eDwards: Multi-agent system implementation

Gustavo Guaragna[1], Héctor Ferraro[1], Guillermo Rodriguez[1], Facundo Miglierini[1], Guillermo Burriel[1]

[1]Snoop Consulting S.R.L., Buenos Aires, Argentina
info@snoopconsulting.com

Abstract. This paper presents a multi-agent system implementation capable of assisting users with various tasks and processes. It explores previous approaches to conversational agents with the goal of streamlining interactions with processes and reducing friction with involved systems through conversational user experiences (CUX). With the advent of pre-trained Large Language Models (LLMs), new techniques and opportunities have emerged for creating these experiences, accelerating the development of specialized multi-agent systems. This paper examines the technical requirements, including the ability to handle both structured and unstructured knowledge, entity recognition, and the transformation of web forms into dialogues. It also explores different frameworks and the state of the art. Finally, it presents an architecture that leverages three agents to solve different problems, discussing challenges and lessons learned from the implementation of various conversational agent architectures.

Keywords: LLM, Agents, Agentic, Multi-agent, Conversational User Experience.

eDwards: implementación de sistema Multi-agente

Resumen. El objetivo de este trabajo es presentar la implementación de un sistema multi-agente capaz de asistir al usuario en diferentes tareas y procesos. Se repasan los primeros acercamientos hacia un agente conversacional con el fin de agilizar las interacciones con los procesos y minimizar la fricción con los sistemas involucrados a través de experiencias de usuario conversacionales (CUX). Con la irrupción de los grandes modelos de lenguaje (LLMs) pre-entrenados surgieron nuevas técnicas y oportunidades en la creación de dichas experiencias, facilitando la implementación de sistemas multi-agente especializados. Se exploran los requisitos técnicos, incluyendo el manejo de conocimiento estructurado y no estructurado, reconocimiento de entidades, y la conversión de formularios web en diálogos, así como los frameworks existentes y el estado del arte. Finalmente se presenta una arquitectura conformada por tres agentes que resuelven distintos problemas, y se discuten los desafíos y aprendizajes obtenidos al comparar diferentes modelos de agentes.

Palabras clave: LLM, Agente, Agentic, Multi-agente, Experiencia de Usuario Conversacional.

1. Introducción

1.1 Historia

In God we trust, all others must bring data. **–William Edwards Deming (1900-1993)**

Los sistemas informáticos permiten realizar diferentes tareas como comprar pasajes para viajar, realizar pagos, agendar un turno con el médico, consultar sobre el estado de una línea de producción en una planta industrial, etc. No obstante, para ello es necesario navegar por interminables menús, completar formularios, seleccionar combos, etc. Esta forma de interactuar con los sistemas genera una fricción que muchas veces impide aprovechar al máximo sus capacidades e incluso lleva a dejar de usarlos simplemente por esa fricción que se genera en la interacción con los mismos.

Estos son algunos ejemplos de solicitudes que puede realizar un usuario:

- ¿Me ayudas a cargar una no conformidad en el sistema de calidad?
- Quiero saber cuantos dias de vacaciones tengo
- Dame la lista de los proyectos en ejecución que tengan al menos un riesgo declarado

En este contexto se introduce el siguiente interrogante: ¿Es posible diseñar un asistente conversacional que ayude a resolver dichas tareas?.

Esta fue la pregunta que nos hicimos en el año 2014 y así nació eDwards: Un asistente conversacional al cual le pudiera hablar o escribir como si lo hiciera con otra persona. El nombre es en honor a William Edwards Deming, un estadístico estadounidense, profesor universitario, autor de textos, consultor y difusor del concepto de calidad total. Con la ayuda de eDwards el usuario ya no tendría que ingresar al CRM (Customer Relationship Management) para crear una nueva oportunidad comercial o ir a buscar en un archivo PDF como ejecutar un proceso determinado, simplemente se lo pediría hablando o escribiendo y el asistente haría el trabajo. La omnipresencia de eDwards en los distintos canales de comunicación será una característica distintiva, permitiendo a los usuarios utilizarlo en todo momento o lugar.

1.2 Requerimientos

Durante el proceso de elicitación se identificaron los siguientes requerimientos clave que eDwards debe cumplir:

- **Formularios web como una conversación**. Los formularios web tradicionales deberán transformarse en interacciones conversacionales, generando una experiencia más natural y user-friendly.
- **Transacciones sincrónicas y asincrónicas**. Deberá permitir tanto la ejecución inmediata de transacciones (modo sincrónico), como la posibilidad de dejar operaciones en espera o pendientes de procesamiento posterior (modo asincrónico), según se requiera.

- **Omnicanal**. eDwards debe estar disponible a través de múltiples canales de comunicación (web, móvil, mensajería, etc.), pudiendo comenzar una transacción por un canal y continuarla más tarde por otro.
- **Multimodal**. Se requiere soporte para diferentes modos de interacción, incluyendo texto, voz y eventualmente imagen o video.
- **Reflexivo**. eDwards debe ser capaz de razonar sobre su propio comportamiento, aprendiendo de las interacciones previas y ajustando sus respuestas o estrategias de diálogo con base en ese conocimiento acumulado.

