

UNIVERSIDAD ALFONSO X EL SABIO

Business Tech

Máster Universitario en Inteligencia Artificial



TRABAJO DE FIN DE MÁSTER

**ORQUESTACIÓN DE AGENTES DE IA CON EL USO DE
RAG**

MAXIM UTICA BABYAK

LEONARDO DULCETTI

June 24, 2025

A mi familia, mi madre y mis abuelos. A mi pareja, Fran. A mi compañero, Ibon.
Y a mi tutor, Leonardo.

RESUMEN

Este trabajo presenta el diseño, implementación y evaluación de un sistema inteligente para la planificación de viajes, basado en una arquitectura de orquestación de agentes y enriquecido mediante técnicas de Generación Aumentada por Recuperación (RAG). El objetivo principal ha sido desarrollar una solución capaz de generar itinerarios personalizados a partir de una petición libre del usuario, considerando sus preferencias, restricciones y el conocimiento específico sobre el destino.

La solución propuesta combina un modelo generativo de lenguaje con un agente recuperador que accede a una base documental estructurada sobre las principales ciudades de Australia. La recuperación contextual se realiza mediante técnicas semánticas que maximizan la relevancia de los documentos aportados al generador, mejorando la calidad final de la respuesta.

Se han definido y ejecutado ocho escenarios de prueba, diseñados para evaluar tanto la eficacia en la recuperación de contexto como la calidad de las respuestas generadas. La evaluación cuantitativa incluye métricas como el recall adaptativo, la cobertura temática y la coherencia semántica. Por otro lado, la evaluación cualitativa analiza la coherencia narrativa, la relevancia del contenido y la adaptación al contexto. Los resultados obtenidos demuestran un desempeño sólido del sistema, con una clara correlación entre la calidad del contexto recuperado y la adecuación de los itinerarios generados.

Este trabajo contribuye al desarrollo de sistemas conversacionales inteligentes centrados en dominios específicos, y demuestra la utilidad de enfoques multiagente y RAG en la generación de contenido personalizado con base documental. Además, la plataforma implementada ofrece un entorno interactivo que facilita la evaluación de las respuestas y puede ser ampliado con nuevos módulos o dominios de conocimiento.

Palabras clave

Planificación de viajes - RAG - Orquestación de agentes - Modelos de lenguaje - Evaluación cualitativa y cuantitativa - Sistemas conversacionales - Turismo inteligente

ABSTRACT

This work presents the design, implementation, and evaluation of an intelligent travel planning system based on a multi-agent orchestration architecture, enhanced by Retrieval-Augmented Generation (RAG) techniques. The main objective is to develop a solution capable of generating personalized travel itineraries from open-ended user requests, taking into account user preferences, constraints, and domain-specific knowledge.

The proposed system integrates a language generation model with a retriever agent that accesses a structured document base about major Australian cities. Context retrieval is performed using semantic similarity techniques, maximizing the relevance of documents provided to the generator and enhancing the quality of the final response.

Eight test scenarios were defined and executed to assess both context retrieval effectiveness and the quality of the generated responses. The quantitative evaluation includes metrics such as adaptive recall, thematic coverage, and semantic coherence. In parallel, a qualitative evaluation was conducted, analyzing narrative coherence, content relevance, and context adaptation. The results show strong system performance and a clear correlation between the quality of the retrieved context and the appropriateness of the generated itineraries.

This work contributes to the development of intelligent conversational systems for specialized domains and demonstrates the usefulness of multi-agent orchestration and RAG approaches in generating personalized, knowledge-grounded content. Additionally, the implemented platform provides an interactive environment that facilitates response evaluation and can be extended with new modules or knowledge domains.

Keywords

Travel planning - Retrieval-Augmented Generation (RAG) - Agent orchestration - Language models - Qualitative and quantitative evaluation - Conversational systems - Intelligent tourism

