serquo

Whitepaper

GenAiA | Serquo

Agente IA Conversacional

Hacia una IA Confiable: aprendizajes en la construcción, testing y validación de un agente conversacional

GenAiA

En el marco de su compromiso con la difusión del conocimiento aplicado en inteligencia artificial generativa, la asociación GenAiA ha impulsado, junto a Serquo, una iniciativa colaborativa orientada a demostrar el potencial real de los agentes IA en entornos organizativos.

Como miembro fundador de la asociación, Serquo ha contribuido pro bono aportando tecnología, metodología y talento especializado para el diseño, construcción, testing y validación de un agente conversacional avanzado desarrollado específicamente para GenAiA.

Este proyecto representa uno de los principios fundacionales de la asociación y sus miembros: fomentar la colaboración entre profesionales y empresas para acelerar la adopción práctica de la IA mediante casos de uso tangibles, compartiendo aprendizajes, capacidades y buenas prácticas que permitan acercar esta tecnología al ecosistema empresarial y a la comunidad.

1. Introducción 4
2. Motivación 5
3. Impacto esperado 6
4. Arquitectura de la aplicación 7
5. Arquitectura del agente 8
6. Evaluación del ChatBot 10
6.1 Modelo de evaluación 11
6.2 Tipos de tests 11
6.3 Métricas 11
7. Problemas encontrados 12
7.1 Gestión de Ambigüedad en el Flujo de Citas 12
7.2 Optimización de la Capa de Seguridad (Guardrails) 13
7.3 Precisión en la Clasificación de Intenciones 14
8. Ámbito de aplicación 15
9. Limitaciones 16
10. Conclusión 17

1. Introducción

Este documento describe el diseño e implementación de un agente conversacional basado en RAG (Retrieval-Augmented Generation) desarrollado para GenAiA, una asociación profesional centrada en inteligencia artificial generativa. El objetivo del ChatBot es responder preguntas y recomendar artículos de la asociación detectando en todo momento el interés de usuario y en caso de detectar un interés alto en convertirse en miembro de la asociación se concreta una cita con un experto.

El enfoque adoptado se aleja deliberadamente de los chatbots tradicionales. En lugar de un sistema basado exclusivamente en prompts, se ha construido una arquitectura agéntica con estado explícito, enrutamiento determinista y separación estricta de responsabilidades.

2. Motivación

Con esta aplicación se quiere facilitar el acceso a la información de la asociación GenAiA en cuanto a contenido y en el proceso de afiliación. Los usuarios dependían del equipo humano para resolver dudas básicas o iniciar cualquier interacción relevante, lo que generaba tiempos de respuesta elevados y una carga innecesaria sobre el equipo.

Este contexto imponía varios requisitos no negociables. Por un lado, el sistema debía ser capaz de resolver consultas utilizando exclusivamente el conocimiento interno, evitando cualquier dependencia de fuentes externas que pudiera comprometer la veracidad de las respuestas. Por otro, debía gestionar flujos conversacionales completos, como la recogida de datos para citas, sin perder información entre turnos. Finalmente, debía ser capaz de detectar señales de intención del usuario para intervenir en el momento adecuado con una propuesta de valor, sin resultar intrusivo.

Un sistema RAG convencional no cumple estos requisitos. Su naturaleza lineal —consulta, recuperación, generación— funciona bien en escenarios simples, pero falla cuando la interacción requiere memoria, control de flujo o decisiones condicionadas por el estado de la conversación. Además, delegar todo el comportamiento en un único prompt introduce un nivel de imprevisibilidad incompatible con un entorno de producción.

La motivación de esta arquitectura es precisamente resolver estas limitaciones: introducir control donde antes había heurística implícita, y estructura donde antes había dependencia total del modelo.

3. Impacto esperado

La introducción de este agente no se limita a mejorar la experiencia del usuario; redefine la forma en que la organización gestiona la interacción digital.

En primer lugar, reduce de forma significativa la carga de soporte manual. El sistema absorbe consultas frecuentes y tareas repetitivas, permitiendo que el equipo humano se centre en interacciones de mayor valor. Este efecto no es solo cuantitativo, sino cualitativo: las respuestas son consistentes, inmediatas y alineadas con la documentación oficial.

