la respuesta. Actualmente no se aplican reintentos automáticos, pero podría incorporarse un módulo para ello en caso de ampliación del proyecto.

4.3.5. Módulo de ejecución

Una vez validada la sentencia SQL, el módulo de ejecución la envía a la BD y obtiene la primera fila del resultado. La función expone una interfaz simple, coherente con el requisito de la herramienta de devolver únicamente el dato de interés.

La función devuelve una cadena en los siguientes casos:

- Si no hay filas: devuelve la cadena `"No results"`.
- Si existe una fila con una sola columna: devuelve ese valor convertido a texto.
- Si la primera fila contiene varias columnas: devuelve los valores concatenados por comas (CSV en una sola línea).

El módulo captura las excepciones de conexión y de ejecución, registrando los detalles en el sistema de *logging*. Al usuario únicamente se le entrega una salida neutra, de modo que nunca se exponen mensajes internos de la BD. Este diseño garantiza robustez y consistencia en la interacción con la herramienta.

4.4. Diagramas, algoritmos y pseudocódigo

En esta sección se representan de forma gráfica y simplificada los aspectos clave del funcionamiento interno de cada componente del sistema. Se adopta una **vista modular**, centrada en los bloques críticos que conforman el *backend*.

Para cada módulo se presentan dos elementos:

- Un diagrama de flujo que muestra de manera visual la secuencia de pasos internos.
- Un pseudocódigo simplificado que refleja las operaciones principales realizadas.

Este enfoque facilita comprender cómo trabaja cada parte de forma aislada antes de analizar en la Sección 4.5 la integración completa del sistema.

4.4.1. Vista modular

Módulo de extracción de intención

Figura 4.5: Diagrama del módulo de extracción de intención. *Listing 4.1*: Pseudocódigo del extractor de intención. En el Anexo I.1 se puede ver el *prompt* completo.

Módulo de generación de SQL

Figura 4.6: Diagrama del módulo de generación de SQL. *Listing 4.2*: Pseudocódigo del generador de SQL. En el Anexo I.2 se puede ver el *prompt* completo.

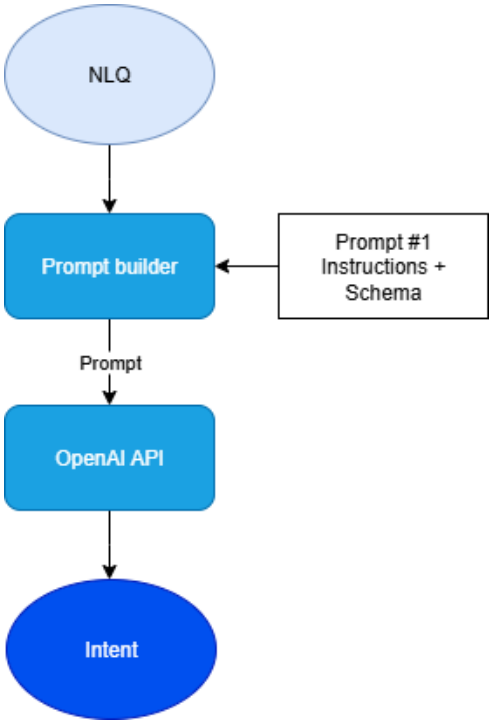

Figura 4.5: Flujo: obtención del *intent*.

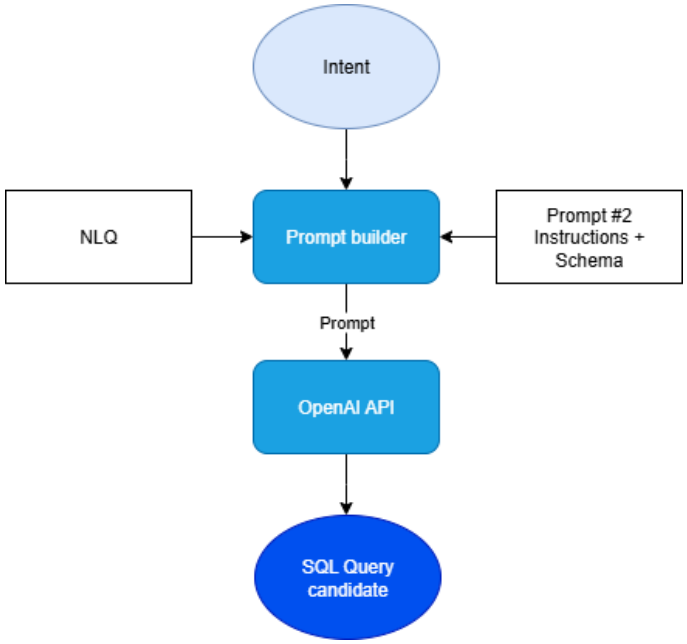

Figura 4.6: Flujo: generación de SQL.

```
procedure obtenerIntent(nlq)
  // Construcción del prompt
  intent_prompt = PromptTemplate(
                ["query"] <- nlq,
                template="intent_prompt.txt"
            )

  // Llamada a la API del modelo de lenguaje
  intent <- OpenAI_API.enviar(intent_prompt)
  return intent
end procedure
```

Listing 4.1: Módulo de extracción de intención

```
procedure generarSQL(nlq, intent)
  // Construcción del prompt
```

```
  sql_prompt = PromptTemplate(
                 ["query"] <- nlq,
                 ["intent"] <- intent,
                 template="sql_prompt.txt"
               )

  // Llamada a la API del modelo de lenguaje
  sql_query <- OpenAI_API.enviar(sql_prompt)
  return sql_query
end procedure
```

Listing 4.2: Módulo de generación de SQL

Módulo de validación

Figura 4.7: Diagrama de flujo del validador. Como se mencionó anteriormente, la validación se hace con dos comprobaciones: que la consulta sea de solo lectura y que no dé error al ejecutar con `EXPLAIN`. *Listing 4.3*: Pseudocódigo del validador.

Módulo de ejecución

Figura 4.8: Diagrama de flujo del ejecutor (transacción RO, *timeout*, primera fila). *Listing 4.4*: Pseudocódigo del ejecutor.

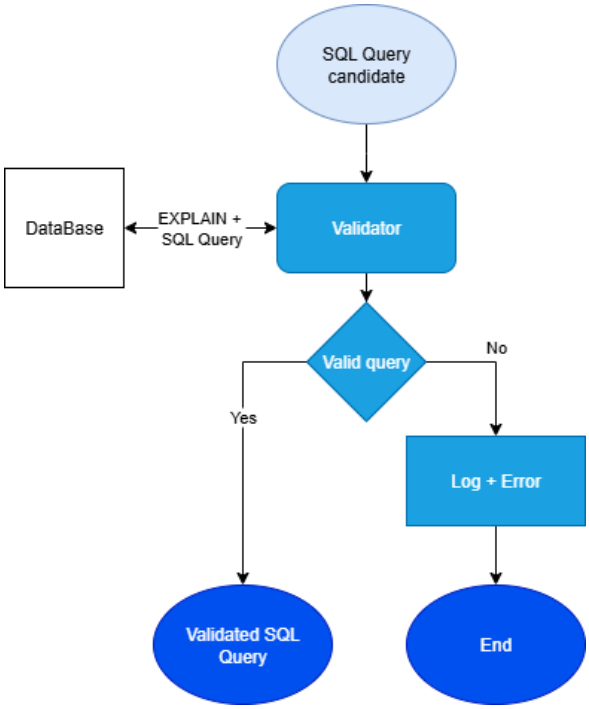

Figura 4.7: Flujo: validación de la consulta candidata.

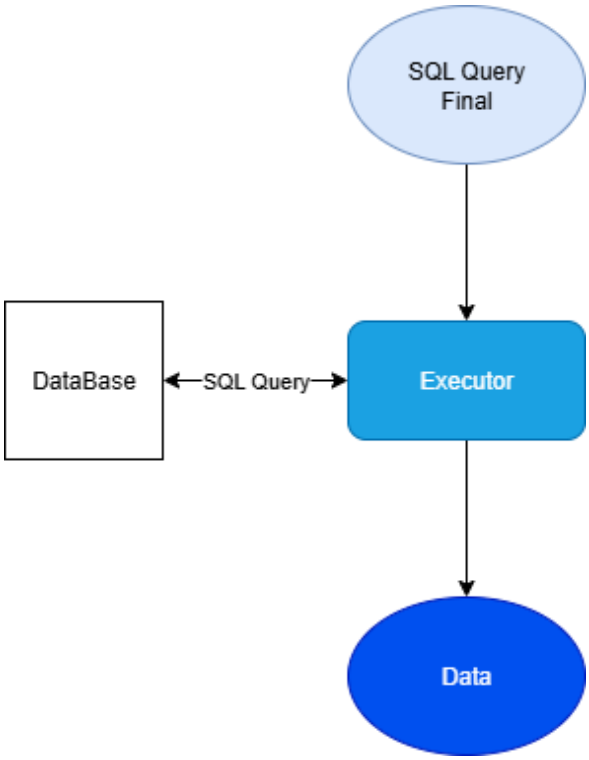

Figura 4.8: Flujo: ejecución de SQL y obtención de datos.

```
procedure validarSQL(sql_query)
  q <- sql_query.trim()

  // Solo se permiten consultas que empiecen por SELECT
  if not startsWith(q, "SELECT") then
      return error("Only SELECT queries are allowed.")
  end if

  // Validación sintáctica con EXPLAIN
  resultado <- DataBase.EXPLAIN(q)
  if resultado es válido then
      return q   // Devuelve la consulta validada
  else
      LogError(resultado)
      return error("Invalid SQL statement.")
  end if
end procedure
```

Listing 4.3: Módulo de validación de consulta SQL

```
procedure ejecutarSQL(sql_query)
  // Ejecutar la consulta ya validada
  resultados <- DataBase.execute(sql_query)

  if resultados.estaVacio() then
      return "No results"
  else
      // Si hay una sola columna, se devuelve su valor como texto.
      // Si hay varias, se concatenan con comas en una sola línea.
      return resultados
  end if
end procedure
```

Listing 4.4: Módulo de ejecución de consulta SQL

Interfaz de usuario (UI)

Este elemento no forma parte del *backend* pero se presenta para mostrar la simplicidad de uso del sistema para el usuario. *Figura 4.9*: Diagrama ligero de eventos entre la UI y el usuario.

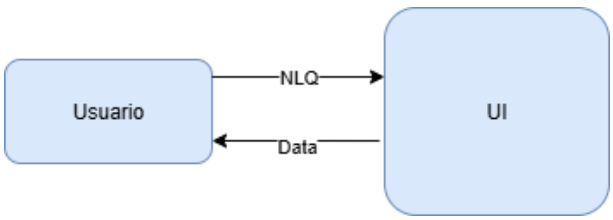

Figura 4.9: Interacción percibida por el usuario final.

Como se puede observar en este último diagrama, el usuario final únicamente introduce una consulta como entrada y recibe como salida los datos solicitados, sin elementos adicionales que distraigan del resultado.

4.5. Integración de componentes y flujo de funcionamiento

A continuación se ofrece una visión general del sistema que combina la arquitectura y el flujo de datos extremo a extremo. La Figura 4.10 resume el recorrido completo: el usuario introduce una NLQ en la UI; el orquestador encadena los módulos *Intent Extraction* (que produce la *intención*), *SQL Generation* (que genera la *SQL candidata*), *Validation* (que verifica que sea de solo lectura y sintácticamente válida) y, únicamente si supera esa etapa, *SQL Execution*, que consulta la BD y devuelve el dato a la interfaz. El diagrama destaca los artefactos intermedios (*Intent*, *SQL Candidate*, *SQL*) y las direcciones de las interacciones entre capas (UI, backend y BD).

De forma complementaria, la Figura 4.11 desglosa lo que ocurre dentro del *backend*, mostrando la secuencia de módulos y mensajes que transforman la NLQ en una consulta SQL válida y ejecutable (construcción de *prompts*, extracción de intención, generación de la consulta candidata, validación con `EXPLAIN` y ejecución).

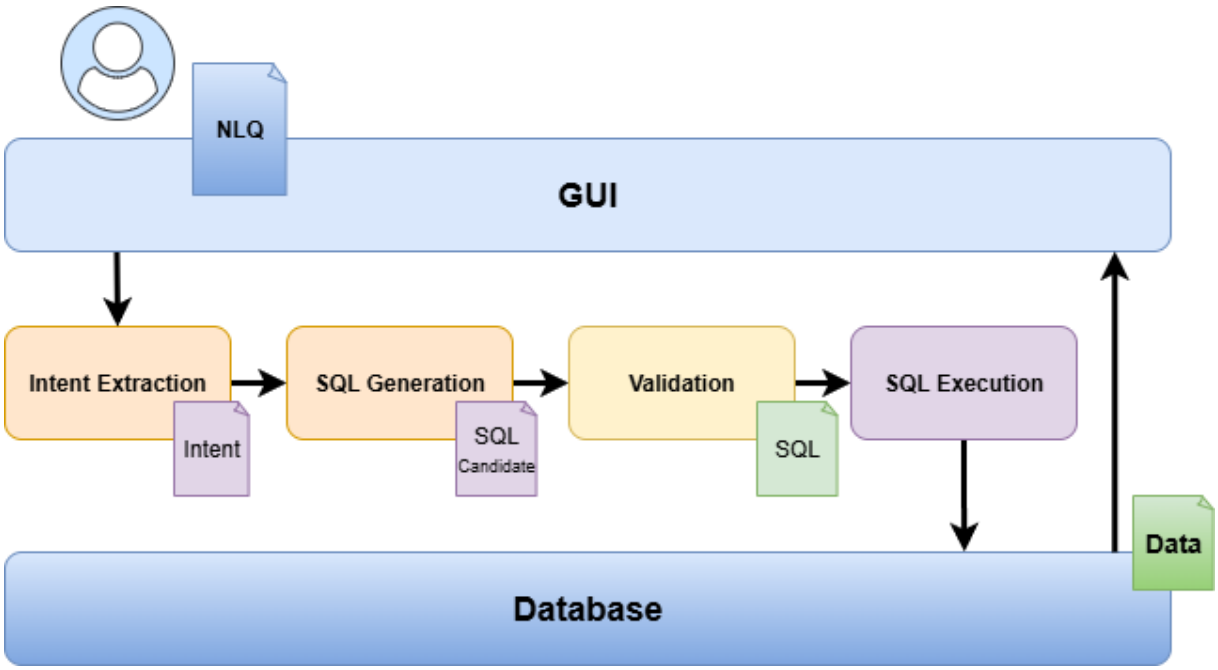

Figura 4.10: Diagrama de flujo de la arquitectura.

Cada módulo recibe y produce información bien definida, lo que permite mantener la modularidad y trazabilidad del flujo:

- **Extract Intent**: Entrada: NLQ + *Prompt 1* (véase I.1). Salida: intención semántica.
- **Generate SQL**: Entrada: NLQ + intención + *Prompt 2* (véase I.2). Salida: consulta SQL candidata.

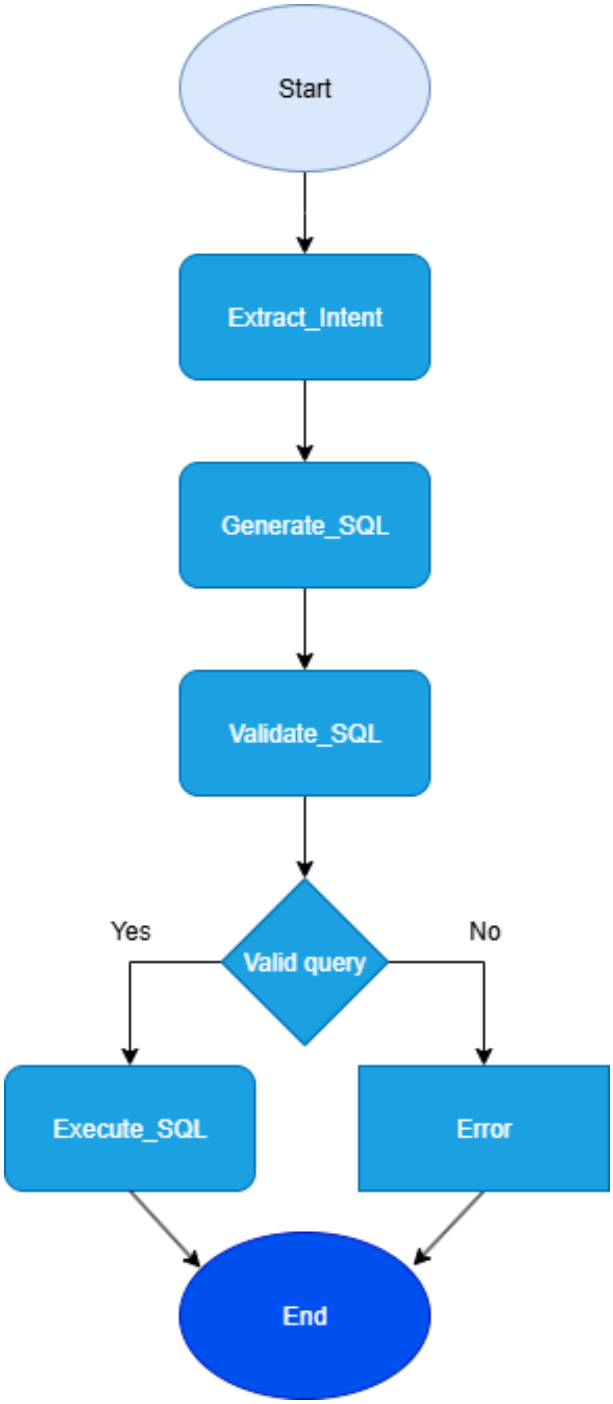

Figura 4.11: Flujo interno del *backend* con todos los módulos.

- **Validate SQL**: Entrada: consulta SQL candidata. Salida: consulta validada o error de PostgreSQL.
- **Execute SQL**: Entrada: consulta SQL validada. Salida: conjunto de resultados o mensaje "No results".

En la Figura 4.12 se presenta un pequeño diagrama de cada módulo para visualizar mejor el flujo de información.