Índice general

1. Introducción al Trabajo Fin de Máster	6
1.1. Justificación del proyecto realizado	6
1.2. Presentación del proyecto y sus contenidos	7
2. Objetivos del TFM	11
2.1. Objetivo general	11
2.2. Objetivos específicos	11
3. Marco Teórico	13
3.1. Contexto	14
3.1.1. IA Generativa	14
3.1.2. Modelos de Lenguaje Grande	15
3.1.3. Orquestación de agentes de IA	17
3.1.4. Generación Aumentada por Recuperación	19
3.2. Estado del Arte	20
3.2.1. Desafíos en LLMs: alucinaciones y fiabilidad	21
3.2.2. Fundamentos para agentes autónomos basados en LLMs	21
3.2.3. Hacia arquitecturas multiagente con RAG	22
4. Marco Metodológico	25
4.1. Construcción de los agentes	26
4.1.1. ControllerAgent: coordinación del flujo RAG	26
4.1.2. RetrieverAgent: recuperación semántica contextualizada	27
4.1.3. LLMAgent: generación de respuesta con LLM	29
4.2. Definición del grafo de ejecución	30
4.2.1. Modelo de estado compartido: AgentState	30
4.2.2. Construcción del grafo: build_langgraph_controller_flow	31
4.3. Diseño de la interfaz web	32
4.3.1. Estructura del formulario de entrada	32
4.3.2. Visualización de tokens y validación	33

4.3.3.	Envío del prompt y presentación del resultado	33
4.3.4.	Gestión de configuración desde archivos YAML	33
4.3.5.	Ventajas del enfoque con Streamlit	33
4.3.6.	Referencias cruzadas con el código fuente	34
5.	Presentación de análisis, resultados y discusión	36
5.1.	Metodología de evaluación	37
5.1.1.	Evaluación de la recuperación de contexto	37
5.1.2.	Evaluación de la calidad de las respuestas generadas	38
5.1.3.	Diseño de los escenarios de prueba	38
5.2.	Eficacia en la recuperación de contexto	40
5.2.1.	Métricas de evaluación	40
5.2.2.	Resultados	40
5.3.	Evaluación de la calidad de las respuestas generadas	43
5.3.1.	Métricas de evaluación	43
5.3.2.	Resultados	43
5.4.	Discusión	50
5.4.1.	Capacidad del sistema para recuperar contexto útil	51
5.4.2.	Generación de itinerarios: solidez narrativa y adecuación temática	51
5.4.3.	Impacto del contexto recuperado en la personalización	51
5.4.4.	Limitaciones detectadas y oportunidades de mejora	52
6.	Conclusiones y líneas futuras	53
6.1.	Conclusiones	53
6.2.	Líneas futuras	54
A.	Itinerarios generados	55
B.	Código fuente	64
	Siglas	107
	Referencias	108

Introducción al Trabajo Fin de Máster

Para la introducción de este trabajo se derivará en dos apartados. El primer apartado tratará la justificación del proyecto realizado, respondiendo sobre su origen y la utilidad que aportaría. El segundo apartado describirá de forma resumida la contribución del proyecto, recorriendo la distribución de los contenidos del proyecto.

1.1 › Justificación del proyecto realizado

Este proyecto surge del creciente interés en la coordinación eficiente de agentes de Inteligencia Artificial (IA) en entornos complejos, abiertos y dinámicos. La evolución reciente de los modelos de lenguaje natural, como el del Modelo de Lenguaje Grande (LLM), ha permitido que sistemas conversacionales sean cada vez más competentes a la hora de generar respuestas fluidas, útiles y en lenguaje natural (Brown et al., 2020). Sin embargo, estas capacidades suelen estar limitadas por su conocimiento estático, lo que lleva a problemas como las alucinaciones o la desactualización de la información generada.

En este contexto, surge la oportunidad de explorar arquitecturas más robustas que combinen la capacidad de generación de los modelos con técnicas de recuperación de información en tiempo real. De este modo, la integración de enfoques RAG (Lewis et al., 2020) con agentes inteligentes abre nuevas posibilidades para construir sistemas que puedan razonar y responder de manera más informada, contextual y específica.

La elección de una arquitectura multiagente no es trivial. Este enfoque promueve la modularidad, la escalabilidad y la asignación especializada de tareas. Cada agente

actúa de forma autónoma sobre un subproblema concreto (recuperación, control, generación...), lo que facilita el mantenimiento, la trazabilidad de errores y la futura extensión del sistema a nuevos dominios de conocimiento. Además, permite emular una forma de razonamiento distribuido, donde las decisiones se basan en la cooperación entre unidades funcionales especializadas.

Este proyecto no solo resulta atractivo desde el punto de vista técnico, sino también desde el punto de vista científico, educativo y social. A nivel científico, aborda uno de los retos más relevantes en el campo de la inteligencia artificial generativa: cómo garantizar que los modelos accedan a información fiable, reciente y relevante. El uso de recuperación semántica desde un almacén vectorial mitiga las limitaciones de memoria estática del modelo, ayudando a reducir errores y respuestas poco fundamentadas.