En segundo lugar, introduce un mecanismo de captación más eficiente. El agente no presenta una llamada a la acción de forma arbitraria, sino que evalúa continuamente la conversación y detecta el momento en el que el usuario muestra suficiente interés. Este enfoque permite equilibrar conversión y experiencia, evitando tanto la pasividad como la sobreexposición.

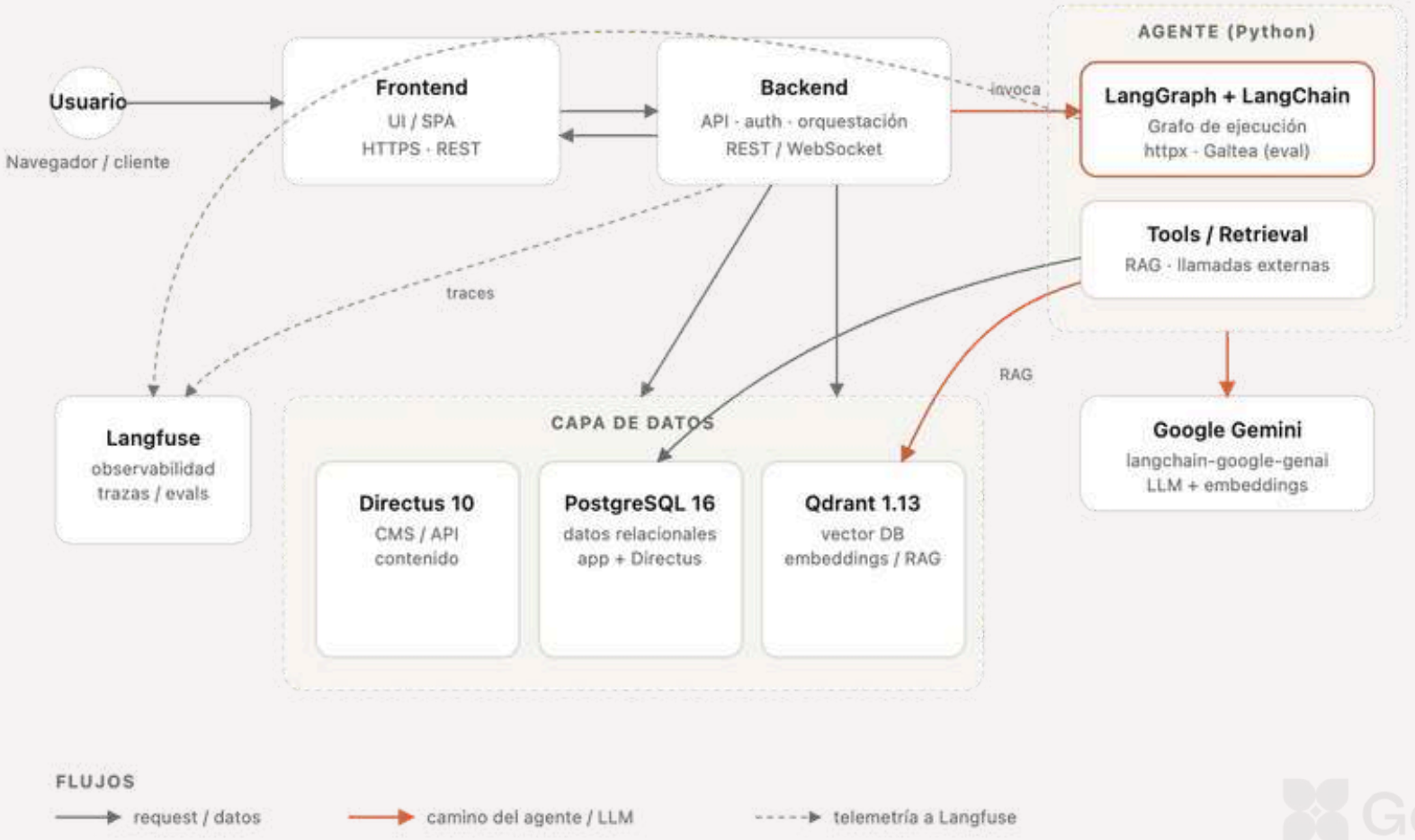
Un tercer impacto relevante aparece en la gestión de citas. El agente no solo recoge los datos necesarios, sino que envía al equipo un resumen estructurado de la conversación previa. Esto transforma la dinámica de las reuniones: el experto ya no parte de cero, sino de un contexto previamente construido.