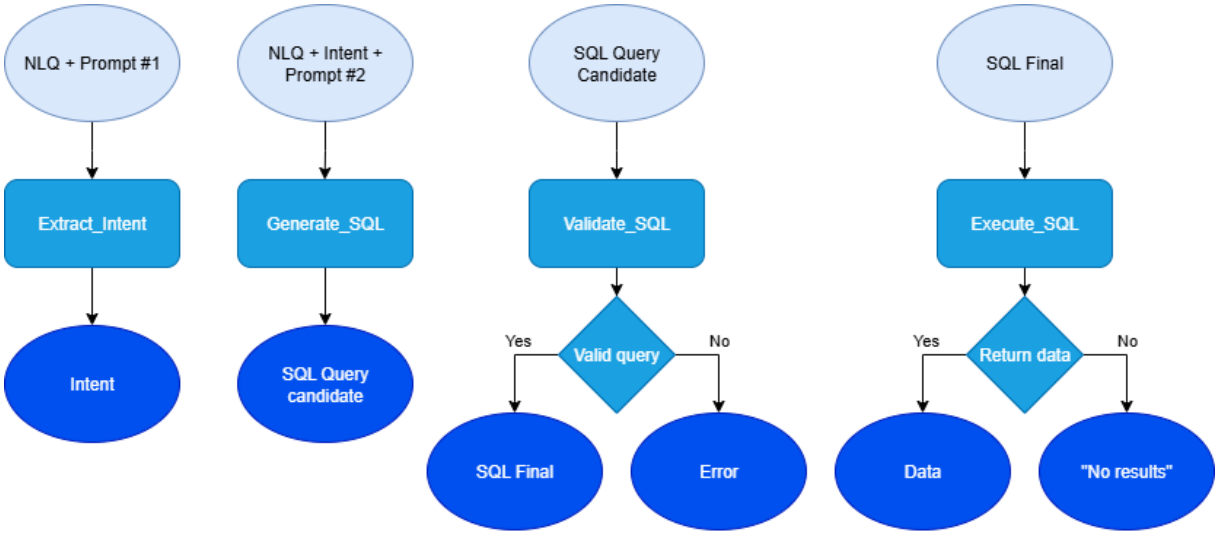

Figura 4.12: Flujo de información por módulos.

En la aplicación se contempla el control de errores principalmente en dos niveles:

- **Errores de infraestructura**: falta de conexión a Internet, caída del servidor o fallo en la comunicación con el modelo de lenguaje. En estos casos, el error real se muestra por consola para su depuración, mientras que el usuario recibe un mensaje genérico indicando que ocurrió un error interno.
- **Errores funcionales en la consulta**: por ejemplo, cuando el modelo genera una sentencia SQL inválida. En este caso no se ejecuta nada en la base de datos y el usuario recibe directamente el mensaje de error correspondiente.

En condiciones normales (conexión estable y sin fallos externos), el flujo siempre ejecuta los módulos *Extract Intent*, *Generate SQL* y *Validate SQL*. El módulo *Execute SQL* únicamente se ejecuta si la validación previa confirma que la sentencia es correcta.

4.6. Aportación original del estudiante

En este proyecto, la aportación original del estudiante se centra en el diseño, desarrollo e integración del sistema completo de interpretación de consultas en lenguaje natural a SQL. Concretamente, se han realizado las siguientes contribuciones propias:

- **Definición de la arquitectura**: diseño modular del sistema con separación de interfaz de usuario, *backend* y módulos especializados para extracción de intención, generación de sentencias SQL, validación y ejecución.
- **Construcción de *prompts***: elaboración de plantillas específicas y estructuradas para guiar al modelo de lenguaje en las tareas de extracción de intención y generación de SQL, ajustadas al esquema concreto de la base de datos empleada.
- **Implementación del *backend***: desarrollo en Python utilizando `Flask`, incluyendo la lógica de comunicación entre la interfaz, el motor de LangGraph y la base de datos PostgreSQL, así como el manejo de errores.
- **Validación y control de flujo**: diseño de mecanismos de validación de las sentencias SQL generadas y de control de errores, asegurando que solo se ejecuten consultas válidas en la base de datos.
- **Integración de módulos**: definición del flujo de interacción entre los distintos componentes y desarrollo del pseudocódigo y diagramas que explican su funcionamiento.

Por otro lado, se han empleado las siguientes herramientas y bibliotecas externas, justificadas por su idoneidad y solidez en la Sección 2.3:

- **Flask**: para la implementación del servidor web y la API de comunicación.
- **LangChain / LangGraph**: para la orquestación de flujos con modelos de lenguaje.
- **OpenAI API**: como motor de modelo de lenguaje, utilizado en la extracción de intención y generación de SQL.
- **PostgreSQL**: como sistema de gestión de base de datos para almacenar y consultar los datos.

La aportación del estudiante radica en cómo se han combinado, adaptado y personalizado estas herramientas para dar lugar a un sistema funcional y robusto. El uso de bibliotecas externas se justifica en tanto que proporcionan capacidades de bajo nivel (por ejemplo, el acceso al modelo de lenguaje o la conexión con la base de datos) que no son el objetivo principal de este proyecto, permitiendo centrar el esfuerzo en la propuesta original: la transformación semántica de lenguaje natural a SQL y la construcción de un flujo modular que lo haga accesible a usuarios no expertos.

4.7. Resumen del capítulo

En este capítulo se ha descrito en detalle la solución propuesta, comenzando por la definición de su arquitectura general y continuando con el análisis de cada uno de los módulos que la conforman. Se han presentado diagramas, algoritmos y pseudocódigo que ilustran el flujo de funcionamiento, desde la recepción de la consulta en lenguaje natural hasta la obtención de los resultados finales.

Asimismo, se han justificado las principales decisiones de diseño adoptadas, destacando la integración modular, el uso de técnicas de validación y el manejo de errores como elementos clave para garantizar la robustez del sistema. Finalmente, se ha puesto de relieve la aportación original del estudiante, tanto en la construcción de los *prompts* y la lógica de validación como en la definición del flujo de interacción entre los distintos componentes.

5. Resultados y validación

5.1. Introducción

Este capítulo presenta la evaluación del sistema desarrollado para la transformación de NLQs a sentencias SQL, siguiendo los principios de verificación y validación definidos en la norma *IEEE 1012-2016* [2] y las directrices de pruebas de la serie *ISO/IEC/IEEE 29119* [3].

En este contexto, se distinguen dos enfoques complementarios:

- **Verificación** ("¿hemos construido el sistema correctamente?"): comprueba que los módulos implementados —extracción de intención, generación de SQL, validación y ejecución— cumplen con las especificaciones técnicas definidas en el capítulo 3.
- **Validación** ("¿hemos construido el sistema correcto?"): evalúa si la solución satisface el propósito previsto, es decir, permitir a usuarios no técnicos acceder a la base de datos mediante consultas expresadas en lenguaje natural, con tiempos de respuesta aceptables y resultados correctos.

Las pruebas se diseñaron con dos objetivos principales: (1) verificar la correcta funcionalidad de cada módulo y del flujo completo NLQ $\rightarrow$ SQL $\rightarrow$ resultado, y (2) validar que el sistema es robusto y usable en escenarios realistas, donde los usuarios pueden formular consultas con distintos niveles de precisión o incluso erróneas.

Para ello, se definió un conjunto de cinco NLQs representativas del dominio, expresadas en tres niveles de dificultad lingüística. Cada combinación NLQ–nivel se ejecutó veinte veces con el fin de obtener métricas de rendimiento estables y evaluar la variabilidad en los tiempos de respuesta. Adicionalmente, se incluyó un conjunto de cinco consultas sin sentido o inválidas, destinadas a comprobar el comportamiento del validador.

Los resultados se analizan tanto desde la perspectiva funcional (exactitud en la traducción NLQ→SQL y corrección de las respuestas) como desde la no funcional (rendimiento, robustez y estabilidad). Se emplean métricas cuantitativas extraídas automáticamente de los registros de ejecución, junto con gráficas comparativas y boxplots que permiten caracterizar el sistema bajo diferentes condiciones de entrada. Finalmente, la validación cualitativa se realizó con el tutor del proyecto, actuando como usuario experto del dominio.

5.2. Pruebas funcionales

Las pruebas funcionales constituyen la base de la verificación del sistema, ya que permiten comprobar su capacidad para transformar NLQs de distinta complejidad en sentencias SQL correctas y, en última instancia, obtener los datos esperados de la base de datos.

El aspecto más relevante de esta evaluación es determinar si el sistema mantiene su eficacia frente a usuarios con diferentes niveles de capacidad lingüística. Por ello, la comprobación más relevante del sistema consiste en preparar una batería de NLQs donde se pueda validar el correcto funcionamiento ante distintas entradas. Para ello, se definieron cinco NLQs representativas, cada una expresada en tres niveles de complejidad: *profesional de BBDD*, *adulto estándar* y *baja capacidad*. Cada combinación se ejecutó veinte veces con el fin de obtener medidas de estabilidad, tasa de acierto y posibles errores.

En esta sección se presentan los resultados obtenidos para cada NLQ, incluyendo:

- El objetivo semántico de la consulta.
- La salida SQL esperada y el valor de resultado correcto en la base de datos.
- Las tasas de acierto, fallo y error obtenidas en las 20 repeticiones por nivel.
- Una representación gráfica comparativa de los tiempos medios de respuesta según el nivel.

Definiciones. *Acierto*: el valor devuelto coincide con el esperado. *Fallo*: la sentencia pasa validación y se ejecuta, pero el valor no coincide con el esperado. *Error*: el flujo se interrumpe (p. ej., validación rechazada o ejecución abortada).

5.2.1. Evaluación NLQs

NLQ 1

Consulta profesional: *"Count the number of distinct collars with temperatureAnomaly = true since 2021-01-01."*
Consulta adulto: *"How many cows have fever this year?"*
Consulta baja capacidad: *"Any cow wit fever this year."*

Objetivo semántico: Obtener el número de collares distintos que han registrado una anomalía de temperatura (`temperatureAnomaly = true`) desde 2021-01-01.

SQL esperada

```
SELECT COUNT(DISTINCT "entityName")
FROM collar
WHERE "time" >= '2021-01-01'
    AND "temperatureAnomaly" = true;
```

Resultados En la tabla 5.1 se resumen las tasas de acierto, fallo y error para las tres formulaciones de la NLQ 1, considerando como acierto la obtención exacta del valor esperado en la base de datos.

Nivel	Aciertos (%)	Fallos (%)	Errores (%)
Profesional	100	0	0
Adulto	75	25	0
Baja capacidad	75	25	0

Tabla 5.1: Resultados de la NLQ 1 por nivel

Análisis temporal Se presentan las gráficas creadas a partir de las medidas temporales de cada una de las fases del proceso y se muestran a continuación:

- **Tiempo absoluto**: La Figura 5.1 muestra la comparación de los tiempos medios de respuesta.
- **Dispersión**: La Figura 5.2 muestra la dispersión en formato caja y bigotes.
- **Repartición modular**: La Figura 5.3 muestra de forma relativa cómo se reparte el tiempo en cada módulo respecto del total.

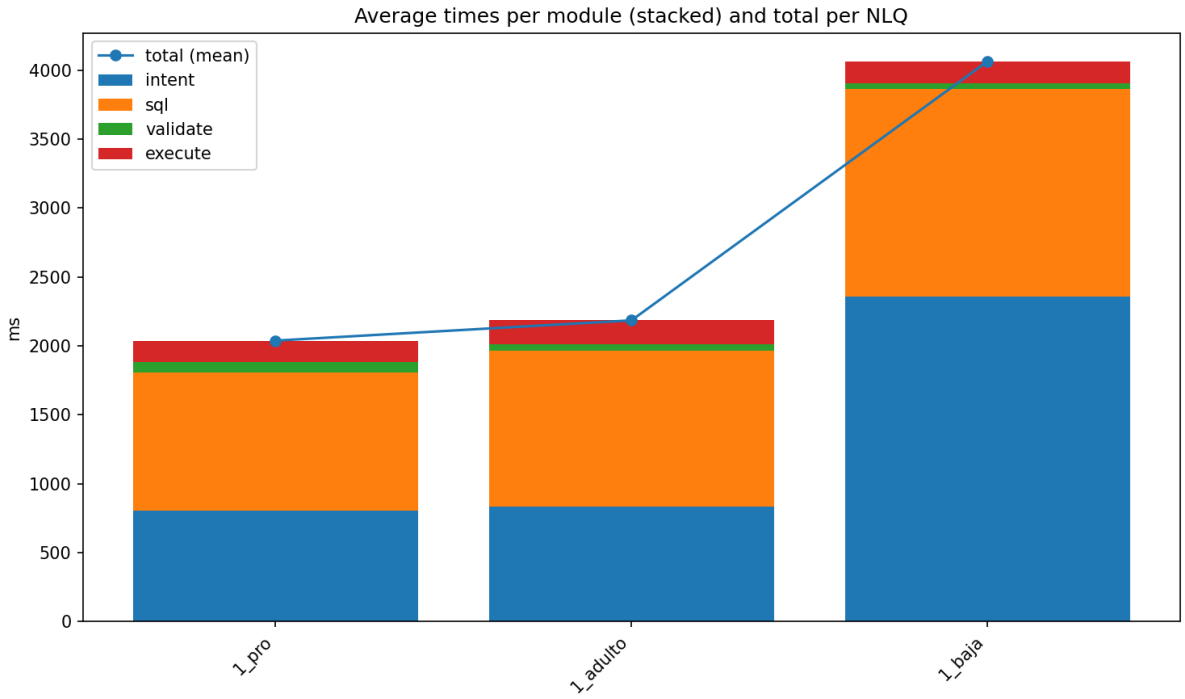

Figura 5.1: Comparativa de tiempos medios por nivel para la NLQ 1.

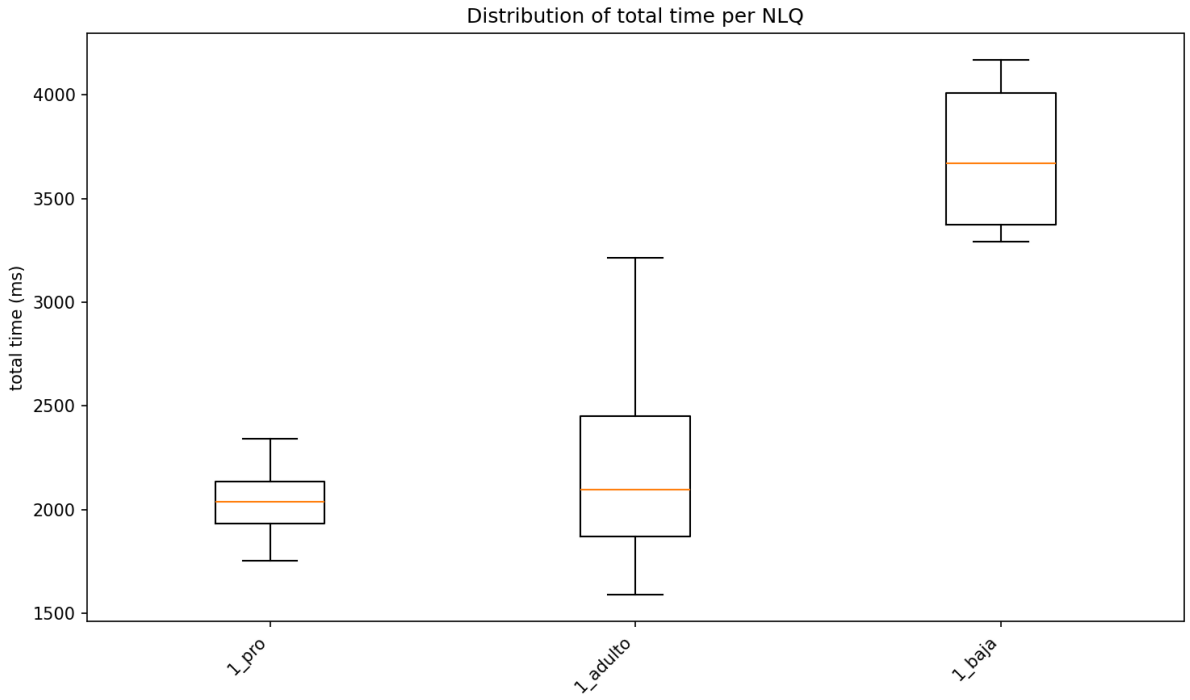

Figura 5.2: Comparativa de dispersión por nivel para la NLQ 1.

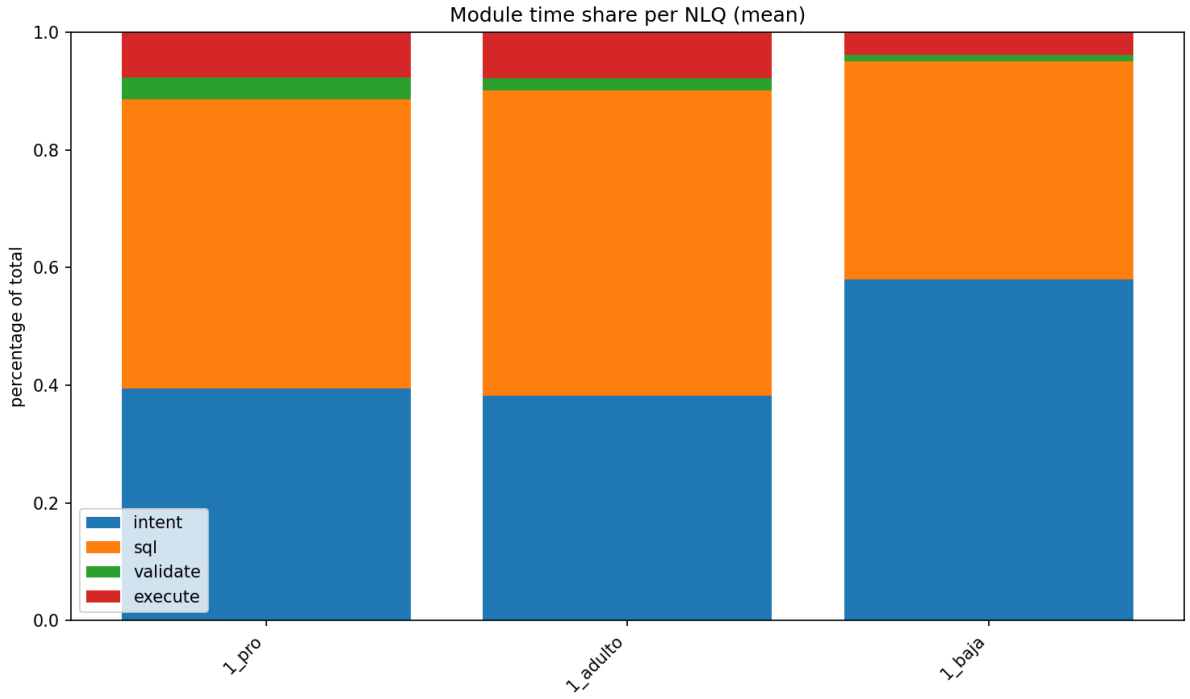

Figura 5.3: Comparativa de tiempo relativo por módulo por nivel para la NLQ 1.