Versiones previas de eDwards fueron implementadas por el equipo de Snoop Consulting utilizando diferentes plataformas de procesamiento de lenguaje natural, como IBM Watson[1], LUIS de Microsoft[2] y Dialogflow de Google[3] tanto para la detección de intenciones, entidades y modelado de las conversaciones.

La motivación del presente trabajo es mostrar cómo eDwards puede ser implementado como un sistema multi-agente basado en los nuevos modelos de lenguaje capaces de generar texto (GPT, LLaMA, etc.), aprovechando las ventajas que tienen estos respecto de las tecnologías precedentes.

2. Estado del Arte

2.1 Agentes con herramientas

A raíz del crecimiento de la inteligencia artificial en los últimos años, el concepto de agentes de IA se volvió habitual. Los Grandes Modelos de Lenguaje (LLMs), alineados con la percepción del entorno y la toma de decisiones de los agentes, muestran un gran potencial en el razonamiento y la planificación.

Si bien para tareas sencillas puede alcanzar con una simple consulta a un LLM, existen numerosos trabajos donde se propone enriquecer estos modelos con la capacidad de utilizar herramientas externas para resolver ciertas tareas y deben ser capaces de determinar cuándo deben utilizarse, permitiendo así mejorar el rendimiento, aunque a costa de una mayor latencia o incremento de los costos (Hongshen et al., 2025). Para aprovechar el potencial de estos modelos, deben ser capaces de resolver tareas en una situación de zero shot (Fang et al., 2025), es decir utilizando herramientas sin haber sido entrenados previamente para ello. El sobreuso de estas herramientas pueden llevar a los LLMs a depender demasiado de ellas, llevándolos a la toma de decisiones erróneas. Si bien estas soluciones pueden ser útiles para algunos dominios, no son suficientes para asistentes de mayor complejidad, capaces de resolver múltiples tareas de distinto tipo, como el que se propone en este trabajo.

[1] Ver https://www.ibm.com/products/natural-language-understanding. Accedido el 20/06/2025

[2] Ver https://learn.microsoft.com/en-us/azure/ai-services/luis/what-is-luis. Accedido el 20/06/2025

[3] Ver https://cloud.google.com/dialogflow/es/docs/basics?hl=es-419. Accedido el 20/06/2025

2.2 Arquitecturas multi-agente

Estos agentes, que combinan el uso de LLMs con herramientas específicas, son capaces de resolver un gran número de problemas. Sin embargo es cada vez más común utilizar arquitecturas multi-agente para la resolución de problemas más complejos, como la manipulación afectiva de imágenes, donde diferentes agentes se encargan de tareas específicas, tales como planificar, editar y realizar evaluación (Mao et al., 2025). Esta secuencia de pasos puede llevarse a cabo en diferentes dominios, por ejemplo (Li et al., 2025) propone también una arquitectura multi-agente para generar gráficos complejos, donde descompone la tarea de la creación de dichos gráficos en una colaboración iterativa entre agentes especializados.

En las arquitecturas de sistemas multi-agente, se pueden identificar diferentes formas de conexión entre los agentes. Por ejemplo, pueden existir redes donde cada agente tiene la capacidad de comunicarse con los demás, o implementaciones en las que un agente supervisor determina qué subagente será responsable de completar una tarea específica. Esta última arquitectura puede ser generalizada mediante la introducción de una estructura jerárquica, en la cual un agente supervisor colabora con un subconjunto de agentes supervisores, lo que permite la gestión de flujos de control más complejos.

Independientemente de la arquitectura de sistemas multi-agente adoptada, uno de los aspectos cruciales a considerar es la comunicación interna entre los agentes y entre un agente particular y sus herramientas. La comunicación entre agentes puede resolverse mediante el uso de estados compartidos que implementan interfaces comunes. Por otro lado, la comunicación entre agentes y herramientas puede llevarse a cabo utilizando tool-calling, es decir la capacidad de los modelos de inteligencia artificial para interactuar con herramientas externas, APIs (Application Programming Interface) o sistemas para mejorar sus funciones[4].

En el caso de estudio aquí presentado, se ha optado por una arquitectura con un supervisor único, dado que se quería centralizar la coordinación de los cuatro agentes implementados, los cuales resuelven tareas de forma atómica sin requerir de la asistencia de otro agente. Este supervisor mantiene el control sobre los agentes especializados en diferentes tareas, utilizando estados compartidos entre el supervisor y los subagentes, los cuales implementan tool-calling para la gestión de herramientas.