Finalmente, el sistema incorpora desde el inicio capacidades de observabilidad. Cada interacción se convierte en una fuente de información que permite identificar lagunas en el conocimiento, errores de clasificación o puntos de abandono.

4. Arquitectura de la aplicación

Para entender cómo funciona el agente primero es necesario revisar la arquitectura global de la solución.

La arquitectura se articula sobre un frontend en TypeScript que consulta de forma segura un CMS Directus, mientras el chat interactúa en tiempo real por SSE con un agente independiente en FastAPI. Este núcleo orquesta la lógica mediante un grafo en LangGraph y Gemini, apoyándose en Qdrant para un RAG robusto.



5. Arquitectura del agente

El sistema se construye como un grafo de decisión donde cada nodo tiene una responsabilidad única y bien definida. Esta elección no es estética, sino funcional: permite controlar el flujo de la conversación sin depender de decisiones implícitas del modelo.

El recorrido de un mensaje sigue siempre la misma estructura. Antes de cualquier procesamiento, pasa por un nodo de protección que evalúa si el contenido es seguro. Esta separación es crítica: evita que la clasificación de intención se vea contaminada por inputs adversariales.

Superado este punto, el sistema determina qué tipo de interacción está teniendo lugar. Esta clasificación no solo decide el camino a seguir, sino que también transforma el mensaje en una representación más adecuada para la recuperación semántica, mejorando la calidad del RAG.

A partir de ahí, la conversación se dirige a un nodo especializado. Las consultas de tipo FAQ se resuelven exclusivamente con información recuperada de una base de datos vectorial, sin permitir que el modelo introduzca conocimiento externo. Las búsquedas de contenido devuelven resultados deterministas basados en fragmentos reales del conocimiento de la asociación, evitando interpretaciones del modelo. El flujo de citas, por su parte, sigue una lógica completamente distinta: combina extracción de información, gestión de estado y generación controlada de lenguaje.