Se aprecia cómo la formulación profesional presenta la respuesta más rápida y estable, mientras que las formulaciones menos estructuradas (adulto y baja capacidad) incrementan tanto la dispersión de los resultados como la proporción de respuestas que no coinciden exactamente con lo solicitado.

NLQs 2-5

Para no sobrecargar este capítulo con detalles individuales, el análisis completo de cada NLQ, incluyendo tasas de acierto, fallo y distribución temporal por nivel, se presenta en el Anexo II.

De la evaluación conjunta de todas las NLQs se desprende que los resultados pueden variar por distintos factores. En algunos casos, las consultas de baja capacidad, al ser más cortas, se procesan más rápidamente si son interpretadas correctamente. Por el contrario, las consultas profesionales, aunque semánticamente más precisas, requieren más tokens debido a su extensión, lo que puede ocasionar un mayor tiempo de procesamiento en ciertas ejecuciones.

En términos generales, se confirma la tendencia ya anticipada: a medida que disminuye la claridad y el nivel de detalle de la NLQ, aumenta la probabilidad de error y la dispersión en los tiempos de respuesta.

NLQs erróneas

Además de las consultas correctas, se diseñó un conjunto de cinco NLQs erróneas o sin sentido, con el objetivo de comprobar el comportamiento del sistema frente a entradas no válidas. Estas consultas incluían desde instrucciones de modificación de la base de datos (prohibidas en este contexto), hasta frases sin relación con el dominio o expresiones completamente aleatorias. La intención de estas pruebas no era obtener un resultado válido en la base de datos, sino verificar que el sistema fuera capaz de:

- Rechazar la consulta de manera controlada sin comprometer la integridad de los datos.
- Generar un mensaje de error informativo o proceder a reintentos según el mecanismo de validación definido.
- Mantener tiempos de respuesta acotados, incluso en presencia de entradas no procesables.

Las NLQs erróneas ejecutadas en el sistema fueron las siguientes:

- `Update the table to set temperature = 50 for all collars.`
- `Hello, how are you?`
- `Who won the World Cup in 2010?`
- `everything`
- `Florp the quantum of collar.`

Los resultados obtenidos mostraron que el validador cumplió su función al bloquear todas las consultas inválidas de tipo no permitido, como en el primer caso, impidiendo que se ejecutaran en la base de datos. Este comportamiento refuerza la robustez del sistema, garantizando que únicamente se procesen consultas de tipo `SELECT` y protegiendo la integridad de los datos frente a entradas maliciosas o no relacionadas con el dominio. En el resto de consultas, el sistema respondió con valores nulos o interpretaciones parciales, pero en ningún caso se produjeron fallos que comprometieran la estabilidad o seguridad de la aplicación.

En la Figura 5.4 se muestran los tiempos de ejecución para las NLQs erróneas. La consulta 1 no aparece representada, ya que siempre resultaba en error durante la validación y el proceso no llegaba a completarse.

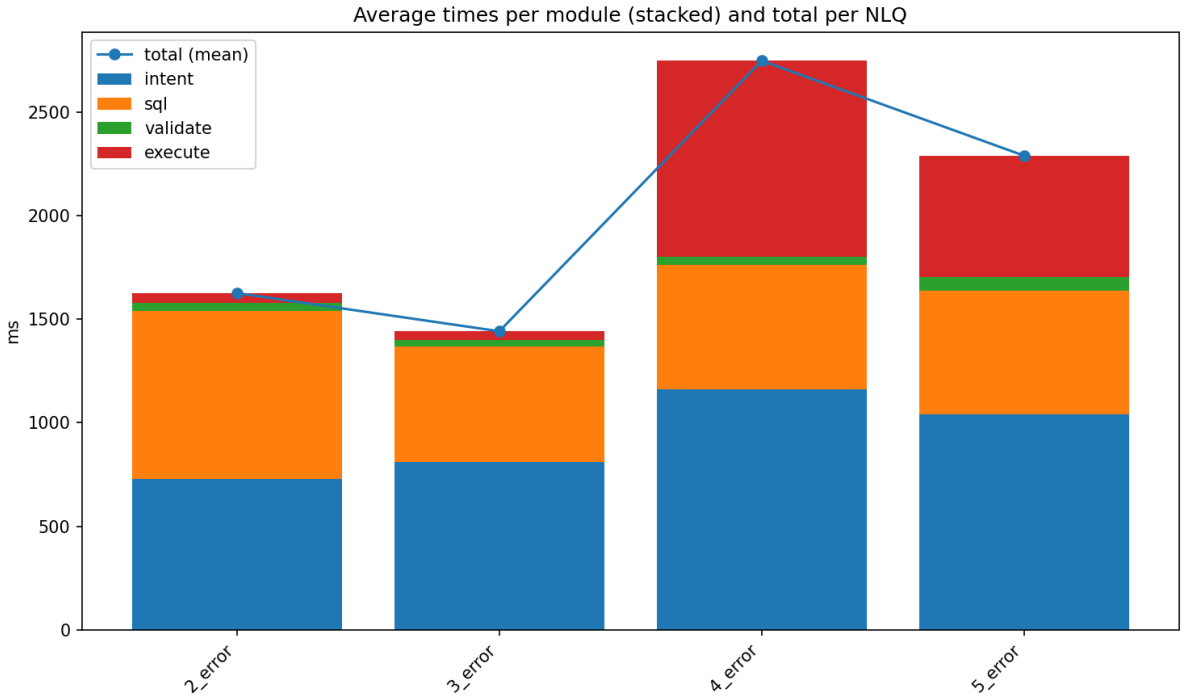

Figura 5.4: Comparativa de tiempos medios para consultas erróneas.

Como se puede observar, los tiempos medios de las consultas erróneas no superan a los de las NLQs correctamente formuladas. Esto se debe a que, ante entradas no válidas, el sistema genera una interpretación rápida y controlada, sin prolongar innecesariamente el procesamiento.

Análisis general de las NLQs

A continuación se muestra una visión global de los resultados agregados por nivel de complejidad, considerando todas las NLQs evaluadas. El objetivo es identificar tendencias generales en el tiempo de procesamiento y en la estabilidad de las respuestas en función de la calidad lingüística de la consulta.

En primer lugar, la tabla 5.2 muestra la media de los resultados obtenidos a partir de la ejecución de todas las NLQs utilizadas para el estudio.

Nivel	Aciertos (%)	Fallos (%)	Errores (%)
Profesional	100	0	0
Adulto	87	9	4
Baja capacidad	63	33	4

Tabla 5.2: Resultados agregados de las NLQs por nivel (100 ejecuciones por nivel)

En la Figura 5.5 se representa el tiempo medio de respuesta por NLQ y nivel. Se observa que las consultas formuladas en nivel profesional (*pro*) mantienen tiempos reducidos y relativamente estables en todas las NLQs, mientras que las de nivel *adulto* presentan un ligero incremento en la media. El nivel *baja capacidad* muestra un aumento más significativo en NLQs complejas , reflejando la dificultad adicional de interpretar entradas poco estructuradas.

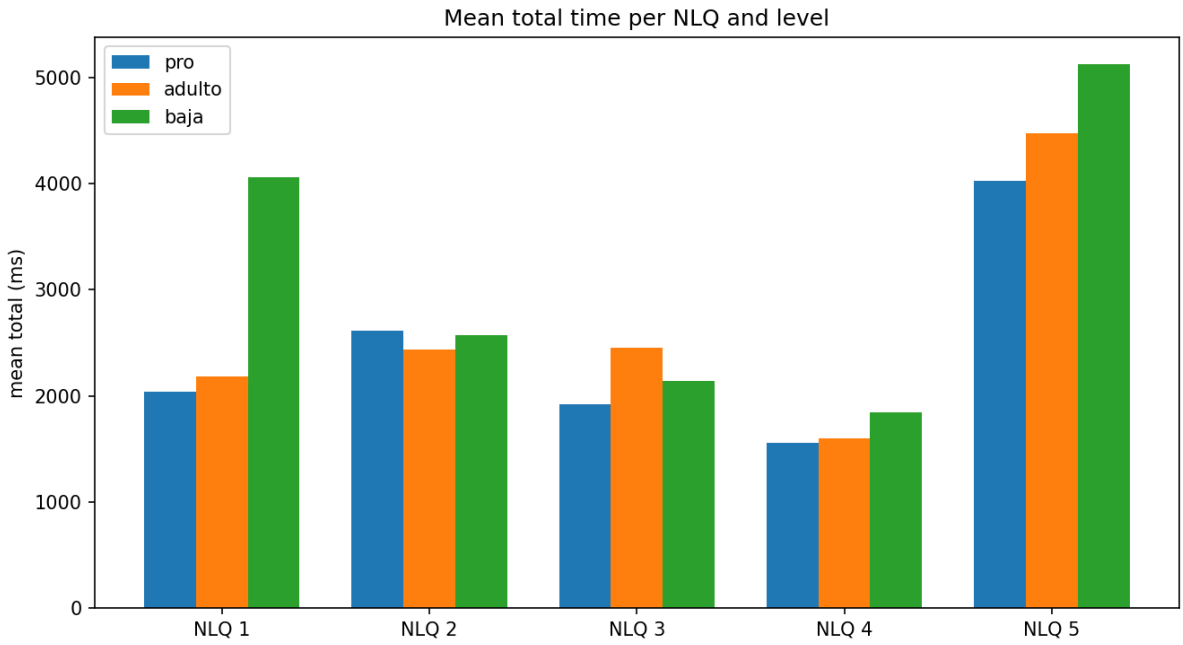

Figura 5.5: Tiempo medio de respuesta por NLQ y nivel.

La Figura 5.6 muestra la descomposición de los tiempos por módulo (extracción de intención, generación de SQL, validación y ejecución), junto con el tiempo total promedio. Se aprecia que los módulos dominantes en coste temporal son la extracción de intención y la generación de SQL, mientras que la validación y ejecución resultan marginales. Asimismo, los tiempos totales crecen desde *pro* hasta *baja*, confirmando que la ambigüedad en la formulación incrementa el esfuerzo de procesamiento.

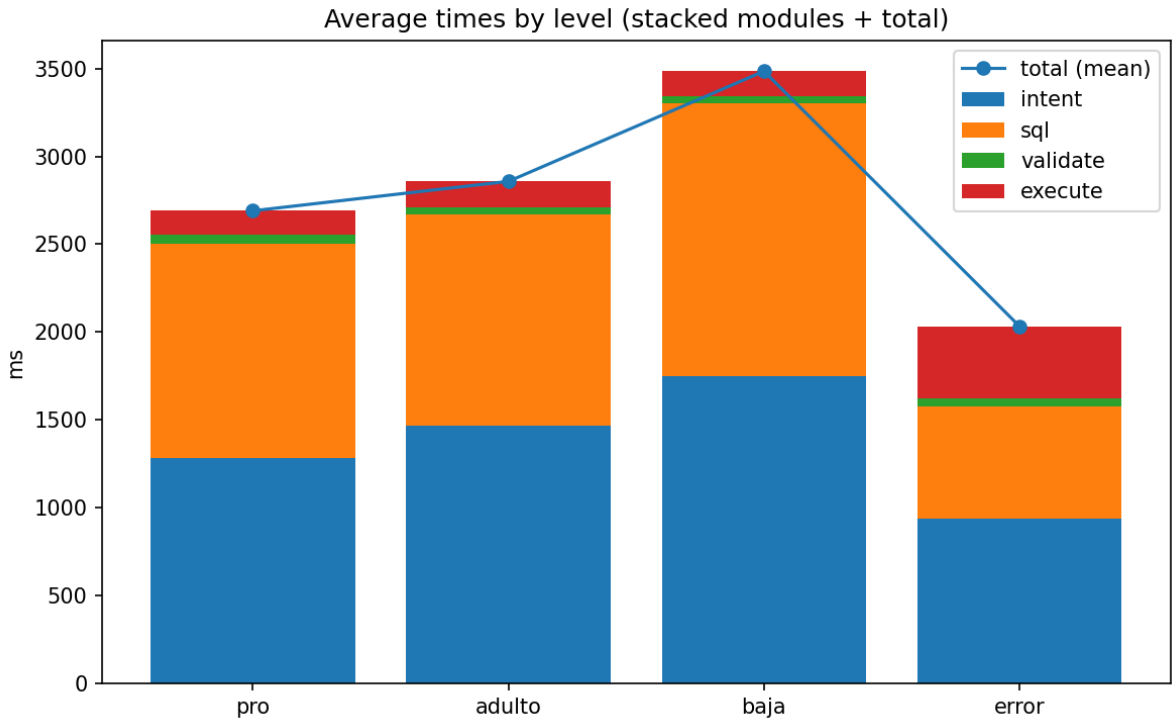

Figura 5.6: Tiempos promedio por módulo y total según el nivel de complejidad.

Finalmente, en la Figura 5.7 se muestra la distribución de los tiempos de respuesta mediante boxplots por nivel. Los resultados evidencian que las consultas profesionales presentan menor dispersión y mayor estabilidad, las de nivel adulto exhiben una variabilidad intermedia, y las de baja capacidad muestran tanto los tiempos más altos como la mayor dispersión, alcanzando valores muy superiores en algunos casos. Este patrón confirma que la robustez del sistema depende en gran medida de la calidad de la entrada proporcionada por el usuario.

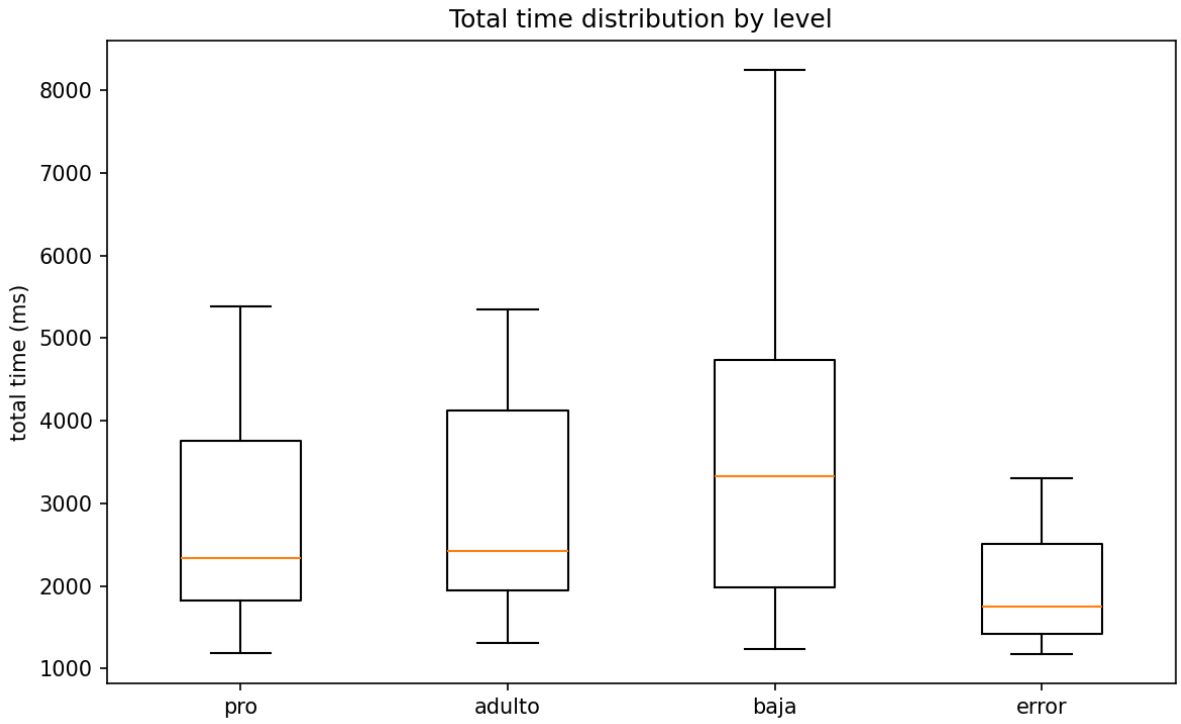

Figura 5.7: Distribución del tiempo total por nivel de complejidad.

5.2.2. Verificación de requisitos funcionales

Además de la evaluación cuantitativa mediante NLQs, se llevó a cabo la verificación sistemática de los requisitos funcionales definidos en la sección 3.2. El objetivo es comprobar que cada funcionalidad especificada (RF-01 a RF-07) está implementada y opera según lo previsto.

Objetivos

- Comprobar que todas las funcionalidades definidas en los requisitos funcionales han sido implementadas correctamente.
- Verificar que las salidas producidas por el sistema son coherentes con lo esperado para cada caso de uso.

Metodología

Las pruebas se ejecutaron a través de la interfaz activa del sistema, introduciendo consultas en lenguaje natural y observando los resultados obtenidos. Para registrar la trazabilidad de cada ejecución, el sistema genera un archivo JavaScript Object Notation (JSON) con la NLQ introducida, la sentencia SQL generada, el estado de validación, la respuesta final y tiempos de ejecución por módulo. Estos registros son los que se han utilizado como base para el análisis presentado en el punto 5.2.1.