Desde una perspectiva educativa, los sistemas de recomendación asistidos por IA, capaces de generar contenido personalizado con respaldo documental, representan una herramienta poderosa para el aprendizaje guiado. Por ejemplo, podrían integrarse en entornos de enseñanza para sugerir lecturas, generar planes de estudio o apoyar en tareas de investigación, todo ello fundamentado en bases de conocimiento estructuradas.

En el ámbito social y aplicado, los sistemas multiagente con RAG pueden adaptarse fácilmente a dominios críticos como la medicina, la asistencia legal, la planificación de rutas logísticas o la generación automatizada de contenidos. La capacidad de adaptar la respuesta a las preferencias del usuario, como ocurre en este caso de uso centrado en planificación de viajes, permite ofrecer experiencias altamente personalizadas que mejoran la interacción y reducen el esfuerzo cognitivo del usuario final.

Finalmente, este proyecto también representa una oportunidad formativa y profesional para consolidar conocimientos avanzados en inteligencia artificial. La experiencia adquirida durante su desarrollo abarca técnicas de orquestación de agentes, uso de modelos generativos, recuperación semántica, representación del estado y desarrollo de interfaces funcionales con *Streamlit* (S. Inc., 2024), lo que supone un entorno de desarrollo realista, alineado con las necesidades actuales del sector tecnológico.

1.2 › Presentación del proyecto y sus contenidos

Este proyecto propone una orquestación de agentes de inteligencia artificial para automatizar la planificación de viajes en Australia, integrando dos casos de uso clave:

- **Consulta de información en tiempo real:** Un agente de IA buscará, filtrará y presentará información relevante sobre vuelos, alojamientos y opciones de transporte, utilizando técnicas de RAG para mejorar la precisión de las respuestas.
- **Generación de respuestas contextualizadas:** Un agente basado en un LLM se encargará de sintetizar la información recuperada y generar itinerarios personalizados, coherentes y adaptados a las preferencias del usuario, teniendo

en cuenta factores como la duración, el presupuesto, el tipo de viaje y los intereses específicos.

Al coordinar estos agentes en una arquitectura integrada, el sistema ofrecerá una solución más precisa, automatizada y eficiente, reduciendo el tiempo de planificación y mejorando la experiencia del usuario.

Desde el punto de vista técnico, la solución ha sido implementada en *Python* utilizando una arquitectura modular basada en agentes orquestados mediante la librería *LangGraph*. Cada agente tiene una responsabilidad específica dentro del flujo de trabajo conversacional y está diseñado para interactuar mediante un esquema de estado compartido. La implementación sigue el paradigma de RAG, permitiendo recuperar información relevante desde un almacén vectorial antes de proceder a la generación de respuestas.

El flujo está compuesto por los siguientes agentes:

- **ControllerAgent:** coordina el flujo global del sistema. Decide si el sistema debe recuperar contexto o generar directamente la respuesta en función del estado actual.
- **RetrieverAgent:** accede a un índice de documentos configurado en *Azure Cognitive Search* (Azure, 2023a), utilizando *embeddings* generados mediante un modelo de *Azure OpenAI* compatible con el índice vectorial. Recupera las secciones más relevantes en base a la consulta del usuario y sus intereses explícitos.
- **LLMAgent:** genera la respuesta final utilizando un modelo de lenguaje de *Azure OpenAI* (como *GPT-3.5-Turbo*), integrando tanto la entrada del usuario como el contexto recuperado.

El estado del sistema se representa mediante una estructura tipada llamada **AgentState**, que contiene las claves `input`, `response` y `last_node`. Este estado fluye entre los nodos del grafo durante la ejecución.

El flujo general sigue una lógica condicional que permite iterar entre recuperación y control hasta llegar a la generación final de la respuesta, definida como:

`controlador → (consulta → controlador)* → llm → END`

Para guiar el comportamiento de los agentes de inteligencia artificial, se han definido una serie de prompts del sistema cuidadosamente diseñados. Estos prompts actúan como instrucciones explícitas que delimitan las capacidades, el tono y la función de cada agente en el flujo de orquestación.

La formulación de estos prompts influye directamente en la calidad de las respuestas generadas, ya que condiciona el contexto de inferencia del modelo LLM. Algunos prompts están orientados a la recuperación de información (por ejemplo, reforzando la búsqueda semántica), mientras que otros están diseñados para la generación de texto instructivo o conversacional.

Un resumen completo de los prompts utilizados, junto con su función específica en el sistema, se recoge en el Anexo 3.