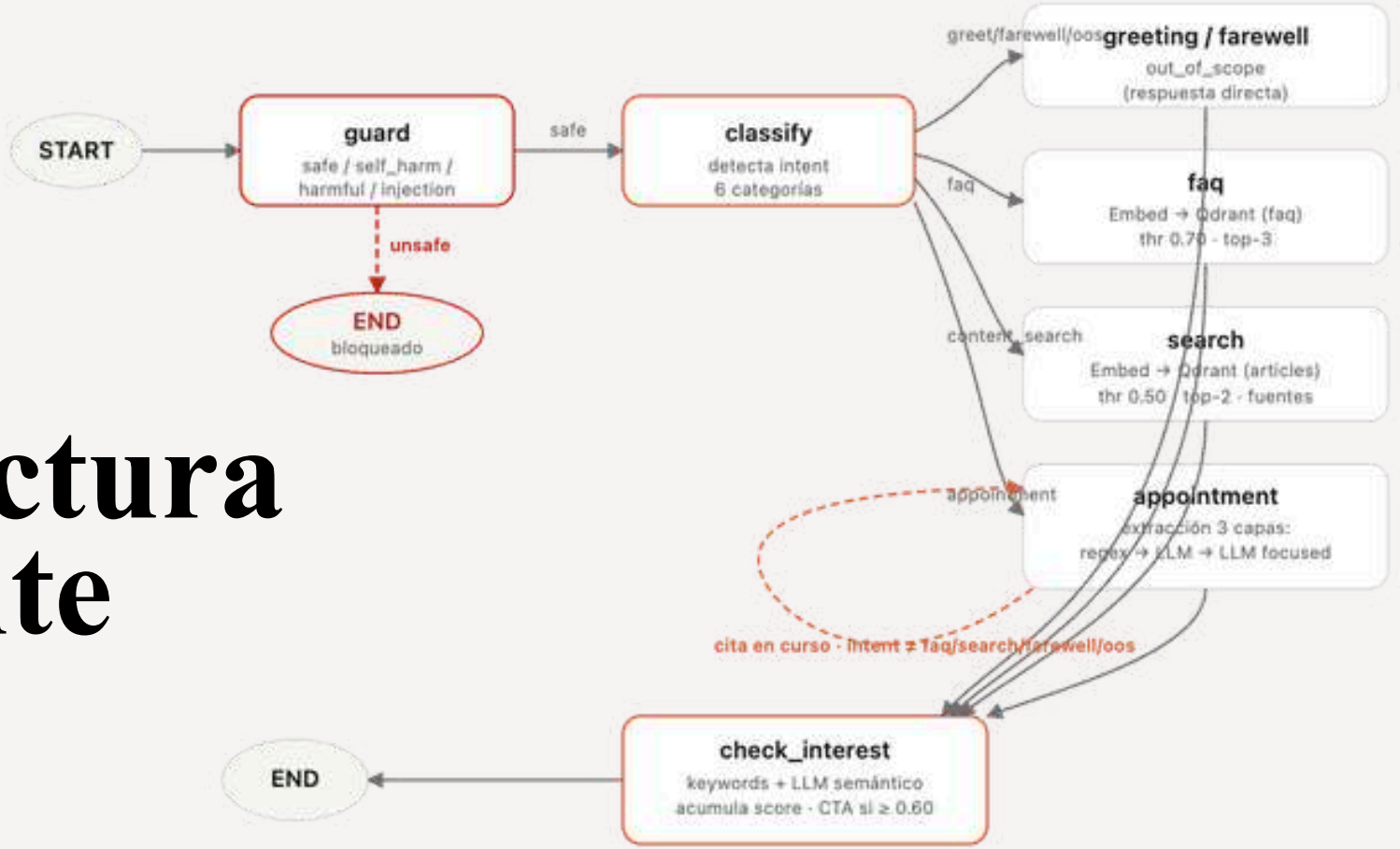
Una vez generada la respuesta, el sistema evalúa si debe intervenir con una llamada a la acción. Esta decisión no es puntual, sino acumulativa: se basa en la evolución de la conversación y se limita a una única vez por sesión.

Uno de los aspectos más relevantes de la arquitectura es su gestión del estado. Aunque la API es stateless, el estado se reconstruye en cada turno a partir del historial reciente. Esto permite escalar horizontalmente sin necesidad de coordinación entre instancias, manteniendo al mismo tiempo coherencia conversacional.

En términos de seguridad, el diseño sigue un enfoque en capas. El filtrado inicial se complementa con prompts estructurados que eliminan la ambigüedad en la salida del modelo, limitaciones en el historial para reducir la superficie de ataque y un acceso estrictamente controlado a los datos. La combinación de estas medidas reduce significativamente el riesgo sin depender de un único mecanismo.

Otro aspecto clave es la reducción deliberada del uso del LLM en aquellos puntos donde no aporta valor. En la búsqueda de artículos, por ejemplo, se opta por respuestas deterministas en lugar de síntesis generativa. Esta decisión mejora la fidelidad al contenido original y reduce el coste por turno de forma significativa.