ID	Requisito funcional	Caso de prueba	Resultado
RF-01	Recepción de consultas en lenguaje natural	Enviar NLQ desde la interfaz chatbot	Cumplido
RF-02	Traducción a SQL válida y coherente	NLQ correcta → SQL esperada	Cumplido
RF-03	Validación no destructiva mediante `EXPLAIN`	Enviar SQL candidata inválida	Cumplido
RF-04	Rechazo de operaciones de modificación	Enviar NLQ tipo `UPDATE`	Cumplido
RF-05	Ejecución de consulta en PostgreSQL	SQL válida ejecutada en BD real	Cumplido
RF-06	Presentación del resultado en interfaz	Resultado de la BD mostrado en texto plano	Cumplido
RF-07	Registro mínimo de actividad	Comprobar JSON con NLQ, SQL, estado y timestamp	Cumplido

Tabla 5.3: Verificación de requisitos funcionales

Los resultados de la tabla 5.3 muestran que todos los requisitos funcionales definidos en el diseño han sido satisfechos por el sistema implementado. Esta verificación complementa las pruebas con NLQs reales, asegurando que tanto la funcionalidad interna como la interacción de usuario cumplen con las especificaciones.

5.3. Pruebas no funcionales

Estas pruebas evalúan características del sistema que no están directamente relacionadas con funcionalidades específicas, sino con su calidad global: rendimiento, robustez, usabilidad, seguridad o mantenibilidad. La validación se ha realizado tanto a partir de métricas obtenidas en la experimentación (tiempos de respuesta, tasas de acierto, errores controlados) como mediante comprobaciones de diseño y despliegue (estructura modular, portabilidad, coste).

Tipos de prueba realizados

- **Rendimiento**: latencia media y percentiles de tiempo de respuesta por petición.
- **Robustez**: comportamiento frente a NLQs erróneas o fuera de dominio.
- **Compatibilidad**: ejecución de la interfaz en distintos navegadores y entornos locales.
- **Usabilidad**: simplicidad de la interfaz conversacional y validación con el tutor como usuario final.

- **Seguridad y privacidad**: bloqueo de operaciones de modificación y uso de HTTPS en llamadas a LLM.
- **Mantenibilidad**: estructura modular en capas y documentación del código.
- **Coste**: estimación del coste de operación por consulta según tarifas del proveedor LLM.

ID	Requisito no funcional	Evidencia	Resultado
RNF-01	Latencia media ≤ 5s, p95 ≤ 10s	Fig. 5.7	Cumplido
RNF-02	Precisión sintáctica ≥ 90 % sin error	Tab. 5.2	Cumplido
RNF-03	0 consultas destructivas	NLQ tipo `UPDATE` bloqueada	Cumplido
RNF-04	Mensajes de error comprensibles	Mensajes en interfaz	Cumplido
RNF-05	Mantenibilidad modular	Arquitectura Cap. 4.3	Cumplido
RNF-06	Portabilidad Python 3.11+	Despliegue reproducible	Cumplido
RNF-07	Observabilidad	Logs JSON por petición	Cumplido
RNF-08	Privacidad y seguridad	Prompts sin datos personales, HTTPS	Cumplido
RNF-09	Coste <0,01€ / petición	Estimación con tarifas OpenAI	Cumplido

Tabla 5.4: Verificación de requisitos no funcionales

Herramientas utilizadas

La validación de requisitos no funcionales se ha apoyado en los registros JSON generados por el sistema (NLQ, SQL, estado de validación, latencia, resultado), así como en las gráficas comparativas presentadas en la sección 5.2. Para la parte de compatibilidad y usabilidad se realizaron pruebas manuales en distintos navegadores (Chrome, Firefox, Edge) y sesiones de validación con el tutor como usuario final.

Análisis del coste individual

Para sustentar el cumplimiento del requisito no funcional de coste (RNF-09: Coste <0,01 € / petición), se ha realizado una estimación empírica a partir de los registros reales obtenidos en la plataforma de *OpenAI*. En la Figura 5.8 y la Figura 5.9 se representan los registros reales obtenidos en la plataforma de OpenAI correspondientes a un día de uso representativo (5 de octubre de 2025), durante el cual el sistema realizó 48 peticiones a la API. Dado que cada consulta completa envía dos *prompts* (uno para la extracción de intención y otro para la generación de SQL), el número real de consultas procesadas es 24.

En ese periodo se registraron 30 310 tokens de entrada y 1 423 tokens de salida, lo que equivale a una media de aproximadamente **1 263 tokens de entrada** y **59 tokens de salida por consulta**.

Aplicando las tarifas oficiales del modelo *gpt-4o-mini* ($0.15 por millón de tokens de entrada y $0.60 por millón de tokens de salida) [48], el coste medio por consulta se calcula como:

$$C_{\text{consulta}} = \frac{1263}{10^6} \times 0{,}15 + \frac{59}{10^6} \times 0{,}60 \approx 2{,}1 \times 10^{-4}\ \text{USD}\ (\approx 0{,}00018\ € \text{ por consulta [50]})$$

Incluso considerando fluctuaciones en el número de tokens o el tipo de cambio, el valor permanece muy por debajo del umbral de 0,01 €, demostrando la viabilidad económica del sistema en uso real.

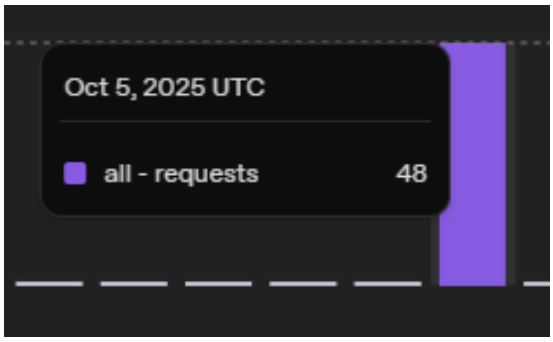

Figura 5.8: Número de peticiones realizadas en un día de uso (5 octubre 2025).

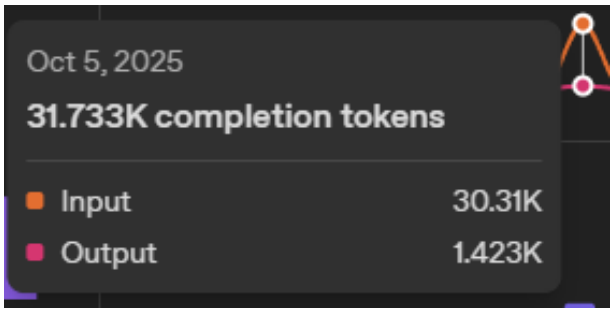

Figura 5.9: Total de tokens procesados (entrada y salida) para el mismo periodo.

5.4. Validación funcional con el usuario final

Una vez verificados los requisitos funcionales y no funcionales en un entorno de laboratorio, se procedió a la validación de la solución en un contexto realista de uso. El objetivo fue comprobar que el sistema resultaba útil y comprensible para un usuario no técnico, tal y como se estableció en los objetivos del proyecto.

Las consultas empleadas para la validación funcional fueron proporcionadas por el tutor del proyecto y se basan en ejemplos reales formulados por agricultores y especialistas del dominio que participaron en el proyecto europeo *AFarCloud*. Estas consultas representan escenarios de uso verosímiles —por ejemplo, detección de anomalías de temperatura, identificación de collares, localización de animales o detección de alarmas— y fueron seleccionadas para reflejar necesidades reales en el ámbito ganadero.

La validación se llevó a cabo mediante la ejecución directa de dichas consultas sobre el sistema, en una sesión supervisada por el tutor actuando como usuario experto del dominio. Durante la evaluación se registraron observaciones cualitativas relacionadas con la claridad de la interfaz, la coherencia de los resultados obtenidos y la velocidad de respuesta.

Por motivos de confidencialidad, la fuente exacta de las consultas originales no se revela, aunque su adaptación conserva íntegramente el propósito y el nivel de complejidad de las peticiones reales. Esto permite afirmar que la validación se realizó bajo condiciones realistas y representativas del entorno de aplicación.

Resultado

El sistema cumplió con los objetivos funcionales definidos: las consultas en lenguaje natural fueron interpretadas correctamente en la mayoría de los casos, devolviendo resultados coherentes con los datos de la base de datos. El tutor confirmó que la interfaz es adecuada para un usuario sin conocimientos de SQL, al presentar los resultados en formato claro y sin información técnica innecesaria.

Se identificaron algunas mejoras menores de presentación, como la necesidad de clarificar etiquetas en la interfaz y ajustar ligeramente los mensajes de error. También alguna mejora de los prompts utilizados para una mejor respuesta en algunos casos. Estas observaciones fueron incorporadas en la versión final. En conjunto, la validación confirmó que la solución es viable para su uso en un entorno real y satisface las necesidades planteadas al inicio del proyecto.

5.5. Resumen del capítulo

Este capítulo ha documentado el proceso de verificación y validación del sistema, siguiendo los estándares *IEEE 1012-2016* [2] e *ISO/IEC/IEEE 29119* [3].

Las pruebas funcionales demostraron que el sistema es capaz de transformar consultas en lenguaje natural de distintos niveles de complejidad en sentencias SQL válidas y de obtener los resultados correctos en la base de datos, con una alta tasa de acierto en los escenarios de referencia. Asimismo, las pruebas con NLQs erróneas confirmaron la robustez del validador, que bloqueó correctamente todas las operaciones no permitidas.

Las pruebas no funcionales mostraron que el rendimiento medio por consulta se mantuvo dentro de los umbrales definidos, que la variabilidad aumenta con consultas menos estructuradas, y que no se ejecutaron consultas destructivas en ningún caso. También se verificaron aspectos de usabilidad, seguridad, mantenibilidad y coste.

Finalmente, la validación con el tutor como usuario experto del dominio confirmó que la interfaz es adecuada para usuarios no técnicos y que los resultados son comprensibles y útiles en un escenario realista.

En conjunto, los resultados obtenidos permiten concluir que la solución desarrollada cumple con los requisitos funcionales y no funcionales y es apta para el propósito definido al inicio del proyecto.

6. Presupuesto

El presente capítulo desarrolla el análisis económico del proyecto, con el objetivo de estimar de forma realista los recursos necesarios para su ejecución. Para ello, se han considerado tanto los costes directos (equipamiento, software y mano de obra) como los costes indirectos asociados al consumo de energía y otros gastos operativos.

La metodología empleada combina un modelo de amortización proporcional para los recursos materiales con el cálculo del coste laboral total, siguiendo criterios habituales en la evaluación de proyectos de ingeniería. De esta forma, se obtiene una estimación global del presupuesto que permite valorar la viabilidad económica del sistema desarrollado y proporciona una base comparativa para futuras implementaciones en entornos reales.

6.1. Estimación de Costes

Los costes del proyecto se han agrupado en las siguientes categorías:

- **Costes de Equipamiento y Software**: amortización proporcional al uso en el proyecto.
- **Costes de Desarrollo y Mano de Obra**: basados en el coste laboral total, no en el salario neto.
- **Costes Indirectos**: estimados como un 15 % del total de personal y equipamiento.

6.2. Costes de Equipamiento y Software

El proyecto se ha desarrollado empleando principalmente un ordenador portátil personal, herramientas de software de libre acceso y una API de modelo de lenguaje que sí tiene un coste asociado. Para estimar el gasto real atribuible al proyecto, se ha aplicado una amortización proporcional en el caso del hardware y un prorrateo del uso en el caso del software de pago.

La fórmula aplicada para el equipamiento es:

$$C_{\text{proyecto}} = \frac{C_{\text{total}}}{\text{vida útil}} \times \text{meses de uso} \tag{6.1}$$

En el caso del portátil:

- Ordenador valorado en 1 500 € con vida útil estimada de 60 meses (5 años).
- Tiempo de uso en el proyecto: 4 meses.
- Coste asignado:

$$1\,500 \times \frac{4}{60} = 100\,€ \tag{6.2}$$

Respecto al software empleado:

- **VS Code**, **PostgreSQL** y librerías como `Flask`, `LangChain` o `psycopg2` son de libre distribución, por lo que su coste económico directo es 0 €.
- **API de OpenAI**: durante el desarrollo se realizaron peticiones de prueba al modelo `gpt-4o-mini` con un precio de (por millón de tokens) $0,15 de entrada y $0,60 de salida [48]. Finalizadas las pruebas, el coste total asciende a 0,05 €.

Concepto	Coste Total (€)	Vida Útil	Meses	Coste (€)
Ordenador portátil	1.500	60 meses	4	100,00
VS Code, PostgreSQL, librerías	0	–	–	0,00
API OpenAI	0,05	–	–	0,05
Total				**100,05**

Tabla 6.1: Costes de Equipamiento y Software

6.3. Costes de Desarrollo y Mano de Obra

Los costes de personal se estiman utilizando el coste total de mano de obra, que incluye el coste laboral, compuesto por el salario y otros costes asociados. El cálculo se basa en los siguientes factores:

- Un ingeniero con un coste laboral total de 36,71 €/h [51].
- Se dedicaron 350 horas al desarrollo del proyecto.
- Coste total:

$$350 \times 36{,}71 = 12\,848{,}50\,€ \tag{6.3}$$

Categoría	Horas	Coste por hora (€)	Coste Total (€)
Ingeniero	350	36,71	12.848,50
Total			**12.848,50**

Tabla 6.2: Costes Laborales

6.4. Costes Indirectos

Los costes indirectos incluyen energía, acceso a Internet y otros gastos operativos. Se estiman como un 15 % del total de personal y equipamiento:

$$C_{\text{indirectos}} = 0{,}15 \times \left(C_{\text{equipamiento}} + C_{\text{mano\_obra}}\right) \tag{6.4}$$

Aplicando la fórmula:

$$0{,}15 \times (100{,}05 + 12\,848{,}50) = 1\,942{,}28\,€ \tag{6.5}$$

6.5. Resumen del Presupuesto

A continuación, se presenta el desglose final del presupuesto del proyecto:

Concepto	**Coste (€)**
Equipamiento y Software	100,05
Mano de Obra	12.848,50
Costes Indirectos (15 %)	1.942,28
Total	**14.890,83**

Tabla 6.3: Resumen del Presupuesto

6.6. Conclusión

El presupuesto estimado del proyecto asciende a 14.890,83 €, considerando el coste realista de los recursos utilizados y la metodología de amortización aplicada. Este análisis permite evaluar la viabilidad económica del sistema propuesto y justificar su posible implementación en un entorno real.

El mayor peso presupuestario recae en la mano de obra (más del 85 % del total), lo que refleja que el valor del proyecto se centra principalmente en el trabajo de desarrollo y no en los costes materiales.

7. Análisis de impacto

Este capítulo analiza los impactos derivados del desarrollo e implementación del sistema de traducción de consultas en lenguaje natural a SQL, considerando dimensiones tecnológicas, científicas, económicas, sociales, éticas, medioambientales y su alineación con los Objetivos de Desarrollo Sostenible (ODS).

El análisis permite valorar la proyección del trabajo más allá del entorno académico, destacando su aplicabilidad en el ámbito de la ganadería inteligente, su contribución a la investigación en procesamiento de lenguaje natural y bases de datos, y su potencial de transferencia tecnológica hacia la industria agro-tecnológica.

7.1. Impacto tecnológico

El sistema desarrollado aporta un avance tecnológico en varios aspectos relevantes:

- Propone un enfoque práctico para la transformación de lenguaje natural en consultas SQL, combinando LLMs con un mecanismo de validación y ejecución controlada sobre PostgreSQL. Este planteamiento puede aplicarse a otros dominios donde los usuarios requieran acceso a bases de datos sin conocimientos técnicos.

- Mantiene compatibilidad con estándares abiertos como SQL y el uso de API REST, lo que facilita la interoperabilidad con distintos sistemas gestores de bases de datos y su integración en arquitecturas web o plataformas existentes.

- Aunque el caso de estudio se centra en el ámbito agrícola y ganadero, la solución es fácilmente adaptable a cualquier dominio donde se requiera que usuarios no técnicos interactúen con bases de datos relacionales. La adaptación requiere únicamente ajustar la base de datos de destino y el *prompt* que guía al modelo, lo que permite su aplicación en sectores como sanidad, logística, educación, comercio electrónico o gestión de infraestructuras inteligentes.

7.2. Impacto en la investigación

Los resultados del proyecto se alinean con varias líneas de investigación activas y presentan potencial de transferencia académica:

- **Interfaces de Lenguaje Natural a Bases de Datos** (NLIDB) y **Text-to-SQL**: el sistema combina LLMs con un ciclo de validación (`EXPLAIN`) y ejecución restringida sobre PostgreSQL, contribuyendo a enfoques prácticos de *schema-grounded prompting* y control de seguridad en consultas generadas.

- **Evaluación empírica con variabilidad lingüística**: la batería de NLQs en tres niveles de complejidad y múltiples repeticiones aporta un protocolo reproducible para estudiar el efecto del nivel de formulación del usuario en la tasa de acierto y la latencia.
- **Metodología y artefactos reutilizables**: los módulos de orquestación, validación y registro JSON con NLQs, SQL, estados y tiempos) facilitan la replicación y la comparación con futuros modelos o *prompts*.
- **Potencial de difusión**: los resultados pueden derivar en una comunicación corta o *work-in-progress* centrada en (i) la evaluación por niveles de usuario en Natural Language to SQL (NL2SQL) y (ii) el diseño de un *guardrail* simple y eficaz para ejecución segura de SQL con LLMs.

7.3. Impacto económico y empresarial

El análisis económico realizado en el capítulo 6 confirma que el sistema puede desarrollarse e implementarse con una inversión relativamente baja, especialmente gracias al uso de software libre y a que el coste principal recae en las horas de desarrollo. El coste asociado a las llamadas al modelo de lenguaje es reducido y controlable, lo que refuerza la viabilidad de la solución en entornos reales.

Desde una perspectiva de transferencia tecnológica, el sistema podría evolucionar hacia un Producto Mínimo Viable (MVP) orientado a usuarios no técnicos que necesiten consultar bases de datos en distintos sectores. Asimismo, el enfoque modular facilita su integración como componente dentro de plataformas ya existentes, ampliando su atractivo comercial.