Existen diversas herramientas para el desarrollo de arquitecturas multi-agente, las cuales permiten representar las interacciones entre los agentes mediante grafos, donde los nodos corresponden a los agentes y las aristas representan las comunicaciones entre ellos. En el presente trabajo, se utilizará LangGraph[5], una herramienta que facilita la creación y gestión de arquitecturas multi-agente, para desarrollar un sistema en el que varios agentes colaboran para resolver tareas específicas dentro de un sistema.

[4] Ver https://www.ibm.com/think/topics/tool-calling. Accedido el 12/06/2025
[5] Ver https://www.langchain.com/langgraph. Accedido el 07/04/2025.

3. Arquitectura de sistema multi-agente eDwards

En este apartado se presentará la arquitectura de grafo utilizada en la implementación de los sistemas multi-agente. Una vez explicada, se expondrá la implementación de *eDwards* bajo dicha arquitectura.

3.1 Agentes como grafos

Los distintos agentes pueden ser implementados de diversas maneras. Por ejemplo, pueden utilizar la lógica ReAct, en la cual el propio LLM formula un plan de trabajo basado en un subconjunto de herramientas disponibles (Yao et al., 2022). Aunque esta implementación ha demostrado ser muy eficaz para resolver problemas complejos, no está exenta de errores debido a la naturaleza generativa de los LLM. Por ello, es común encontrar implementaciones de agentes que funcionen como flujos de trabajo ya que ofrecen predictibilidad y consistencia para tareas bien definidas[6].

Este tipo de agentes con flujos de trabajo definidos puede implementarse mediante grafos dirigidos, donde el grafo G cuenta con un conjunto de nodos N, cuyos elementos se encargan de ejecutar una función que resuelve un problema en particular dentro del agente. Los nodos reciben información como entrada y generan información como salida, y se encuentran conectados por aristas unidireccionales que determinan el flujo de ejecución del agente. Notar que a partir de un punto de entrada determinado, el grafo se ejecuta en orden topológico para resolver una tarea, y la respuesta final del agente se corresponde con la salida del último nodo ejecutado.

3.2 eDwards como grafo multi-agente

A continuación se hablará sobre la implementación del sistema eDwards como un sistema multi-agente (ver Figura 1) basado en LLMs diseñado a partir de la arquitectura de grafo explicada previamente.

Para llevar a cabo un diseño de estas características se eligió LangGraph como framework de desarrollo, ya que facilita el diseño de agentes mediante grafos, proveyendo un conjunto de elementos que permite diseñar nodos que transfieren su información por medio de estados y se conectan por medio de aristas de forma legible y eficiente a partir de streams de datos.

Como se mencionó en los apartados anteriores, eDwards se enfoca principalmente en la resolución de tres problemas independientes: la generación de respuestas fundamentadas en el contenido de un conjunto de documentos a los que el sistema tiene acceso, la obtención de información almacenada en una base de datos SQL (Structured Query Language) a partir de consultas realizadas en lenguaje natural por el usuario, y el registro de información en distintos sistemas reemplazando la carga de formularios web en base a la extracción de la información recibida en una consulta. Estas tareas son resueltas por un agente RAG (Retrieval-Augmented Generation), un agente SQL y un agente transaccional, respectivamente.

[6] Ver https://www.anthropic.com/engineering/building-effective-agents. Accedido el 12/06/2025

Además se suma un cuarto agente, denominado agente de asistencia, que informa las capacidades que el mismo eDwards posee, a fin de aportar una guía durante la interacción del usuario.

Es importante destacar que se optó por implementar agentes especializados con el objetivo de que los mismos actúen como sistemas aislados, de manera que las responsabilidades de uno no interfiera con las del resto y viceversa. Esta arquitectura se impuso sobre su contraparte compuesta por un único agente que dispone de las herramientas suficientes para resolver todos los problemas, cuyos resultados de rendimiento se expondrán más adelante.

Debido a que se dispone de cuatro agentes diferentes, el primer desafío de eDwards consiste en poder identificar cuál es el agente que debe atender una petición dada. Para esto, se implementó un nodo que actúa como punto de entrada del grafo, llamado *supervisor*, que recibe la petición del usuario como entrada y responde con el nombre del agente responsable de atenderla. Posteriormente, el grafo determina qué agente invocar por medio de *aristas condicionales* vinculadas a cada uno de ellos, cuya tarea consiste en evaluar la respuesta generada por el supervisor y activarse cuando el agente asociado es elegido.

Concretamente, los agentes son grafos cuyos nodos de entrada son establecidos como nodo destino de las aristas condicionales, lo que provoca que la invocación del grafo principal del eDwards dirija su flujo de ejecución hacia una implementación de un subgrafo especializado para resolver la tarea solicitada.

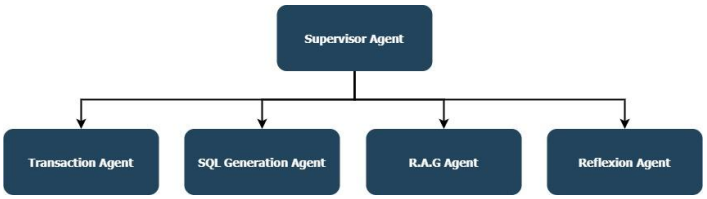

Figura 1. Arquitectura multi-agente eDwards.