El `default_system_prompt` indica que el modelo debe comportarse como un planificador experto en viajes por Australia, e impone límites estrictos como:

- No hablar de países, culturas ni destinos fuera de Australia.
- Responder siempre en español.
- Adaptar los itinerarios al presupuesto, intereses, tipo de viaje y duración especificados.
- Respetar criterios de eficiencia geográfica y realismo logístico (por ejemplo, evitar traslados largos en días consecutivos).
- Recomendar el medio de transporte más adecuado y tener en cuenta horarios de atracciones y restaurantes.
- Mantener un tono amable, claro y directo.

Además, en caso de que la recuperación de contexto falle o no proporcione contenido útil, se activa un `fallback_prompt` que instruye al modelo para responder con conocimiento general, pero manteniendo el idioma y tono apropiado.

Por otro lado, el `prompt_base` funciona como plantilla dinámica que se adapta a cada usuario, formateando una consulta a partir de las variables seleccionadas en la interfaz (`days`, `budget`, `travel_type`, `interests`), lo que permite construir prompts estructurados, coherentes y fácilmente trazables.

Este enfoque basado en ingeniería de prompts no solo ayuda a limitar el rango de generación del modelo, sino que también mejora la consistencia, evita alucinaciones y garantiza que las respuestas estén alineadas con el propósito del sistema.

Para la interacción con el usuario final, se ha desarrollado una interfaz web con *Streamlit*, que permite introducir detalles del viaje (duración, presupuesto, tipo de viaje e intereses), mostrar en tiempo real el uso de tokens del mensaje y generar un itinerario personalizado al pulsar un botón.

Finalmente, la ejecución del flujo se centraliza en el módulo `runner.py`, donde una función de alto nivel llamada `run_prompt` inicializa el estado con los datos del usuario y desencadena la ejecución completa del grafo, devolviendo la respuesta generada por el modelo.

Gracias a esta arquitectura, el sistema no solo mejora la precisión y relevancia de las respuestas, sino que también proporciona una base extensible para futuras mejoras y nuevos dominios de aplicación.

La estructura del trabajo se organiza de la siguiente manera: en la Sección 2 se presentan los objetivos del estudio, comenzando con el objetivo general y, posteriormente, los objetivos específicos. A continuación, en la Sección 3, se expone el marco teórico, abordando el concepto de Inteligencia Artificial Generativa (IAG), la orquestación de agentes de IA, el uso de RAG y la tecnología *Azure AI Search* (Microsoft Corporation, 2024), además de un análisis del estado del arte. La Sección 4 detalla la metodología empleada en el desarrollo del trabajo, incluyendo las decisiones de diseño e implementación de la propuesta. Posteriormente, en la Sección 5, se presentan los resultados obtenidos y se evalúa el desempeño del sistema

desarrollado. Finalmente, en la Sección 6, se sintetizan las principales conclusiones del estudio y se plantean posibles líneas futuras de investigación y desarrollo.

Objetivos del TFM

En este capítulo se establecen las bases del estudio, definiendo el objetivo general del trabajo y los objetivos específicos que guiarán su desarrollo. Se describe la meta principal del proyecto y se desglosa en objetivos concretos, alineados con las distintas secciones del estudio: el marco teórico, la metodología, la implementación del sistema, la evaluación técnica y el análisis de resultados. Estos objetivos permiten estructurar el trabajo de manera ordenada y garantizan que el proyecto cumpla con su propósito: desarrollar un sistema funcional de orquestación de agentes de IA para la planificación de viajes, basado en principios de RAG y modelos de lenguaje generativo.

2.1 › Objetivo general

- Diseñar y desarrollar un sistema de orquestación de agentes de inteligencia artificial basado en técnicas de RAG y modelos de lenguaje, capaz de automatizar la planificación de viajes personalizados en Australia mediante la integración de consultas en tiempo real y generación de itinerarios adaptados a las preferencias del usuario.

2.2 › Objetivos específicos

- Estudiar el estado del arte en orquestación de agentes de IA, técnicas de recuperación aumentada (RAG) y aplicaciones de modelos generativos en tareas de planificación, contextualizando el proyecto en el marco actual de la inteligencia artificial aplicada.