En términos de ventaja competitiva, la solución aporta eficiencia en el acceso a datos, reducción de la curva de aprendizaje para usuarios sin conocimientos de SQL y un mecanismo de seguridad que limita riesgos en la ejecución de consultas. Estas características diferencian al sistema frente a soluciones genéricas de asistencia por IA y lo posicionan como una propuesta con potencial de explotación empresarial en contextos como agricultura de precisión, sanidad, logística o gestión de infraestructuras.

7.4. Impacto social y ético

El sistema desarrollado presenta implicaciones sociales y éticas relevantes:

- **Accesibilidad y reducción de la brecha digital**: al permitir que usuarios sin conocimientos técnicos interactúen con bases de datos mediante lenguaje natural, se facilita el acceso a información que de otro modo estaría restringida a perfiles especializados. Esto contribuye a democratizar el uso de tecnologías de análisis de datos en ámbitos como la agricultura, la educación o la gestión pública.
- **Privacidad y uso responsable de la IA**: el diseño garantiza que no se incluyen datos personales en los *prompts*, que todas las consultas se ejecutan en modo lectura y que se bloquean operaciones potencialmente destructivas. De este modo, se refuerza la transparencia y se minimizan los riesgos de un uso indebido de la inteligencia artificial.

- **Transparencia y confianza**: al validar y registrar cada consulta procesada, el sistema ofrece trazabilidad de su funcionamiento, lo que aumenta la confianza de los usuarios y facilita auditorías en entornos donde la fiabilidad es un requisito crítico.

7.5. Impacto ambiental

Aunque el proyecto no implica el uso de componentes físicos ni procesos industriales, sí tiene un impacto medioambiental asociado al consumo de recursos computacionales:

- **Consumo energético reducido**: el sistema se ejecuta en entornos locales ligeros (ordenador personal y base de datos PostgreSQL) y delega el procesamiento intensivo en un servicio en la nube, lo que evita la necesidad de infraestructuras adicionales de alto consumo.
- **Uso eficiente de llamadas a modelos**: las consultas al LLM se han optimizado para minimizar la longitud de los *prompts* y el número de repeticiones necesarias, reduciendo así el número de tokens procesados y, por tanto, la huella energética indirecta.
- **Escalabilidad sostenible**: en un eventual despliegue, el sistema puede adaptarse a infraestructuras en la nube con certificaciones de eficiencia energética, alineándose con prácticas de sostenibilidad tecnológica.

7.6. Impacto en los Objetivos de Desarrollo Sostenible (ODS)

Los ODS constituyen una referencia internacional para el desarrollo sostenible. El proyecto contribuye a estos objetivos de forma directa o indirecta:

- **ODS 9 – Industria, Innovación e Infraestructura**: al promover el desarrollo de una solución tecnológica innovadora y escalable.
- **ODS 11 – Ciudades y Comunidades Sostenibles**: si la solución se aplica a servicios públicos, comunicaciones o sostenibilidad urbana.
- **ODS 13 – Acción por el Clima**: al reducir el consumo energético, las emisiones y fomentar el uso de tecnologías limpias.

La inclusión de estos objetivos en el diseño del proyecto refuerza su valor social y su alineación con las metas de desarrollo global promovidas por la Organización de las Naciones Unidas (ONU).

Resumen del capítulo

El análisis de impacto realizado permite valorar la proyección y el alcance del proyecto más allá de su implementación técnica. Se ha evaluado su repercusión tecnológica, económica, social, ética y ambiental, así como su contribución a los ODS. Esta evaluación integral refuerza la relevancia del trabajo desarrollado y muestra su potencial de transferencia a otros dominios. Además, establece un puente natural hacia las conclusiones generales del proyecto, donde se sintetizan los logros alcanzados y se identifican posibles líneas de evolución futura.

Dimensión de impacto	Descripción / resumen	Tipo de impacto
Tecnológico	Transformación de NLQ a SQL con validación y ejecución controlada; solución interoperable con estándares abiertos.	Directo
Investigador	Proyecto alineado con la línea de investigación del grupo GRyS; reutilizable en futuros trabajos académicos (Natural Language Interface to Databases (NLIDB)/NL2SQL).	Potencial
Económico	Presupuesto razonable y posibilidad de transferencia como módulo o MVP en entornos reales.	Directo
Social y ético	Facilita el acceso de usuarios no técnicos a bases de datos; considera privacidad y uso responsable de IA.	Indirecto
Ambiental	Consumo energético reducido mediante optimización de recursos computacionales.	Directo
ODS	Contribuye a los ODS 9, 11 y 13 según el enfoque de sostenibilidad tecnológica.	Directo

Tabla 7.1: Resumen de impactos del proyecto

8. Conclusiones y trabajos futuros

8.1. Conclusiones generales del proyecto

El presente Proyecto Fin de Grado ha abordado de forma sistemática la definición, diseño, desarrollo e implementación de un sistema capaz de transformar consultas en lenguaje natural (NLQ) en sentencias SQL válidas, ejecutarlas de forma segura sobre una base de datos PostgreSQL y devolver al usuario el resultado en texto plano a través de una interfaz conversacional.

A lo largo del documento se han establecido y justificado las especificaciones funcionales y no funcionales, se ha desarrollado una arquitectura modular basada en el uso de LLMs y se han implementado los componentes necesarios para cubrir todo el flujo de interacción: extracción de intención, generación de consulta SQL, validación, ejecución en la base de datos y presentación del resultado.

Las fases de verificación y validación han demostrado que la solución es operativa, robusta y adecuada para el propósito previsto: permitir que usuarios no técnicos accedan a la información contenida en una base de datos relacional sin necesidad de conocimientos en SQL.

Finalmente, el análisis de impacto realizado en el capítulo 7 ha puesto de manifiesto la proyección real del sistema, no solo desde el punto de vista técnico, sino también en términos de viabilidad, sostenibilidad y beneficios sociales asociados al uso responsable de la IA.

8.2. Logros alcanzados

- Diseño e implementación de una arquitectura modular que integra LLMs, un generador de *prompts* estructurado, un validador de consultas y un ejecutor SQL sobre PostgreSQL.
- Desarrollo de una interfaz conversacional sencilla que permite a usuarios no técnicos interactuar con la base de datos en lenguaje natural.
- Cumplimiento de los requisitos funcionales definidos, garantizando la recepción de NLQs, la traducción a SQL, la validación segura de las sentencias y la presentación del resultado en texto plano.
- Verificación de los requisitos no funcionales principales, incluyendo precisión sintáctica, seguridad en la ejecución y costes controlados en el uso de la API de LLMs.
- Realización de una campaña de pruebas con consultas de distinto nivel de complejidad y consultas erróneas, que ha permitido validar la robustez del sistema y caracterizar su rendimiento.
- Evaluación cualitativa con el tutor como usuario experto, confirmando la utilidad y viabilidad del sistema en un entorno realista.

8.3. Limitaciones del trabajo

Como todo proyecto técnico, este trabajo presenta ciertas limitaciones que es importante reconocer para contextualizar los resultados obtenidos:

- El alcance ha estado condicionado por el tiempo disponible y la naturaleza individual del proyecto, lo que ha limitado la posibilidad de realizar despliegues a gran escala o con múltiples usuarios.
- La validación con usuarios finales se restringió a la evaluación por parte del tutor como experto de dominio, sin pruebas con un grupo más amplio de usuarios no técnicos.
- El sistema depende de un proveedor externo de modelos de lenguaje (*OpenAI*), lo que introduce cierta dependencia en términos de coste, latencia y disponibilidad futura.
- Algunas pruebas no funcionales, como la escalabilidad en entornos de producción o la robustez frente a cargas elevadas, no pudieron abordarse dentro del marco del proyecto y quedaron fuera del alcance experimental.
- La precisión en las consultas de nivel bajo depende en gran medida de la claridad de la entrada. Aunque el sistema es robusto frente a errores sintácticos simples, la ambigüedad semántica sigue siendo un desafío.

Estas limitaciones no invalidan los logros alcanzados, pero sí señalan líneas claras de mejora para futuros desarrollos.

8.4. Líneas de trabajo futuro

A partir de los resultados obtenidos y de las limitaciones identificadas, se proponen las siguientes posibles líneas de evolución del proyecto:

- **Extensión de funcionalidades**: incorporar nuevos mecanismos de interacción con el usuario, como interfaces gráficas enriquecidas o integración con asistentes de voz, que amplíen el alcance del sistema más allá del chatbot actual.
- **Mejora de la precisión**: explorar técnicas de ajuste fino (*fine-tuning*) sobre LLMs, o el uso de *retrieval augmented generation* (RAG), para aumentar la robustez en consultas ambiguas o de baja capacidad lingüística.
- **Validación a mayor escala**: realizar pruebas con un número significativo de usuarios no técnicos, en un entorno más cercano a producción, con el fin de medir usabilidad, fiabilidad y aceptación real del sistema. Además, una evaluación extensa permitiría optimizar los *prompts* utilizados.
- **Adaptación a otros dominios**: generalizar la herramienta a ámbitos como sanidad, logística o educación, demostrando su versatilidad mediante el cambio del esquema de base de datos y del *prompt* utilizado.

- **Optimización de costes y rendimiento**: evaluar proveedores alternativos de modelos de lenguaje, explorar despliegues en entornos locales o híbridos y aplicar técnicas de compresión de modelos para reducir tiempos de respuesta y costes asociados.
- **Apertura y transferencia**: liberar el código bajo una licencia de software abierto o plantear la publicación de resultados en congresos y revistas técnicas, fomentando su reutilización y la colaboración investigadora.
- **Mejora de la seguridad y control de acceso**: incorporar un módulo de gestión de roles y permisos que permita identificar, mediante autenticación y políticas de acceso, qué datos puede consultar cada usuario. Esta funcionalidad aportaría un mayor control sobre la información disponible, garantizando la trazabilidad y la protección de los datos almacenados en la base de datos. Además, sentaría las bases para una futura escalabilidad multiusuario del sistema, facilitando su despliegue en entornos corporativos o institucionales.

8.5. Resumen del capítulo

En este capítulo se han sintetizado las principales conclusiones del proyecto, destacando el cumplimiento de los objetivos iniciales, los logros alcanzados y las limitaciones detectadas. El sistema desarrollado ha demostrado ser una solución viable para la traducción de consultas en lenguaje natural a sentencias SQL, con un funcionamiento robusto y adaptable a distintos dominios.

Asimismo, se han propuesto líneas de trabajo futuro orientadas a la ampliación de funcionalidades, la mejora de la precisión, la validación a mayor escala y la transferencia a otros ámbitos de aplicación. Estas perspectivas aseguran la continuidad y el potencial de evolución del proyecto más allá del contexto académico en el que se ha desarrollado.

En conjunto, el proyecto cumple los objetivos planteados y sienta una base sólida para futuras líneas de investigación y desarrollo aplicadas a interfaces en lenguaje natural.

9. Referencias

[1] *Aggregate Farming in the Cloud | AFarCloud | Project | Fact Sheet | H2020*. url: https://cordis.europa.eu/project/id/783221 (visitado 2025-08-14).

[2] *IEEE Standard for System, Software, and Hardware Verification and Validation*. doi: 10.1109/IEEESTD.2017.8055462. url: https://ieeexplore.ieee.org/document/8055462/ (visitado 2024-07-25).

[3] *ISO/IEC/IEEE International Standard - Software and Systems Engineering - Software Testing – Part 2: Test Processes*. doi: 10.1109/IEEESTD.2021.9591508. url: https://ieeexplore.ieee.org/document/9591508/ (visitado 2024-07-25).

[4] *IEEE/ISO/IEC International Standard for Software and Systems Engineering–Software Testing–Part 3:Test Documentation*. doi: 10.1109/IEEESTD.2021.9591577. url: https://ieeexplore.ieee.org/document/9591577/ (visitado 2024-07-25).

[5] Xiaohu Zhu, Qian Li, Lizhen Cui y Yongkang Liu. *Large Language Model Enhanced Text-to-SQL Generation: A Survey*. Oct. de 2024. doi: 10.48550/arXiv.2410.06011. arXiv: 2410.06011 [cs]. (Visitado 2025-08-07).

[6] Segev Shlomov et al. *IDA: Breaking Barriers in No-code UI Automation Through Large Language Models and Human-Centric Design*. Ago. de 2024. doi: 10.48550/arXiv.2407.15673. arXiv: 2407.15673 [cs]. (Visitado 2025-08-07).

[7] *LLM Text-to-SQL Solutions: Top Challenges and Tips*. url: https://www.k2view.com/blog/llm-text-to-sql/ (visitado 2025-08-07).

[8] *(PDF) The Lunar Sciences Natural Language Information System*. url: https://www.researchgate.net/publication/24285293_The_Lunar_Sciences_Natural_Language_Information_System (visitado 2025-10-08).

[9] Bozenn H. Thompson y Frederick B. Thompson. «Introducing ASK, A Simple Knowledgeable System». En: *First Conference on Applied Natural Language Processing*. Santa Monica, California, USA: Association for Computational Linguistics, feb. de 1983, págs. 17-24. doi: 10.3115/974194.974198. (Visitado 2025-10-08).

[10] I. Androutsopoulos, G. D. Ritchie y P. Thanisch. *Natural Language Interfaces to Databases - An Introduction*. Mar. de 1995. doi: 10.48550/arXiv.cmp-lg/9503016. arXiv: cmp-lg/9503016. (Visitado 2025-08-07).

[11] Victor Zhong, Caiming Xiong y Richard Socher. *Seq2SQL: Generating Structured Queries from Natural Language Using Reinforcement Learning*. Nov. de 2017. doi: 10.48550/arXiv.1709.00103. arXiv: 1709.00103 [cs]. (Visitado 2025-08-07).

[12] Tao Yu et al. *Spider: A Large-Scale Human-Labeled Dataset for Complex and Cross-Domain Semantic Parsing and Text-to-SQL Task*. Feb. de 2019. doi: 10.48550/arXiv.1809.08887. arXiv: 1809.08887 [cs]. (Visitado 2025-08-07).

[13] Adrián Bazaga, Pietro Liò y Gos Micklem. *SQLformer: Deep Auto-Regressive Query Graph Generation for Text-to-SQL Translation*. Mayo de 2024. doi: 10.48550/arXiv.2310.18376. arXiv: 2310.18376 [cs]. (Visitado 2025-08-07).

[14] Tom B. Brown et al. *Language Models Are Few-Shot Learners*. Jul. de 2020. doi: 10.48550/arXiv.2005.14165. arXiv: 2005.14165 [cs]. (Visitado 2025-08-07).

[15] Mark Chen et al. *Evaluating Large Language Models Trained on Code*. Jul. de 2021. doi: 10.48550/arXiv.2107.03374. arXiv: 2107.03374 [cs]. (Visitado 2025-08-07).

[16] Dawei Gao et al. «Text-to-SQL Empowered by Large Language Models: A Benchmark Evaluation». En: *Proceedings of the VLDB Endowment* 17.5 (ene. de 2024), págs. 1132-1145. issn: 2150-8097. doi: 10.14778/3641204.3641221. (Visitado 2025-08-07).

[17] *ISO/IEC/IEEE International Standard - Software and Systems Engineering –Software Testing –Part 1:General Concepts*. doi: 10.1109/IEEESTD.2022.9698145. url: https://ieeexplore.ieee.org/document/9698145/ (visitado 2024-07-25).

[18] *IEEE/ISO/IEC International Standard - Software and Systems Engineering–Software Testing–Part 4: Test Techniques*. doi: 10.1109/IEEESTD.2021.9591574. url: https://ieeexplore.ieee.org/document/9591574/ (visitado 2024-07-25).

[19] *ISO/IEC 25010:2023*. url: https://www.iso.org/es/contents/data/standard/07/81/78176.html (visitado 2025-08-08).

[20] *Directrices Para La Redacción de Referencias Bibliográficas y de Citas de Recursos de Información*. Ver. 2024. Madrid, 2024-44. url: https://www.une.org/encuentra-tu-norma/busca-tu-norma/norma/?c=norma-une-iso-690-2024-n0072823.

[21] *Regulation - EU - 2024/1689 - EN - EUR-Lex*. url: https://eur-lex.europa.eu/eli/reg/2024/1689/oj/eng (visitado 2025-08-08).

[22] PostgreSQL Global Development Group. *PostgreSQL*. Oct. de 2025. url: https://www.postgresql.org/ (visitado 2025-10-08).

[23] *Welcome to Flask — Flask Documentation (3.1.x)*. url: https://flask.palletsprojects.com/en/stable/ (visitado 2025-10-08).

[24] *GPT-4o mini: una apuesta por la inteligencia rentable*. url: https://openai.com/es-ES/index/gpt-4o-mini-advancing-cost-efficient-intelligence/ (visitado 2025-08-08).

[25] *LangChain*. url: https://www.langchain.com (visitado 2025-10-08).

[26] *LangGraph*. url: https://www.langchain.com/langgraph (visitado 2025-10-08).

[27] *ATIS SLSP Corpus*. url: https://catalog.ldc.upenn.edu/docs/LDC93S4B/corpus.html (visitado 2025-10-08).

[28] *(PDF) Evaluating the Text-to-SQL Capabilities of Large Language Models*. doi: 10.48550/arXiv.2204.00498. url: https://www.researchgate.net/publication/359709736_Evaluating_the_Text-to-SQL_Capabilities_of_Large_Language_Models (visitado 2025-10-08).

[29] Adhityasing Rajaput et al. «Text to SQL Generation Using Beam Search and an Enhanced Bert Model:» en: *Proceedings of the 3rd International Conference on Futuristic Technology*. Hotel Crowne Plaza pune, India: SCITEPRESS - Science and Technology Publications, 2025, págs. 49-55. isbn: 978-989-758-763-4. doi: 10.5220/0013608600004664. (Visitado 2025-10-08).

[30] Albert Wong et al. *Translating Natural Language Queries to SQL Using the T5 Model*. Dic. de 2023. doi: 10.48550/arXiv.2312.12414. arXiv: 2312.12414 [cs]. (Visitado 2025-10-08).

[31] *Django*. url: https://www.djangoproject.com/ (visitado 2025-10-08).

[32] *FastAPI*. url: https://fastapi.tiangolo.com/ (visitado 2025-10-08).