A continuación se realiza una breve descripción de los nodos que comprenden cada uno de los cuatro subgrafos que constituyen los agentes especializados.

Agente RAG. Se compone de nodos para buscar contenido en las fuentes de información provistas, calificar la importancia de los documentos a la hora de resolver la consulta, mejorar dicha consulta para facilitar la búsqueda sobre los documentos, decidir si continuar mejorando la consulta de forma iterativa, identificar si la respuesta generada es producto de una alucinación y generar una respuesta final. La identificación de alucinaciones se realiza mediante un nodo específico que compara la respuesta generada con las fuentes de información recuperadas y, si detecta inconsistencias o información no respaldada, puede activar un proceso de revisión o regeneración. En la Figura 2, puede verse el grafo asociado a este agente.

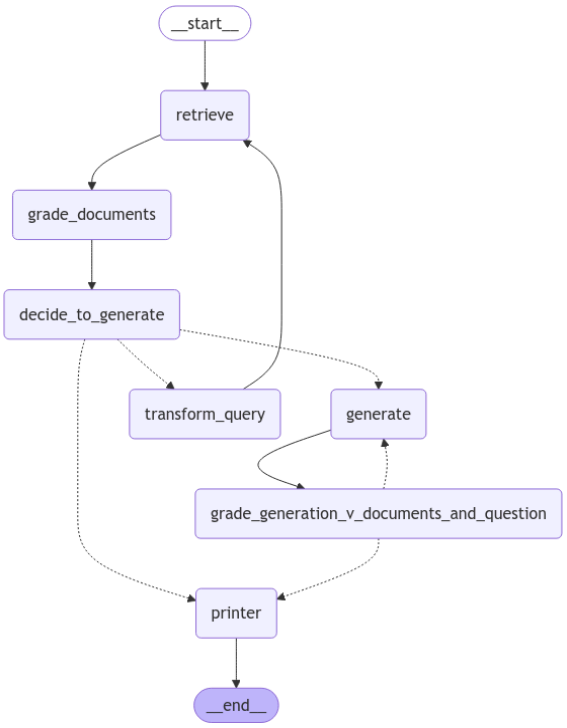

Figura 2. Estructura de grafo del agente RAG. La consulta del usuario ingresa al nodo "retrieve", donde se busca información a partir de una base de datos vectorial. Luego, se evalúan alucinaciones mediante los nodos "grade_documents" y "decide_to_generate". Si se detectan alucionaciones, el nodo "transform_query" mejora la consulta inicial para buscar nuevamente en la base de datos. En caso contrario, el nodo "generate" redacta una respuesta, que nuevamente es evaluada, pero esta vez por el nodo "grade_generation_v_documents_and_question". Se genera un bucle entre los dos últimos nodos hasta que el agente considera que el resultado es apropiado. Así, el nodo "printer" devuelve la respuesta final al usuario.

Agente transaccional. Está formado por nodos que sirven para extraer la información relevante del mensaje, validar el resultado conforme a las reglas de negocio, ejecutar una transacción y proveer feedback al usuario ya sea para la corrección de algún error o para solicitar la confirmación de una transacción. En la Figura 3 puede verse el grafo asociado a este agente.

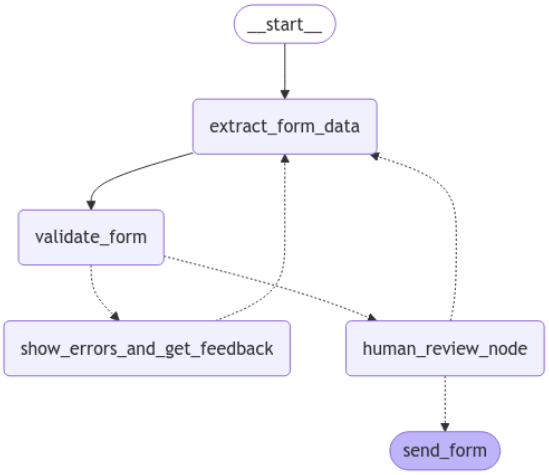

Figura 3. Estructura de grafo del agente transaccional. La petición del usuario ingresa al nodo “extract_form_data”, encargado de extraer la información relevante para la transacción. Dicha información se valida en el nodo “validate_form”. Si es válida, el nodo “human_review_node” le pide al usuario una confirmación de la transacción, que luego es efectuada por el nodo “send_form”. En caso contrario, el nodo “show_errors_and_get_feedback” muestra un mensaje al usuario que contiene los errores de validación detectados, y solicita la información requerida.

Agente reflexivo. cuenta con un único nodo que conoce las capacidades de eDwards y es capaz de reflexionar sobre las mismas para poder guiar al usuario, a través de lenguaje natural, en la utilización del sistema.