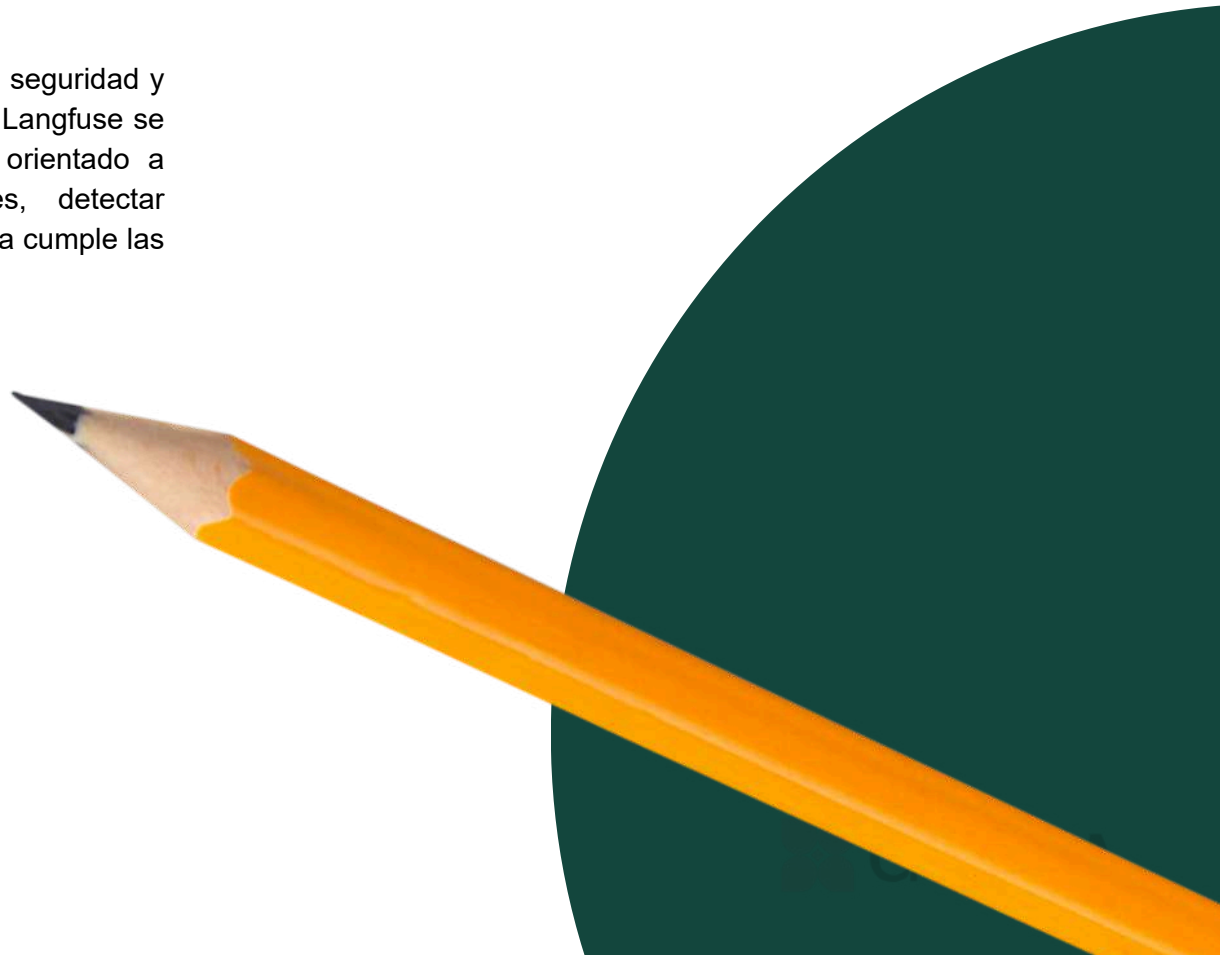
Arquitectura del agente



6. Evaluación del ChatBot

Las pruebas manuales permiten detectar errores evidentes, pero no escalan ni garantizan consistencia en un sistema basado en LLMs. Pequeños cambios en prompts, modelos o contexto pueden alterar el comportamiento del agente de forma impredecible, generando regresiones difíciles de detectar manualmente. Por ello, es necesario complementar la validación humana con evaluaciones automáticas y reproducibles que permitan medir calidad, seguridad y coherencia entre versiones.

Galtea es la plataforma utilizada para evaluar precisión, seguridad y calidad conversacional del agente de GenAiA. Mientras Langfuse se centra en observabilidad y trazabilidad, Galtea está orientado a validación y mejora continua: comparar versiones, detectar regresiones y verificar que el comportamiento del sistema cumple las especificaciones definidas.