[33] *Introducing GPT-5*. Oct. de 2025. url: https://openai.com/index/introducing-gpt-5/ (visitado 2025-10-08).

[34] meta-llama. *Llama 3: MODEL_CARD.md at main · meta-llama/llama3*. url: https://github.com/meta-llama/llama3/blob/main/MODEL_CARD.md (visitado 2025-10-08).

[35] *Overview | Claude*. url: https://www.claude.com/product/overview (visitado 2025-10-08).

[36] *MySQL*. url: https://www.mysql.com/ (visitado 2025-10-08).

[37] *MariaDB Foundation*. url: https://mariadb.org/ (visitado 2025-10-08).

[38] *SQLite Home Page*. url: https://sqlite.org/ (visitado 2025-10-08).

[39] *SQL Server 2022 | Microsoft*. url: https://www.microsoft.com/es-es/sql-server/sql-server-2022 (visitado 2025-10-08).

[40] *MongoDB: The World's Leading Modern Database*. url: https://www.mongodb.com/ (visitado 2025-10-08).

[41] markingmyname. *Microsoft Copilot in Azure with Azure SQL Database Overview - Azure SQL*. url: https://learn.microsoft.com/en-us/azure/azure-sql/copilot/copilot-azure-sql-overview?view=azuresql (visitado 2025-08-10).

[42] *How Dataherald Makes Natural Language to SQL Easy*. Feb. de 2024. url: https://blog.langchain.com/dataherald/ (visitado 2025-08-10).

[43] Netflix Technology Blog. *Automation as a Service — Introducing Scriptflask*. Mayo de 2017. url: https://netflixtechblog.com/automation-as-a-service-introducing-scriptflask-17a8e4ad954b (visitado 2025-08-11).

[44] Netflix Technology Blog. *Python at Netflix*. Abr. de 2019. url: https://netflixtechblog.com/python-at-netflix-bba45dae649e (visitado 2025-08-11).

[45] reddit-archive. *Architecture Overview*. url: https://github.com/reddit-archive/reddit/wiki/Architecture-Overview (visitado 2025-08-11).

[46] Instagram Engineering. *Instagration Pt. 2: Scaling Our Infrastructure to Multiple Data Centers*. Mayo de 2016. url: https://instagram-engineering.com/instagration-pt-2-scaling-our-infrastructure-to-multiple-data-centers-5745cbad7834 (visitado 2025-08-11).

[47] Fujitsu Australia Software Technology. «Fujitsu's Strategy in Incorporating PostgreSQL into Its Enterprise Database». En: ().

[48] *Pricing - OpenAI API*. url: `https://platform.openai.com` (visitado 2025-08-26).

[49] *Grupo de Redes y Servicios de Próxima Generación (GRyS) - Universidad Politécnica de Madrid*. url: `https://grys.etsist.upm.es/about` (visitado 2025-08-14).

[50] *Tipo de cambio de EUR a USD y noticias relacionadas*. url: `https://www.google.com/finance/quote/EUR-USD` (visitado 2025-10-09).

[51] *Coste laboral por hora efectiva por divisiones de la CNAE-09(6037)*. INE. url: `https://www.ine.es/jaxiT3/Tabla.htm?t=6037&L=0` (visitado 2025-04-13).

I. Prompts utilizados

En este anexo se incluyen los *prompts* completos empleados en el sistema, de manera que el lector pueda consultar la plantilla exacta enviada al modelo de lenguaje.

I.1. Prompt para extracción de intención (Prompt #1)

Se incluye la NLQ del usuario en {query}.

```
You are an intelligent system that helps interpret questions from farmers with
limited semantic clarity.
Extract the main intent of the following query as a clear and concise sentence
that can help a database expert write an SQL query.
Return ONLY the intent, without explanations or formatting.

Query: {query}

Use the following database schema to better understand the terms used in the
query.

Schema:
Table: collar

Field: "time"
Type: TEXT

Field: "entityName"
Type: TEXT
Description: Unique ID for each collar, always 2 uppercase letters followed by
3 digits (e.g., "AH306"). Case-sensitive.

Field: "scenario"
Type: TEXT

Field: "provider"
Type: TEXT

Field: "service"
Type: TEXT

Field: "latitude"
Type: NUMERIC

Field: "longitude"
```

```
Type: NUMERIC

Field: "altitude"
Type: NUMERIC

Field: "accX"
Type: NUMERIC

Field: "accY"
Type: NUMERIC

Field: "accZ"
Type: NUMERIC

Field: "temperature"
Type: NUMERIC

Field: "temperatureAnomaly"
Type: BOOLEAN
Description: True if the recorded temperature is unusually high.

Field: "activityAnomaly"
Type: BOOLEAN
Description: True if the collar detects abnormal movement patterns (e.g. moving
too fast).

Field: "distanceAnomaly"
Type: BOOLEAN
Description: True if the collar moved an unusually long distance in a short
time.

Field: "positionAnomaly"
Type: BOOLEAN
Description: True if the GPS data is invalid or erratic.

Field: "locationAnomaly"
Type: BOOLEAN
Description: True if the collar is located outside its usual area of activity.

Field: "resourceAlarm"
Type: BOOLEAN
Description: True if a resource-related alert was triggered (e.g. battery
warning).

Field: "sequenceNumber"
Type: INTEGER
Description: Order of the measurement within the device's timeline.
### Response:
```

Listing I.1: Prompt #1: extracción de intención

I.2. Prompt para generación de SQL (Prompt #2)

Se incluye la NLQ del usuario en {query} y la intención extraída en {intent}.

```
### Instructions:
Your task is to convert a question into a SQL query, given a Postgres
database schema.
Adhere to these rules:
- **Deliberately go through the question and database schema word by word**
to appropriately answer the question
- When creating a ratio, always cast the numerator as float
- When the identifier of a table field is enclosed in quotation marks (e.g.
"entityName") its use in the query must also be enclosed in quotation marks.
- Return ONLY the SQL code, without explanations or formatting (no Markdown).

### Temporal context:
For the purpose of this task, consider that "today" refers to the date
"2021-07-28".

If the question refers to the current time (e.g. "right now", "currently",
"now", "actual position"), then select the **latest available reading from
today**.

If the user asks how many animals met a condition (e.g., had fever, showed
abnormal activity, etc.) over a period of time, return the number of
*distinct* animals ("entityName"), not the number of records.

### Input:
Generate a SQL query that answers the question "{query}"
The extracted intent of the question is: "{intent}"

This query will run on a database whose schema is represented in this string:
CREATE TABLE collar (
  "time" TEXT, -- Timestamp of the measurement in ISO 8601 format
  (e.g., 2021-07-28T18:33:11Z, UTC)

  "entityName" TEXT, -- Unique ID for each collar, always 2 uppercase letters
  followed by 3 digits (e.g., "AH306"). Case-sensitive.

  scenario TEXT, -- ID of the actuation scenario in which the collar has been
  deployed

  provider TEXT, -- ID of the device provider

  service TEXT, -- Service provided by the device

  type TEXT, -- Type of device (this field always has the value "collar")

  latitude NUMERIC, -- Latitude coordinate of the geoposition of the device
  at the specific timestamp of measurement generation
```

```
  longitude NUMERIC, -- Longitude coordinate of the geoposition of the
  device at the specific timestamp of measurement generation

  altitude NUMERIC, -- Altitude coordinate of the geopositioning of the
  device at the specific timestamp of measurement generation

  geohash TEXT, -- Geohash that express the geoposition of the device at
  the specific timestamp of measurement generation

  "accX" NUMERIC, -- X-axis accelerometer value

  "accY" NUMERIC, -- Y-axis accelerometer value

  "accZ" NUMERIC, -- Z-axis accelerometer value

  "activityAnomaly" BOOLEAN,  -- True if the collar detects abnormal
  movement patterns

  "distanceAnomaly" BOOLEAN,  -- True if the collar moved a long distance
  in a short time

  "positionAnomaly" BOOLEAN,  -- True if GPS data is invalid or erratic

  "locationAnomaly" BOOLEAN,  -- True if the collar is outside its usual
  activity zone

  "temperatureAnomaly" BOOLEAN,  -- True if the recorded temperature is
  unusually high

  "resourceAlarm" BOOLEAN,  -- True if the device raised an alert (e.g.
  battery warning)

  "sequenceNumber" INTEGER, -- Sequence number of the measurement with
  respect to the other measurements generated by the same device

  temperature NUMERIC, -- Temperature value generated by the device

  CONSTRAINT key PRIMARY KEY ("entityName", "time")
);

### Response:
```

Listing I.2: Prompt #2: generación de SQL

II. Análisis individual de NLQs

En este anexo se incluyen los análisis de las 5 NLQs divididas por niveles de dificultad; *Profesional de BBDD*, *adulto* y *baja capacidad*.

II.1. NLQ 1

Consulta profesional: *"Count the number of distinct collars with temperatureAnomaly = true since 2021-01-01."*
Consulta adulto: *"How many cows have fever this year?"*
Consulta baja capacidad: *"Any cow wit fever this year."*

Objetivo semántico: Obtener el número de collares distintos que han registrado una anomalía de temperatura (`temperatureAnomaly = true`) desde 2021-01-01.

SQL esperada

```
SELECT COUNT(DISTINCT "entityName")
FROM collar
WHERE "time" >= '2021-01-01'
    AND "temperatureAnomaly" = true;
```

II.1.1. Resultados

En la tabla II.1 se resumen las tasas de acierto, fallo y error para las tres formulaciones de la NLQ 1, considerando como acierto la obtención exacta del valor esperado en la base de datos.

Tabla II.1: Resultados de la NLQ 1 por nivel

Nivel	Aciertos (%)	Fallos (%)	Errores (%)
Profesional	100	0	0
Adulto	75	25	0
Baja capacidad	75	25	0

II.1.2. Análisis temporal

Se han generado 3 gráficas a partir de las medidas temporales de cada una de las fases del proceso y se muestran a continuación:

- **Tiempo absoluto**: La Figura II.1 muestra la comparación de los tiempos medios de respuesta.

- **Dispersión**: La Figura II.2 muestra la dispersión en formato caja y bigotes.

- **Repartición modular**: La Figura II.3 muestra de forma relativa como se reparte el tiempo en cada módulo respecto del total.

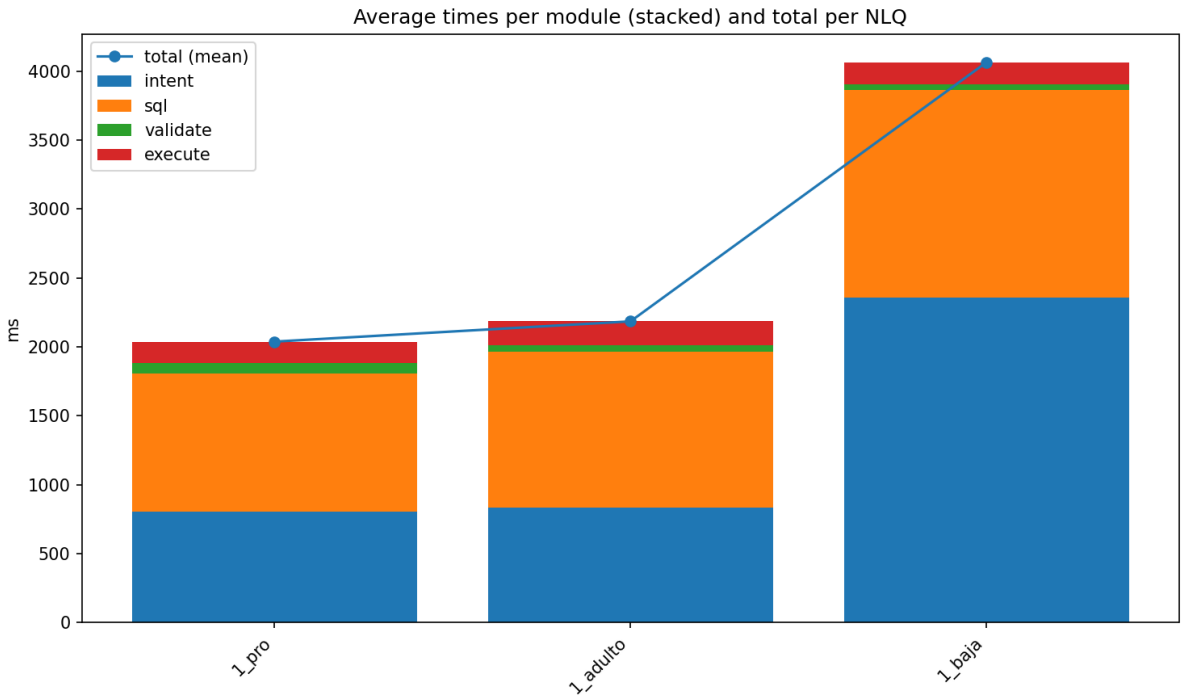

Figura II.1: Comparativa de tiempos medios por nivel para la NLQ 1.

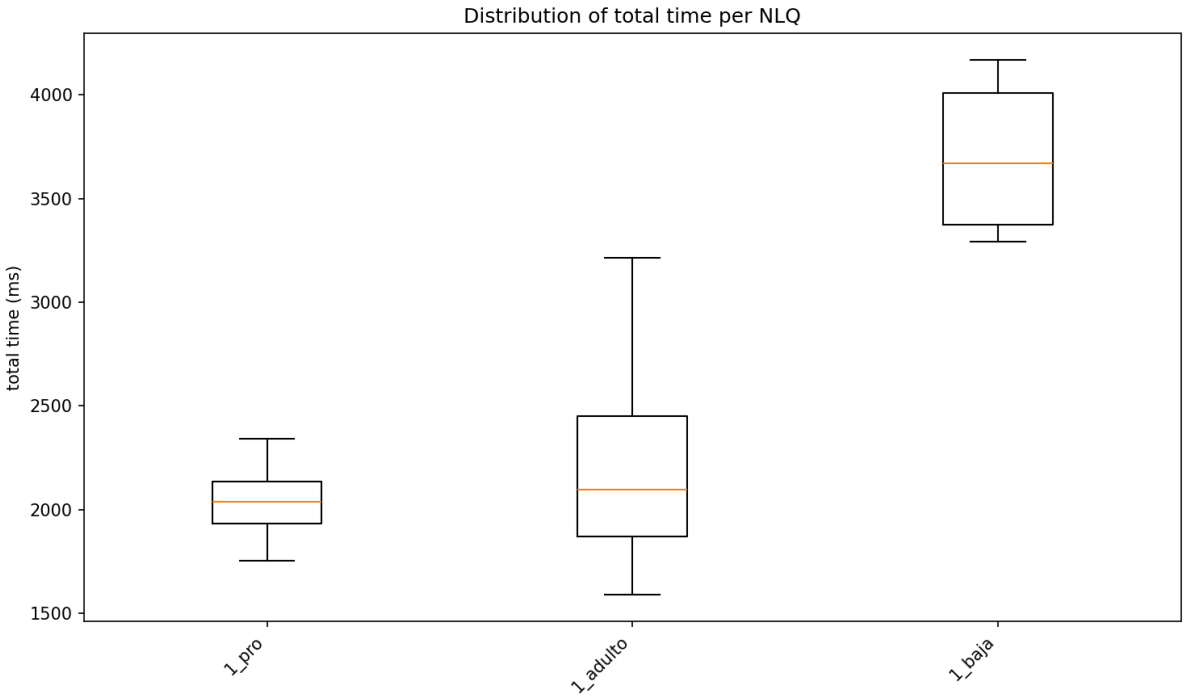

Figura II.2: Comparativa de dispersión por nivel para la NLQ 1.

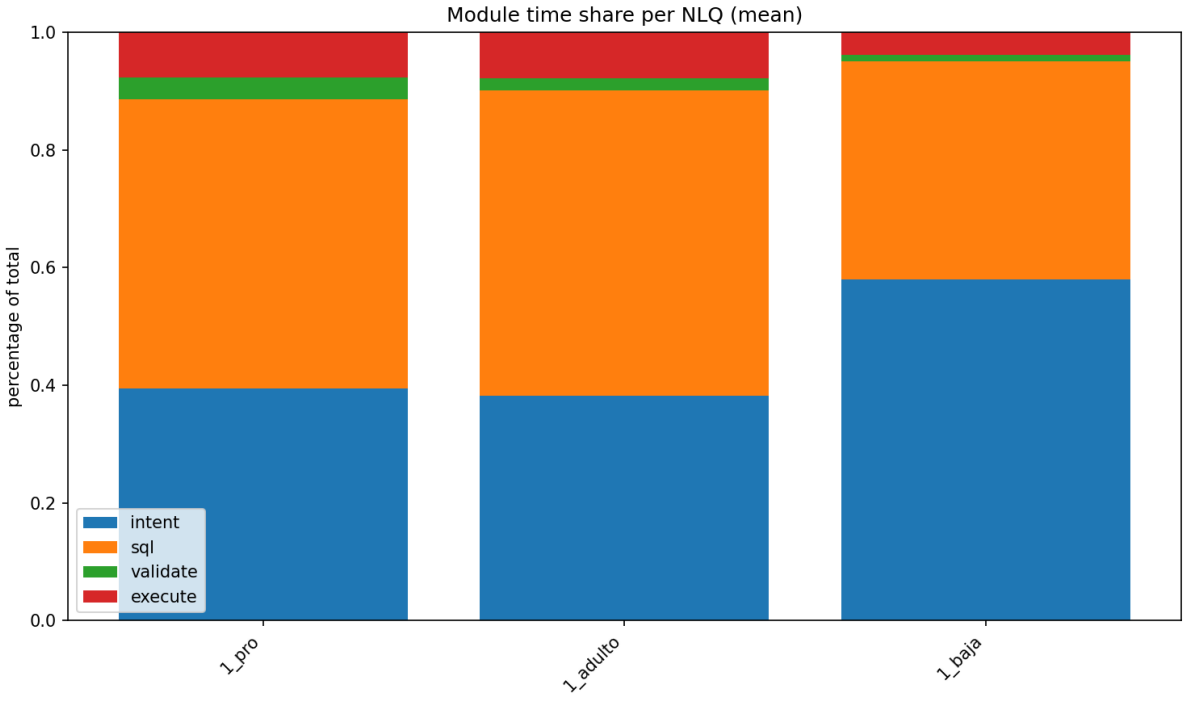

Figura II.3: Comparativa de tiempo relativo por módulo por nivel para la NLQ 1.