Agente SQL. Consta de nodos para listar las tablas disponibles en la base de datos, consultar el schema de las tablas relevantes a la consulta del usuario, desambiguar sustantivos propios que potencialmente pudo haber ingresado el usuario incorrectamente, generar consultas SQL, verificar que las consultas sean correctas sintácticamente y ejecutarlas. Si bien estos nodos trabajan de forma secuencial en el orden en que fueron presentados, a la hora de ejecutar una consulta el agente evalúa si la respuesta obtenida resuelve la petición del usuario. De no ser así, se ejecuta el grafo iterativamente hasta lograr la respuesta esperada. Para evitar bucles largos que afecten negativamente la experiencia del usuario debido al tiempo de ejecución implicado, se estableció una cantidad máxima de iteraciones permitida. En caso de que se alcance dicho máximo y no se encuentre una respuesta, el agente informa que no se ha logrado encontrar la información solicitada. En la Figura 4, puede verse el grafo asociado a este agente.

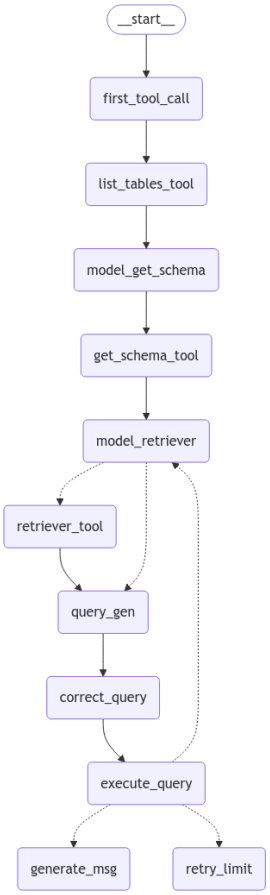

Figura 4. Estructura de grafo del agente SQL. El agente comienza solicitando las tablas relacionadas a la consulta y su esquema, recurriendo a los nodos “first_tool_call”, “list_tables_tool”, “model_get_schema” y “get_schema_tool”. Posteriormente, los nodos “model_retriever” y “retriever_tool” corrigen aquellos sustantivos propios que el agente considere que fueron ingresados incorrectamente. El nodo “query_gen” genera una consulta SQL que luego es corregida sintácticamente por el nodo “correct_query”, especializado de acuerdo a la base de datos utilizada. Una vez validada la consulta SQL, “execute_query” la ejecuta y verifica si logra responder la petición del usuario. En tal caso, el nodo “generate_msg” genera la respuesta final, con la consulta generada y los datos que ella retorna. En caso contrario, si se excedió el límite de reintentos, el nodo “retry_limit” informa que no se pudo resolver la petición del usuario. Si dicho límite no fue excedido, el flujo de ejecución vuelve al nodo “model_retriever” para intentar generar la consulta SQL nuevamente.

4. Arquitectura de solución

A continuación, se describen los componentes que conforman la arquitectura completa de la solución propuesta (ver Figura 5), diseñada tomando como referencia los requerimientos detallados en las secciones anteriores.

- **Canales.** Representan los distintos medios a través de los cuales un usuario puede iniciar o mantener una conversación con el asistente conversacional.
- **Orquestador.** Es un web service encargado de gestionar la comunicación entre los diversos canales y el asistente mediante una API. Además, implementa la lógica necesaria para coordinar el correcto funcionamiento de los distintos agentes, resolviendo las peticiones del usuario mediante la ejecución de las herramientas adecuadas. Este componente ha sido desarrollado utilizando Python.
- **Asistente multi-agente.** Es el elemento central de la solución propuesta (ver sección "Enfoque"). Utiliza Grandes Modelos de Lenguaje (como GPT-4) y una arquitectura basada en grafos dirigidos para proporcionar respuestas precisas y relevantes a las peticiones del usuario.
- **Data Sources.** Son los recursos empleados para recuperar información y generar respuestas con contexto relevante según las consultas del usuario. En el caso específico de información documental, se utilizó una base de datos vectorial basada en embeddings generados con el modelo *text-embedding-ada-002*. Por otro lado, para información estructurada o tabulada, se optó por utilizar BigQuery, con propósitos experimentales.
- **MLOps.** Se incorpora la plataforma LangSmith[7], que facilita el monitoreo continuo del ciclo de vida completo de aplicaciones basadas en LLMs.