6.1 Modelo de evaluación

Galtea organiza la evaluación en tres niveles:

- Producto: define el agente, sus versiones y las especificaciones esperadas.
- Ejecución: registra conversaciones reales mediante sesiones e inference results por turno.
- Evaluación: aplica tests y métricas sobre esos resultados.

Las especificaciones del agente se dividen en:

- capacidades, como responder FAQs o gestionar citas;
- limitaciones, como no inventar información;
- políticas, como bloquear contenido dañino o emitir el CTA una sola vez.

6.2 Tipos de test

El agente utiliza tres categorías principales de evaluación.

Los tests de accuracy validan la calidad factual del sistema RAG: precisión de respuestas, relevancia de recuperación y ausencia de alucinaciones.

Los tests de security evalúan resiliencia frente a prompt injection, jailbreaks y solicitudes dañinas.

Los tests de behavior analizan el comportamiento multi-turno, incluyendo coherencia conversacional, mantenimiento de contexto y resolución de objetivos como completar una cita.

6.3 Métricas

Galtea combina métricas deterministas —similitud textual, validación estructural o comparación de campos— con métricas semánticas basadas en LLM-as-a-judge.

Entre las métricas más relevantes se encuentran:

- factual accuracy y faithfulness para RAG;
- role adherence y conversation completeness para comportamiento conversacional;
- misuse resilience y jailbreak resilience para seguridad.

Además, el agente registra métricas propias:

- interest-signal: estimación del interés del usuario en afiliarse;
- qdrant-search-score: calidad del contexto recuperado desde Qdrant.

7. Problemas encontrados

A continuación, se presenta el desarrollo detallado de los desafíos técnicos identificados y las soluciones arquitectónicas implementadas para el agente de GenAiA.

7.1 Gestión de Ambigüedad en el Flujo de Citas

Durante la fase de pruebas, se detectó un problema de fricción conversacional relacionado con la identificación de nombres propios. En procesos multi-turno como la recogida de datos para citas, el modelo tendía a clasificar entradas como "Francisco" como un simple saludo (greeting) en lugar de una respuesta al flujo de datos.

Impacto: Esta clasificación errónea provocaba la pérdida inmediata del contexto y de los datos ya recogidos del usuario.

Solución Técnica: Se implementó una regla de routing determinista en el código del grafo. Esta regla establece que, si existe una cita en curso, el sistema fuerza la permanencia en el nodo de citas a menos que se detecte una intención de cambio de tema explícita, evitando que el agente pierda el hilo de la gestión sin intervención humana.

7.2 Optimización de la Capa de Seguridad (Guardrails)

Uno de los descubrimientos más críticos fue la degradación de la seguridad al utilizar prompts multifuncionales. Se observó que al mezclar la detección de riesgos (como inyecciones de código o contenido dañino) con la resolución de intenciones, el LLM operaba bajo un sesgo "caritativo".

El Problema: El modelo priorizaba la utilidad de la respuesta frente a entradas ambiguas, relajando involuntariamente los criterios de seguridad y disminuyendo la precisión ante ataques.

Solución Técnica: La arquitectura se rediseñó para separar el nodo de seguridad (guard) del nodo clasificador (classify). Al aislar estas funciones, el nodo de seguridad actúa como un filtro rígido e independiente que evalúa exclusivamente si el input es seguro (safe), previniendo alucinaciones por fuentes externas o exfiltración de datos antes de que la lógica de negocio procese el mensaje.