II.2. NLQ 2

Consulta profesional: *“Retrieve the latitude and longitude of entityName AH277 at the latest time today.”*
Consulta adulto: *“Where is collar AH277 right now?”*
Consulta baja capacidad: *“where is ah277”*

Objetivo semántico: Obtener la posición (latitud y longitud) del collar ”AH277”.

SQL esperada

```
SELECT "latitude", "longitude"
FROM collar WHERE "entityName" = 'AH277'
AND "time" = (SELECT MAX("time") FROM collar
    WHERE "entityName" = 'AH277'
    AND "time"::timestamp::date = '2021-07-28');
```

II.2.1. Resultados

Tabla II.2: Resultados de la NLQ 2 por nivel

Nivel	Aciertos (%)	Fallos (%)	Errores (%)
Profesional	100	0	0
Adulto	100	0	0
Baja capacidad	85	15	0

II.2.2. Análisis temporal

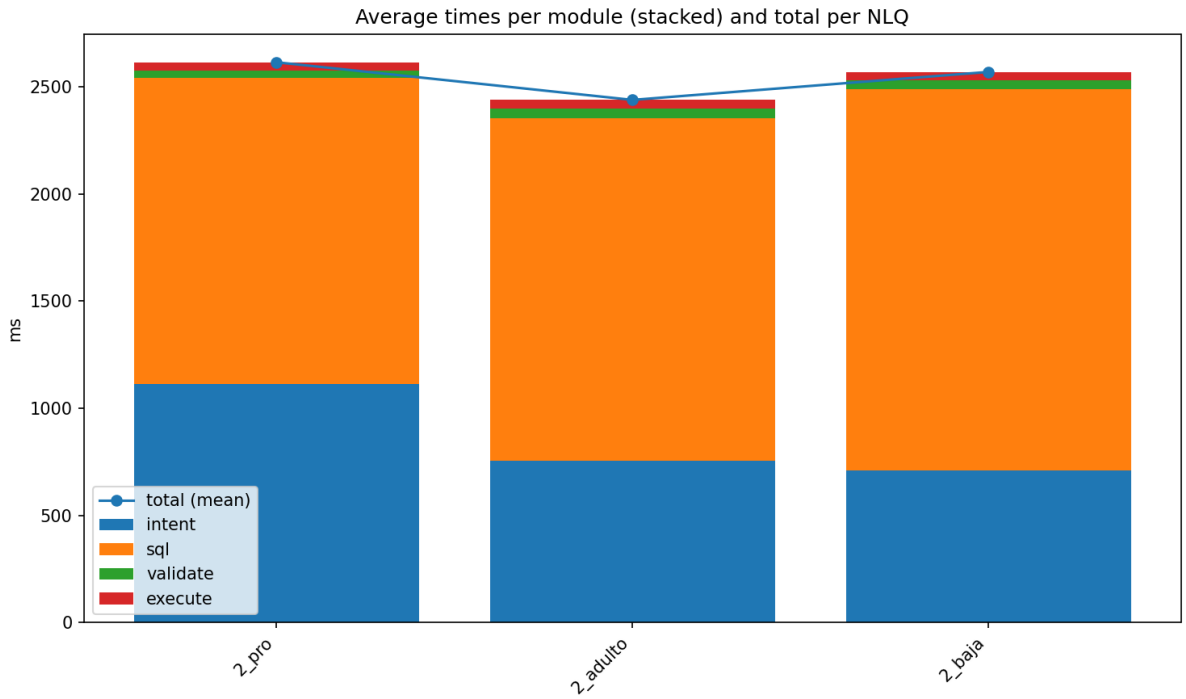

Figura II.4: Comparativa de tiempos medios por nivel para la NLQ 2.

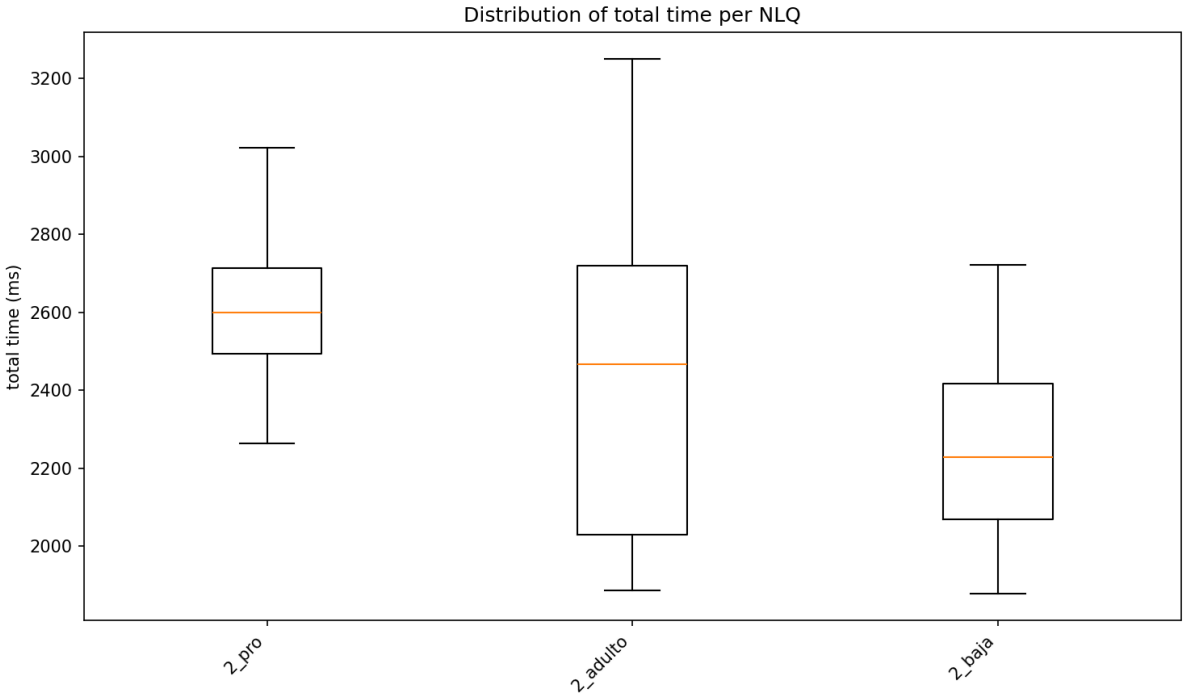

Figura II.5: Comparativa de dispersión por nivel para la NLQ 2.

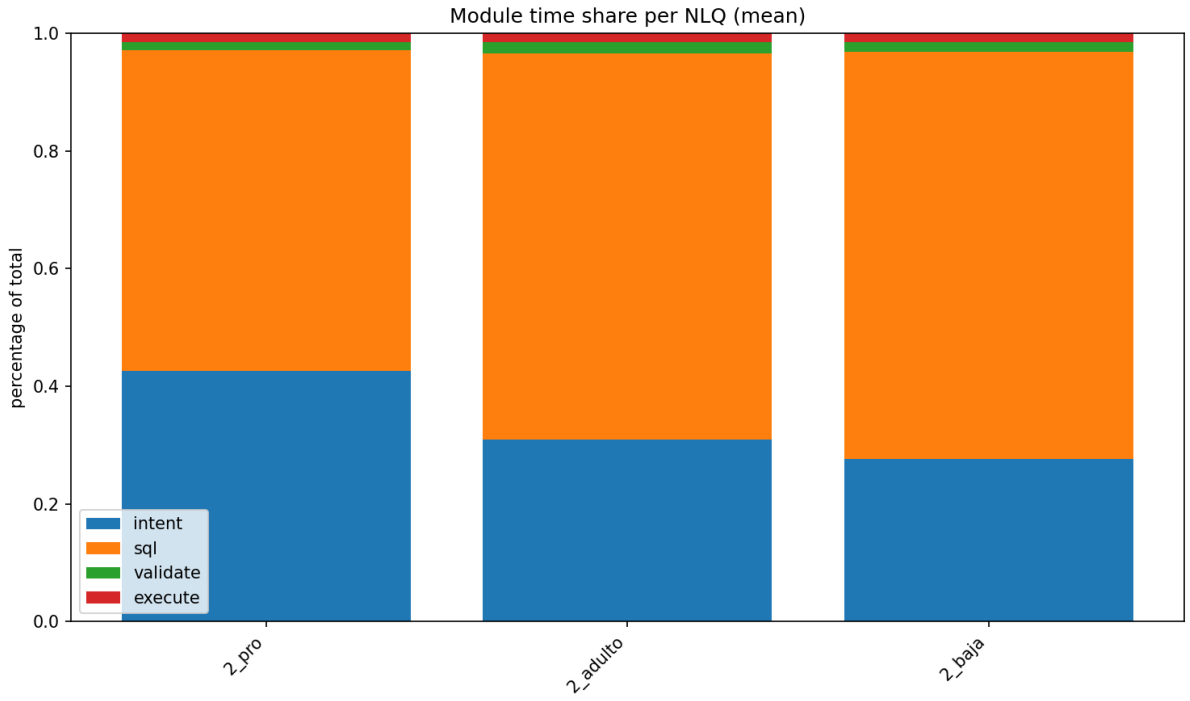

Figura II.6: Comparativa de tiempo relativo por módulo por nivel para la NLQ 2.

II.3. NLQ 3

Consulta profesional: *"Get the maximum temperature recorded by any collar since 2020-07-28."*
Consulta adulto: *"What's the maximum temperature recorded one year from today?"*
Consulta baja capacidad: *"Maxima temperate from one year"*

Objetivo semántico: Obtener la máxima temperatura registrada en cualquier collar desde 2020-07-28.

SQL esperada

```
SELECT MAX(temperature)
FROM collar
WHERE "time" >= '2020-07-28';
```

II.3.1. Resultados

Tabla II.3: Resultados de la NLQ 3 por nivel

Nivel	Aciertos (%)	Fallos (%)	Errores (%)
Profesional	100	0	0
Adulto	75	20	5
Baja capacidad	30	55	15

II.3.2. Análisis temporal

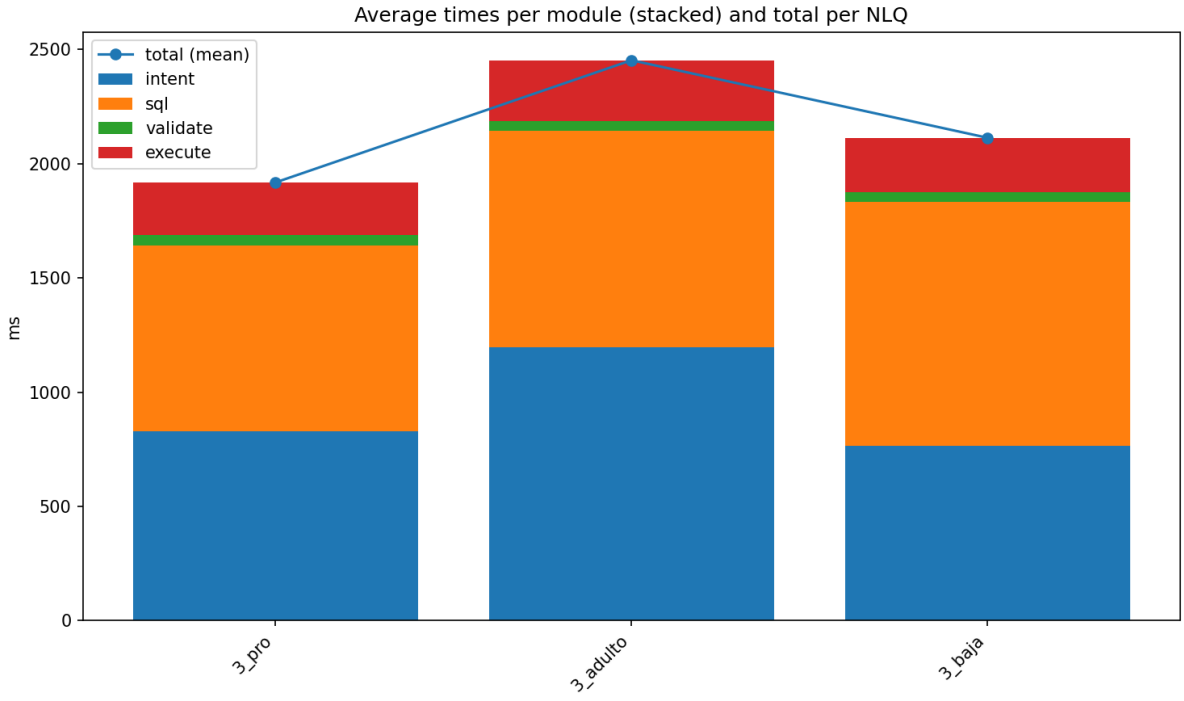

Figura II.7: Comparativa de tiempos medios por nivel para la NLQ 3.

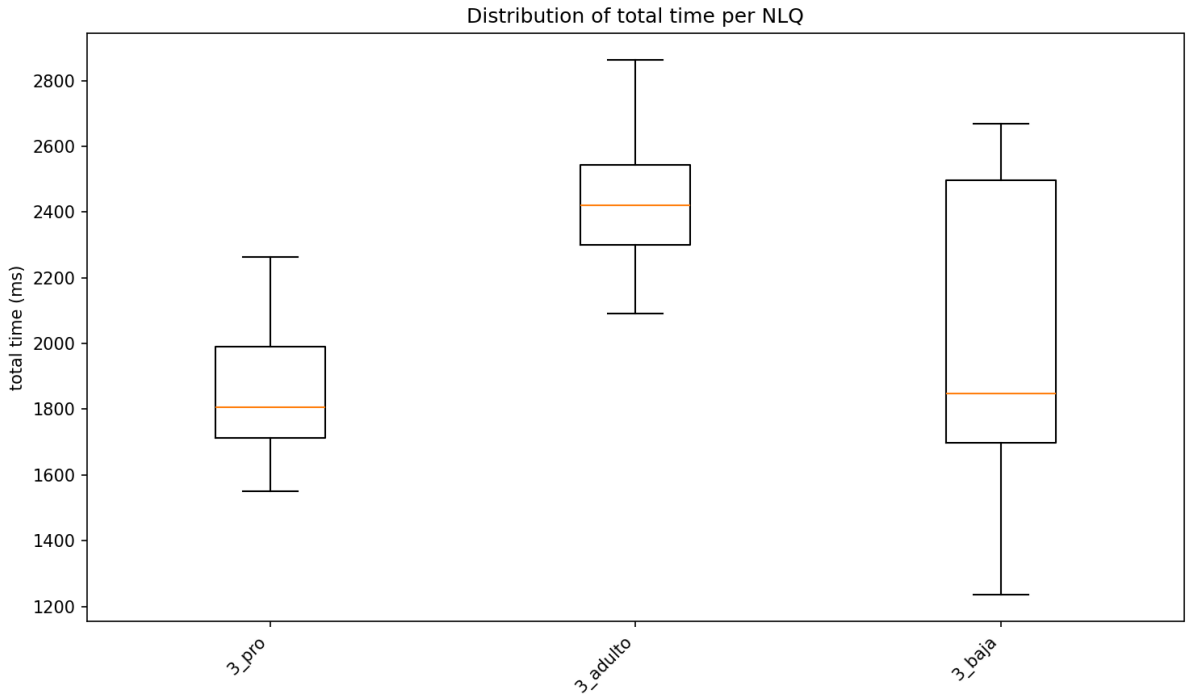

Figura II.8: Comparativa de dispersión por nivel para la NLQ 3.

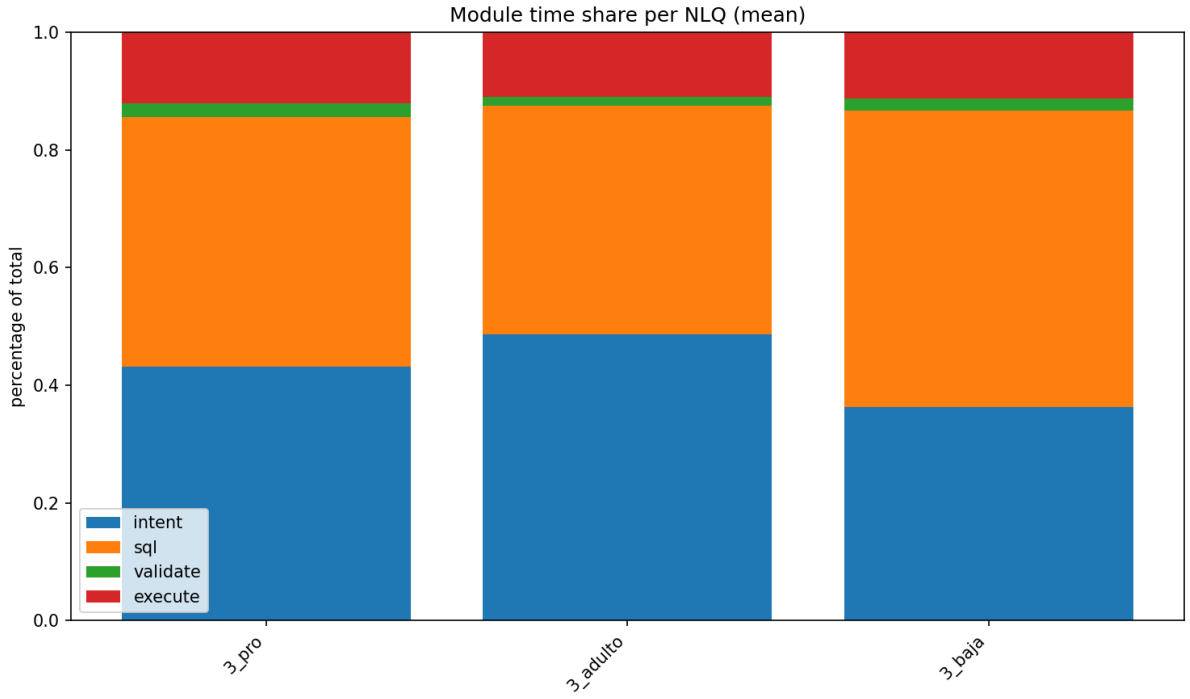

Figura II.9: Comparativa de tiempo relativo por módulo por nivel para la NLQ 3.