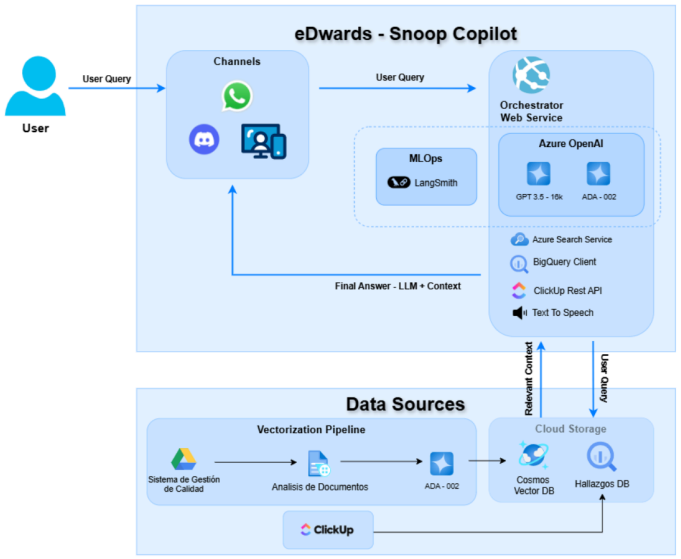

Figura 5. Arquitectura del asistente con multi-canalidad.

[7] Ver https://www.langchain.com/langsmith. Accedido el 07/04/2025.

5. Evaluación

Con el fin de evaluar los diferentes enfoques propuestos y validar las posibles ventajas de un asistente conversacional basado en una arquitectura multi-agente, se crearon varios conjuntos de datos (datasets) de evaluación específicos. Dichos datasets constan de preguntas hipotéticas realizadas por usuarios, así como de sus respuestas esperadas, que sirven como Ground Truth para la evaluación. El dataset final comprende 70 preguntas diseñadas para medir distintas capacidades del agente: responder con contexto relevante mediante técnicas de Retrieval-Augmented Generation, formular consultas SQL correctas sobre bases de datos relacionales, y evaluar correctamente qué herramientas deben invocarse para resolver adecuadamente la solicitud del usuario (Reflexión). Para esto, se comparan tres implementaciones diferentes de agentes.

- **Agente básico.** Consiste en un agente basado en un LLM cuyo comportamiento se define mediante prompting. Aunque este agente es capaz de invocar un conjunto predeterminado de herramientas (toolkit) para resolver las tareas planteadas, carece de un proceso explícito de planificación y razonamiento.
- **Agente ReAct.** Corresponde a un agente basado en un LLM que incorpora explícitamente la lógica ReAct, proporcionándole capacidades de razonamiento y planificación. Este agente utiliza el mismo conjunto de herramientas disponibles para el Agente Básico.
- **Multi-Agente.** Se trata de una implementación que sigue la arquitectura propuesta detalladamente en la sección dedicada al enfoque. Este modelo cuenta con agentes especializados para cada uno de los problemas planteados, coordinados mediante un agente principal encargado de gestionar y supervisar las interacciones y cooperación entre subagentes.

El proceso de evaluación consistió en medir la precisión de las respuestas generadas por cada una de estas implementaciones, comparándolas con el Ground Truth elaborado previamente. Esta comparación es llevada a cabo por un LLM externo que ha sido dotado de capacidades de evaluación, en lo que se conoce como *LLM-as-a-judge*. De esta manera, es posible automatizar la comparación al tiempo que se mantiene un criterio de evaluación uniforme. Para asegurar la fiabilidad de los resultados, cada implementación fue evaluada mediante 10 ejecuciones consecutivas.

Las 70 preguntas del dataset fueron diseñadas para cubrir un rango diverso de complejidad y tipo de consulta, incluyendo:

Preguntas RAG (26 preguntas): Consultas que requieren la recuperación de información no estructurada de documentos, como "¿Qué es un hallazgo?" o "¿Cuáles son las políticas de calidad de la empresa?". También se incluyeron preguntas para las cuales el agente debía ser capaz de reconocer que no cuenta con información suficiente para responder correctamente, como por ejemplo "¿En qué liga de fútbol juega Messi?".

Preguntas de Generación SQL (25 preguntas): Consultas que implican la traducción de lenguaje natural a sentencias SQL para bases de datos estructuradas, por ejemplo, "¿Quién es el usuario con más tareas cerradas del área Sistemas Internos?" o "¿Cuáles son los 3 usuarios con más hallazgos cargados?", "¿Cuántos hallazgos creados suman las 5 personas que más hallazgos tienen creados?".

Preguntas de Reflexión (20 preguntas): Consultas sobre las capacidades del propio asistente, como "Cuáles son tus funcionalidades?" o "¿Qué podés hacer?".

En la Tabla 1 se presenta un resumen comparativo basado en los valores promedio de exactitud (accuracy) alcanzados por cada agente durante estas simulaciones. Se presenta también, en la Tabla 2, la comparación del tiempo que demoró cada enfoque y la cantidad de tokens utilizados hasta generar la respuesta final, en la Tabla 3.