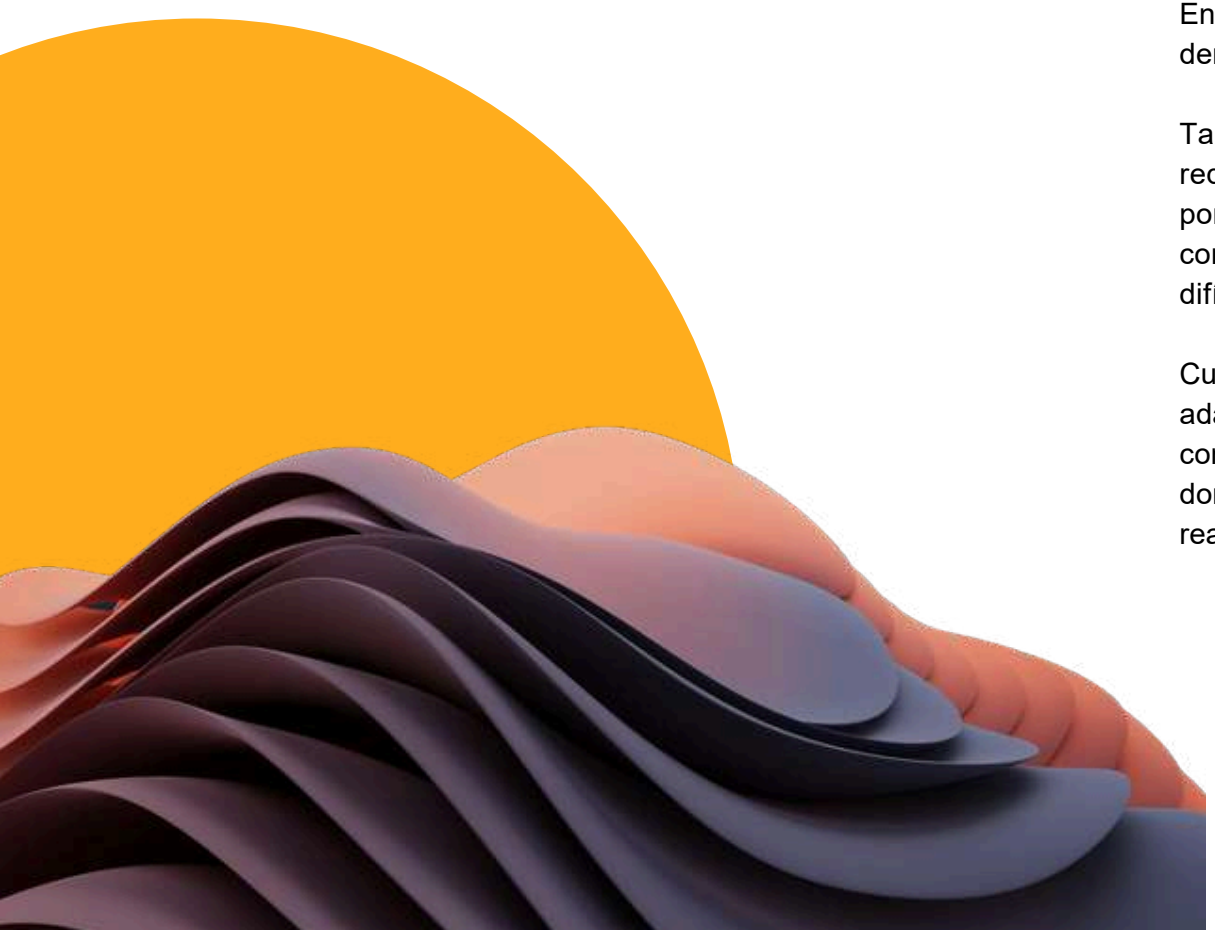
7.3 Precisión en la Clasificación de intenciones

En entornos de producción, el routing basado exclusivamente en prompts demostró ser impredecible ante consultas complejas o poco claras de los usuarios.

El Problema: Se detectaron casos donde la intención se identificaba incorrectamente, derivando al usuario a flujos de FAQ o búsqueda de contenido que no correspondían con su necesidad real.

Solución Técnica: Se adoptó un modelo de estado persistente por nodo a través de LangGraph, donde cada paso de la conversación es rastreado de forma tipada y determinista.

Para mitigar errores residuales, se integró un sistema de observabilidad con Langfuse. Esto permite al equipo técnico analizar sesiones con intents mal clasificados o consultas sin cobertura, facilitando un ciclo de mejora que ajusta los umbrales de búsqueda (como el threshold de 0.70 en FAQ) y mejora la trazabilidad del comportamiento del agente.