II.4. NLQ 4

Consulta profesional: *"List all distinct providers registered in the collar table."*
Consulta adulto: *"Which providers are there for the collars?"*
Consulta baja capacidad: *"Providers"*

Objetivo semántico: Nombrar todos los distintos proveedores de la tabla collar.

SQL esperada

```
SELECT DISTINCT "provider"
FROM collar;
```

II.4.1. Resultados

Tabla II.4: Resultados de la NLQ 4 por nivel

Nivel	Aciertos (%)	Fallos (%)	Errores (%)
Profesional	100	0	0
Adulto	100	0	0
Baja capacidad	100	0	0

II.4.2. Análisis temporal

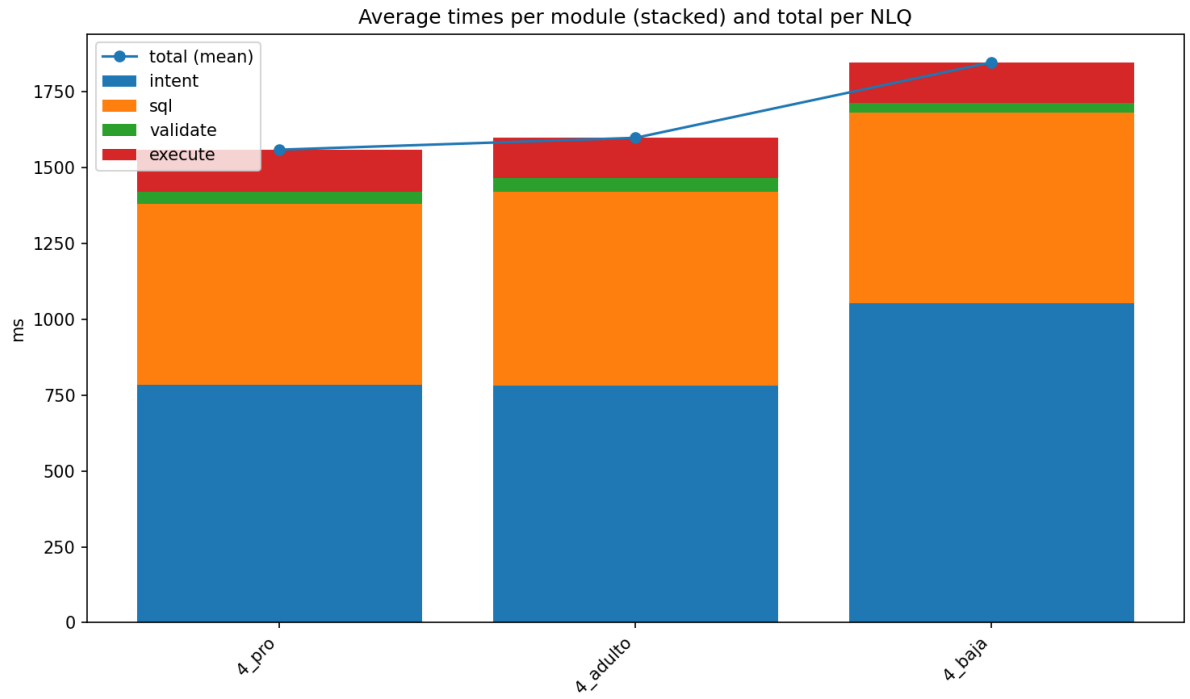

Figura II.10: Comparativa de tiempos medios por nivel para la NLQ 4.

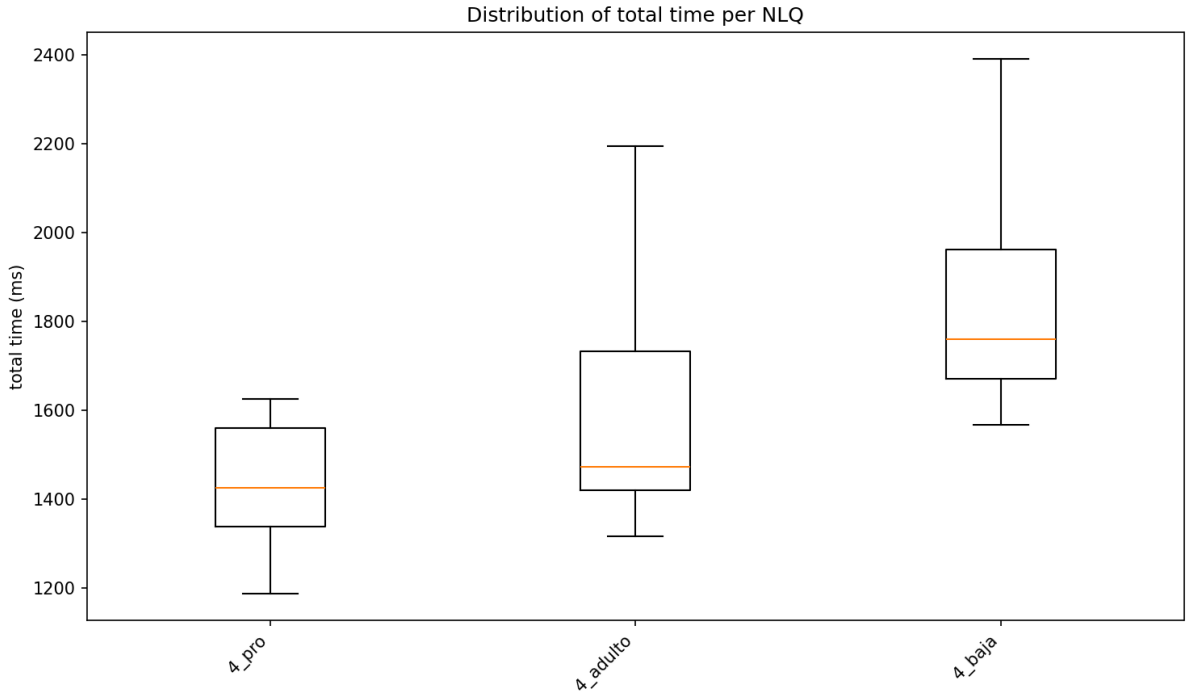

Figura II.11: Comparativa de dispersión por nivel para la NLQ 4.

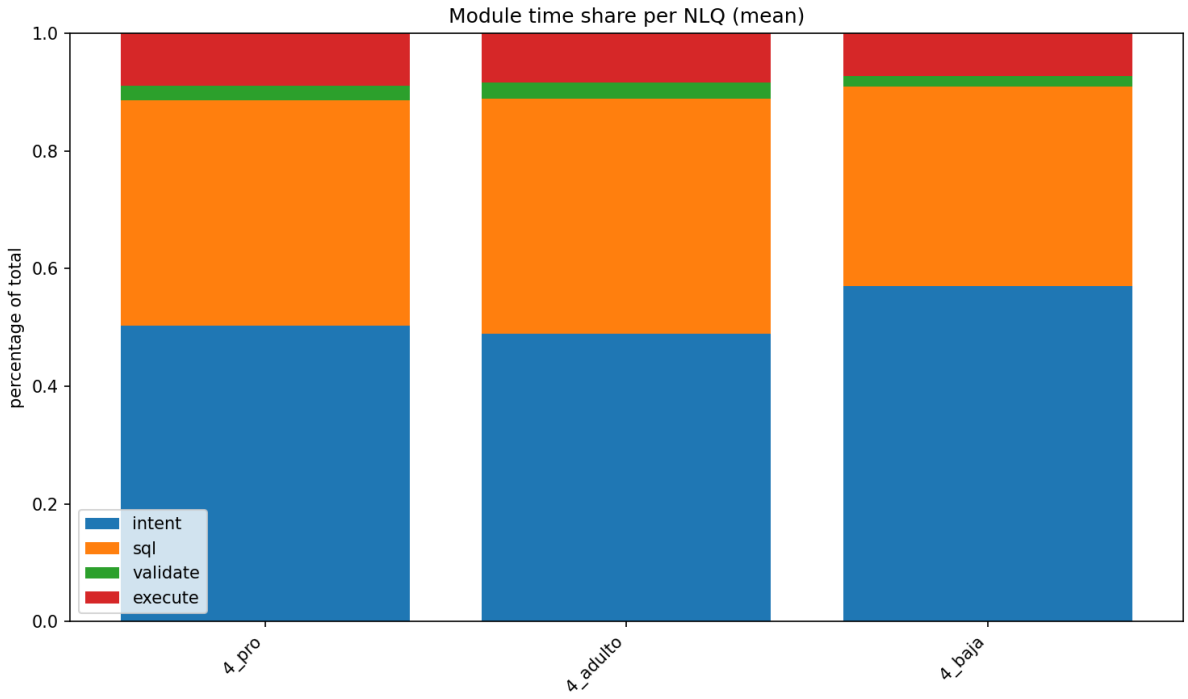

Figura II.12: Comparativa de tiempo relativo por módulo por nivel para la NLQ 4.

II.5. NLQ 5

Consulta profesional: *"Retrieve the list of distinct collars (entityName) that had resourceAlarm = true between 2021-07-21 and 2021-07-28."*
Consulta adulto: *"List the collars with an alarm since last week."*
Consulta baja capacidad: *"alarm this week"*

Objetivo semántico: Nombrar todos los collares con resourceAlarm = true entre 2021-07-21 y 2021-07-28.

SQL esperada

```
SELECT DISTINCT "entityName"
FROM collar
WHERE "resourceAlarm" = true
AND "time" BETWEEN '2021-07-21' AND '2021-07-28';
```

II.5.1. Resultados

Tabla II.5: Resultados de la NLQ 5 por nivel

Nivel	Aciertos (%)	Fallos (%)	Errores (%)
Profesional	100	0	0
Adulto	85	0	15
Baja capacidad	25	70	5

II.5.2. Análisis temporal

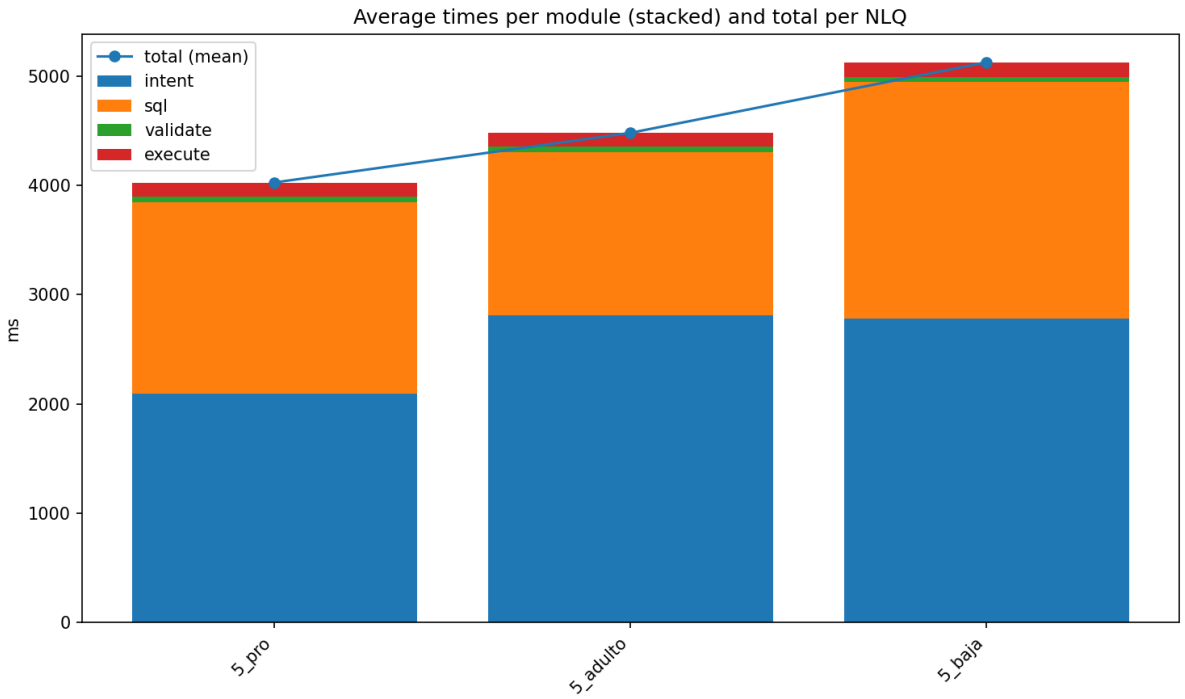

Figura II.13: Comparativa de tiempos medios por nivel para la NLQ 5.

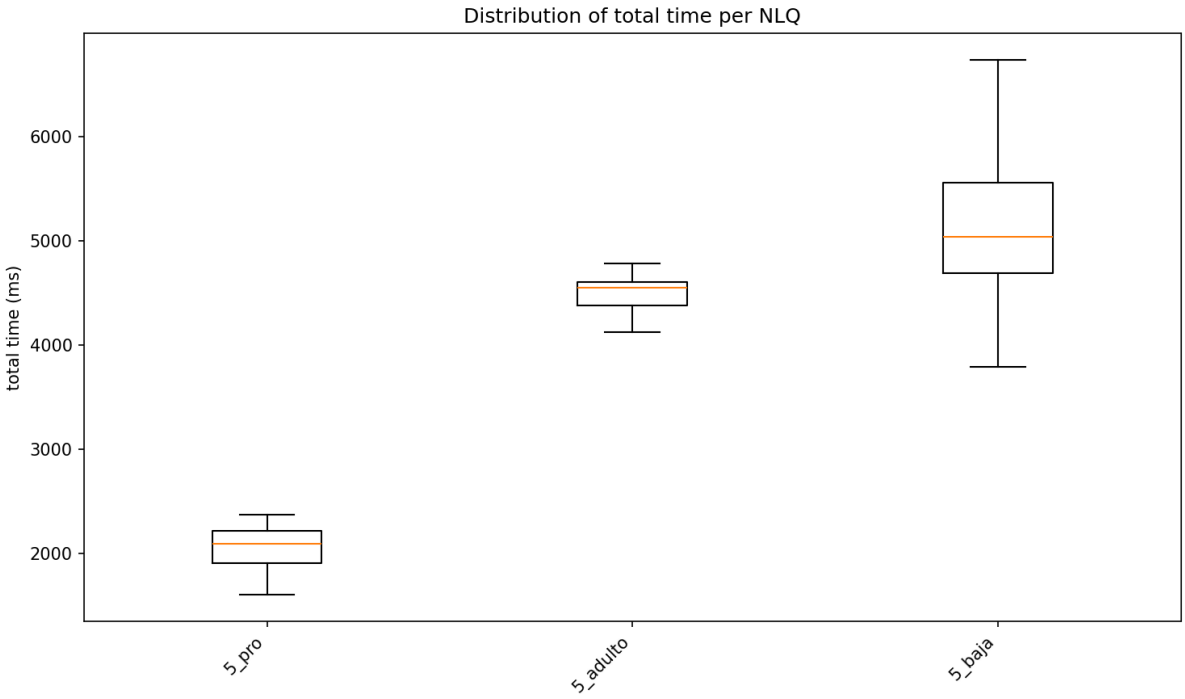

Figura II.14: Comparativa de dispersión por nivel para la NLQ 5.

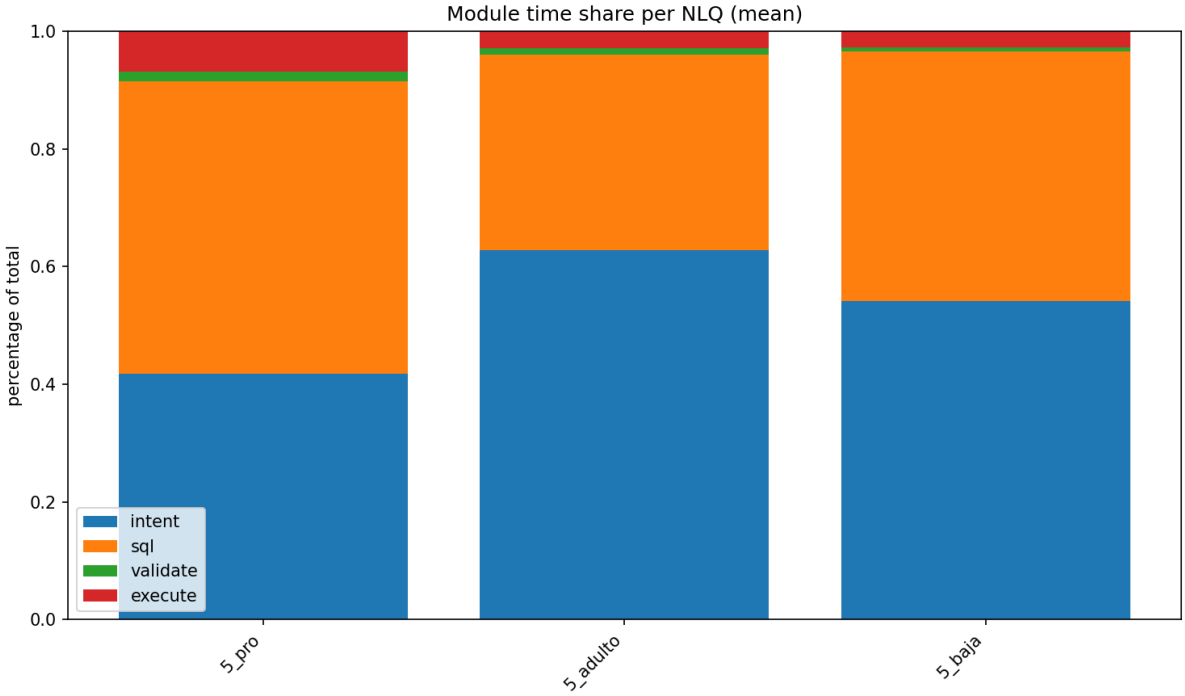

Figura II.15: Comparativa de tiempo relativo por módulo por nivel para la NLQ 5.