Tabla 1. Comparación de los diferentes enfoques

Evaluación	Agente Básico	Agente ReAct	Multi-Agente
Reflexión	0.68	0.74	**1**
RAG	0.77	0.81	**1**
Generación SQL	0	0.18	**0.77**

Los resultados de exactitud demuestran la superioridad del enfoque Multi-Agente, especialmente en tareas de Reflexión y RAG, donde alcanzó una exactitud perfecta (1.0). En la Generación SQL, si bien no fue perfecto, mostró una mejora sustancial (0.77) en comparación con el Agente ReAct (0.18) y el Agente Básico (0), que no fue capaz de resolver ninguna consulta SQL dado que el modelo no logró invocar correctamente las herramientas necesarias para recuperar el esquema de la base de datos, lo que llevó a la generación de consultas SQL incorrectas en todos los casos.

Tabla 2. Latencia promedio, en segundos

Evaluación	Agente Básico	Agente ReAct	Multi-Agente
Reflexión	5.94	**2.42**	10.03
RAG	7.22	**5.95**	10.34
Generación SQL	-	**6.68**	12.74

La tabla de latencia promedio muestra que, si bien el Multi-Agente ofrece mayor exactitud, introduce mayores niveles de latencia. Esto es atribuible a los cambios de estado internos y a la coordinación entre los nodos dentro del grafo asociado a cada agente especializado, así como al proceso de selección del agente supervisor, que añaden sobrecarga computacional necesaria para producir la respuesta final. Por ejemplo, en el agente SQL, las iteraciones para refinar la consulta (model_get_schema, model_retriever, query_gen, correct_query) contribuyen a un mayor tiempo de ejecución, a pesar de su mejora en la precisión.

Tabla 3. Cantidad de tokens consumidos

Evaluación	Agente Básico	Agente ReAct	Multi-Agente
Reflexión	1055866	1043971	**94479**
RAG	629382	568786	**37131**
Generación SQL	-	1527676	**93128**

En cuanto a la cantidad de tokens consumidos, el enfoque Multi-Agente demuestra ser significativamente más eficiente. Esto se debe a la capacidad de los agentes

especializados de enfocar su "ventana de contexto" y utilizar sus herramientas de manera más precisa para descomponer y resolver la consulta original. En contraste, los agentes Básico y ReAct, al intentar resolver problemas de múltiples dominios con un único toolkit y sin la misma granularidad de especialización, a menudo requieren más tokens para el razonamiento y la generación de la respuesta final.

Los mediciones de latencia en la generación de la respuesta y la cantidad de tokens consumidos fueron obtenidos de las trazas de ejecución generadas en Langsmith como parte de la automatización de la evaluación. En la figura 7 se ve un ejemplo para una ejecución del agente SQL (llamado BIGQUERY_AGENT en la plataforma). En el caso del Agente Básico, no se consideran válidos para la generación de sentencias SQL , ya que no fue capaz de resolver lo requerido.

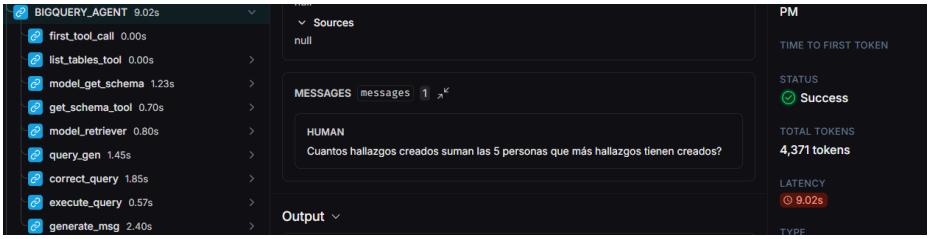

Figura 7. Ejemplo de traza de ejecución de un caso de evaluación.

6. Conclusiones y trabajo futuro

En este trabajo se presentaron los motivos por los que la implementación de un sistema conversacional puede resultar beneficiosa para la experiencia de los usuarios en los procesos diarios que debe llevar a cabo o mejorar la adopción y la experiencia global con el uso de diferentes sistemas como lo puede ser pedir un turno médico, comprar un pasaje para viajar, entre otras. A partir de esta motivación inicial, se realizó el relevamiento de requerimientos que un sistema conversacional debe satisfacer, repasando el estado del arte hasta llegar a una propuesta concreta de implementación de un sistema multi-agentes basados en grafos.

Para validar dicha propuesta se planteó un experimento orientado a evaluar la capacidad del sistema de responder de forma asertiva y precisa, comparándola con técnicas habituales de resolución de este tipo de tareas, como lo son utilizar LLMs con un conjunto de herramientas o agentes simples pero con capacidades de razonamiento y planificación.

Los resultados muestran que la implementación de una arquitectura multi-agente permite obtener respuestas más correctas y efectivas, favoreciendo además un uso eficiente de tokens gracias a su capacidad de resolver problemas mediante agentes especializados. Por otro lado, se observó que la generación de respuestas introduce mayores niveles de latencia, atribuibles a los cambios de estado internos que ocurren en los nodos dentro del grafo asociado a cada agente especializado, necesarios para producir la respuesta final.