8. Ámbito de aplicación

Esta arquitectura está diseñada para contextos donde existe un conocimiento propio que actúa como fuente de verdad y donde la interacción con el usuario no se limita a una única pregunta-respuesta.

Es especialmente adecuada para organizaciones que combinan contenido informativo con procesos de captación o conversión, como asociaciones profesionales, empresas B2B o plataformas formativas. En estos casos, la capacidad de responder, guiar y evaluar intención dentro de la misma conversación aporta un valor diferencial claro.

También resulta relevante en entornos donde la trazabilidad es un requisito. La posibilidad de analizar qué ha ocurrido en cada turno, por qué se ha tomado una decisión y cómo ha evolucionado la conversación permite operar el sistema con un nivel de control difícil de conseguir en arquitecturas más simples.

Cuando se dispone de un corpus estructurado, el tiempo de adaptación es relativamente acotado. La mayor parte del esfuerzo se concentra en la ingestión del conocimiento, el ajuste de prompts al dominio y la calibración inicial del sistema con conversaciones reales.

9. Limitaciones

A pesar de sus ventajas, esta arquitectura no es universal.

La dependencia del conocimiento es absoluta. El sistema solo puede responder con la información que tiene disponible, lo que implica que cualquier lag en la ingestión o deficiencia en la estructura del contenido impacta directamente en la calidad de las respuestas.

El clasificador de intención, aunque robusto, no es infalible. Inputs ambiguos o extremadamente cortos pueden producir errores de enrutamiento, especialmente en las primeras fases de la conversación.

La latencia depende de servicios externos, en particular del proveedor de LLM. Aunque el sistema está optimizado para minimizar llamadas, en escenarios de alta carga o limitaciones de rate puede haber degradación en los tiempos de respuesta.

El guardrail reduce significativamente el riesgo, pero no lo elimina por completo. Como cualquier sistema basado en modelos, puede ser vulnerable a inputs adversariales especialmente elaborados.

Por último, el sistema no mantiene memoria entre sesiones. Esta decisión simplifica la arquitectura y reduce riesgos de privacidad, pero limita la personalización a largo plazo.

10. Conclusión

El desarrollo de este agente demuestra que la diferencia entre un chatbot funcional y un sistema listo para producción no está en el modelo, sino en la arquitectura.

Un enfoque basado solo en prompts puede servir para POCs, pero no para operar con garantías reales. La gestión explícita del estado, la separación de responsabilidades, el control del flujo y la evaluación continua son los elementos que convierten una capacidad técnica en un producto fiable.

La combinación de RAG sobre conocimiento propio, comportamiento determinista y evaluación sistemática no solo mejora la calidad de las respuestas, sino que permite evolucionar el sistema con evidencia y control.

La construcción de sistemas conversacionales robustos requiere talento experto capaz de anticipar fallos reales de producción, diseñar metodologías de validación y traducir comportamientos probabilísticos en flujos controlados y medibles. La diferencia entre un prototipo y un sistema operativo está precisamente en esa capa de ingeniería invisible: decisiones arquitectónicas, control de estados, evaluación sistemática y gestión de excepciones.

El enfoque human-in-the-loop es parte esencial de esa metodología. La supervisión humana no actúa como respaldo puntual, sino como mecanismo continuo de aprendizaje y calibración del sistema: revisión de conversaciones ambiguas, ajuste de evaluaciones, análisis de regresiones y validación de nuevos comportamientos antes de su despliegue.

En este contexto, Galtea aporta una capacidad crítica: convertir la calidad del agente en algo observable y cuantificable. La posibilidad de evaluar versiones, medir comportamiento conversacional, detectar degradaciones silenciosas y correlacionar métricas con señales reales de usuario transforma el desarrollo del agente en un proceso de ingeniería iterativo basado en evidencia, no en percepción subjetiva.

Ese es el verdadero valor de esta arquitectura: no solo construir un agente que funciona, sino un sistema que puede mantenerse, evaluarse y evolucionar de forma fiable a medida que aumenta su complejidad y su impacto en producción.