En particular se destaca la capacidad del sistema multi-agente propuesto para resolver satisfactoriamente consultas en lenguaje natural que requieren ser traducidas a sentencias SQL. Aunque está demostrado que los LLMs cuentan con capacidades

para generar sentencias SQL correctas al proporcionarles información suficiente, la tarea se dificulta cuando se debe seleccionar y recuperar dinámicamente un conjunto de tablas apropiado, o al momento de desambiguar términos utilizados por el usuario como pueden ser nombres incompletos o apodos informales, para realizar búsquedas precisas en bases de datos donde los usuarios se encuentran cargados con nombre y apellido completo.

En una arquitectura donde los agentes son altamente especializados, esto se vuelve más sencillo para el modelo ya que puede aprovechar mejor sus herramientas y su ventana de contexto para descomponer la consulta original (en lenguaje natural) sin ser sobrepasado como en el caso de un LLM sin capacidades adicionales.

Si bien este caso fue planteado como una mejora de un Sistema de Gestión de Calidad, la arquitectura propuesta puede generalizarse para la resolución de formularios donde el usuario deba cargar información estructurada (Agente Transaccional) o para resolver consultas de conocimientos específicos en cualquier dominio (Agente RAG y Agente SQL) mediante la recuperación de información almacenada, ya sea de forma estructurada o no.

Como líneas de trabajo a futuro queda pendiente la utilización de modelos que incorporan capacidades de razonamiento (OpenAI o1, DeepSeek-R1, Anthropic Claude 3.7, entre otros) y técnicas incipientes de comunicación entre los modelos y sus herramientas y fuentes de datos como lo puede ser el Model Context Protocol (MCP). Se contempla además avanzar sobre la arquitectura para incorporar capacidades de API Discovery, dotando al sistema con la capacidad de descubrir y entender una API Rest de forma tal que pueda utilizarla sin estar desarrollado explícitamente para ello; y la incorporación de multimodalidad para que pueda asistir a los usuarios a través del entendimiento de voz, imágenes y videos, incluyendo la capacidad de responder por los mismos medios.

7. Referencias

1. Walton, M. (1986). The Deming management method; Perigee.
2. William Edwards Deming, https://es.wikipedia.org/wiki/William_Edwards_Deming, last accessed 2025/04/07.
3. Deming, E. (1964). Some Remarks on Recent Advances in the Statistical Control of Quality in Japan
4. Fang, W., Zhang, Y., Qian, K., Glass, J., & Zhu, Y. (2025). PLAY2PROMPT: Zero-shot Tool Instruction Optimization for LLM Agents via Tool Play.
5. Xu, H., Wang, Z., Zhu, Z., Pan, L., Chen, X., Chen, L., & Yu, K. (2025). Alignment for Efficient Tool Calling of Large Language Models.
6. Mao, Q., Hu, H., He, Y., Gao, D., Chen, H., & Jin, L. (2025). EmoAgent: Multi-Agent Collaboration of Plan, Edit, and Critic, for Affective Image Manipulation.
7. Li, B., Wang, Y., Gu, J., Chang, K., & Peng, N. (2025). METAL: A Multi-Agent Framework for Chart Generation with Test-Time Scaling. *ArXiv, abs/2502.17651*.
8. Wu, Q., Bansal, G., Zhang, J., Wu, Y., Li, B., Zhu, E., Jiang, L., Zhang, X., Zhang, S., Liu, J., Awadallah, A.H., White, R.W., Burger, D., & Wang, C. (2023). AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversation.
9. Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., & Cao, Y. (2022). ReAct: Synergizing Reasoning and Acting in Language Models. *ArXiv, abs/2210.03629*.
10. LangGraph Docs, https://langchain-ai.github.io/langgraph/, last accessed 2025/04/07.
11. Uber blog, https://www.uber.com/blog/query-gpt/, last accessed 2025/04/07.
12. Yin, W., Hay, J., & Roth, D. (2019). Benchmarking Zero-shot Text Classification: Datasets, Evaluation and Entailment Approach. ArXiv, abs/1909.00161.
13. Liu, B., Li, X., Zhang, J., Wang, J., He, T., Hong, S., Liu, H., Zhang, S., Song, K., Zhu, K., Cheng, Y., Wang, S., Wang, X., Luo, Y., Jin, H., Zhang, P., Liu, O., Chen, J., Zhang, H., Yu, Z., Shi, H., Li, B., Wu, D.M., Teng, F., Jia, X., Xu, J., Xiang, J., Lin, Y., Liu, T., Liu, T., Su, Y., Sun, H., Berseth, G., Nie, J., Foster, I.T., Ward, L.T., Wu, Q., Gu, Y., Zhuge, M., Tang, X., Wang, H., You, J., Wang, C., Pei, J., Yang, Q., Qi, X., & Wu, C. (2025). Advances and Challenges in Foundation Agents: From Brain-Inspired Intelligence to Evolutionary, Collaborative, and Safe Systems.