- Definir la arquitectura del sistema propuesto, seleccionando las herramientas tecnológicas y servicios en la nube más adecuados (como *Azure AI Search* y LLMs) para una implementación eficiente y escalable.
- Implementar un conjunto de agentes inteligentes con responsabilidades diferenciadas (recuperación, razonamiento, generación) que colaboren entre sí mediante una orquestación modular y extensible.
- Integrar fuentes de datos externas en tiempo real (como información sobre transporte, eventos y destinos) utilizando mecanismos de actualización dinámica y filtrado semántico.
- Evaluar el desempeño del sistema mediante un enfoque mixto: se emplean métricas cuantitativas como la precisión en la recuperación (*Recall@k*), la coherencia semántica entre la respuesta y una referencia, y la cobertura temática alineada con los intereses del usuario; y métricas cualitativas centradas en la calidad de los itinerarios generados, valorando la coherencia narrativa, la relevancia del contenido y su adaptación al contexto proporcionado.
- Identificar las limitaciones del enfoque propuesto y proponer líneas futuras de investigación orientadas a mejorar la coordinación entre agentes, la incorporación de preferencias más complejas o la adaptación a otros dominios.

Marco Teórico

El presente marco teórico proporciona los fundamentos conceptuales y tecnológicos sobre los que se sustenta este trabajo, estructurándose en dos partes complementarias: el contexto y el estado del arte.

En primer lugar, se introduce el concepto de IAG, una rama de la IA centrada en la producción autónoma de contenido, impulsada por los avances recientes en modelos de lenguaje de gran escala (LLMs). Se analizan las capacidades de estos modelos, así como sus limitaciones en entornos dinámicos, como la tendencia a generar información incorrecta o desactualizada. Ante estos desafíos, se presentan los enfoques de orquestación multiagente y recuperación aumentada (RAG) como soluciones que permiten distribuir responsabilidades entre agentes especializados y mejorar la fiabilidad de las respuestas generadas mediante el acceso a fuentes externas de información.

A continuación, se examina el estado del arte relacionado con la intersección entre LLMs, RAG y arquitecturas multiagente. La integración de estos elementos representa una evolución significativa en el diseño de sistemas inteligentes, al permitir la construcción de agentes capaces de colaborar, razonar de forma distribuida y adaptarse a contextos complejos. Esta revisión se organiza en torno a tres grandes bloques: los desafíos y fundamentos de los LLMs, los principios para el diseño de agentes autónomos basados en estos modelos, y las soluciones aplicadas en arquitecturas multiagente con capacidades de recuperación. Este análisis proporciona el marco necesario para contextualizar las decisiones técnicas y arquitectónicas del sistema desarrollado.

3.1 › Contexto

La presente sección proporciona el contexto teórico necesario para comprender los fundamentos sobre los que se apoya este trabajo. En primer lugar, se introduce el concepto de IAG, una rama de la IA orientada a la producción autónoma de contenido, que ha cobrado especial relevancia con el desarrollo de modelos LLM. A continuación, se analizan las características y capacidades de estos modelos, así como los retos asociados a su uso en entornos dinámicos y de propósito específico. Posteriormente, se explora el paradigma de la orquestación de agentes de IA, que permite distribuir responsabilidades entre componentes inteligentes especializados, fomentando la colaboración y modularidad del sistema. Finalmente, se aborda el enfoque de RAG, una técnica que combina modelos generativos con mecanismos de recuperación de información para mejorar la relevancia y precisión de las respuestas generadas, especialmente en dominios donde el conocimiento actualizado o específico es crítico. Este marco conceptual establece las bases para el diseño e implementación del sistema propuesto.

3.1.1 › IA Generativa

La inteligencia artificial generativa es una tecnología que permite a los sistemas informáticos crear contenido nuevo y original (Bommasani et al., 2021), como si fuera generado por una persona. Este contenido puede ser de muchos tipos: texto, imágenes, audio, vídeo, e incluso modelos en 3D o código de programación. A diferencia de otros sistemas de IA que simplemente analizan o clasifican información existente, la IA generativa aprende de grandes cantidades de datos para luego producir resultados nuevos que no estaban en esos datos de forma literal.

Este tipo de IA se basa principalmente en modelos de aprendizaje automático, como las redes neuronales profundas, que imitan el funcionamiento del cerebro humano al reconocer patrones complejos en la información.

- **Modelos generativos de texto.** Estos modelos son capaces de escribir textos en lenguaje natural, responder preguntas, traducir idiomas o generar descripciones automáticas. Un ejemplo muy conocido es *ChatGPT*, que genera respuestas coherentes y personalizadas a partir de preguntas o instrucciones (OpenAI, 2022). También se utilizan para tareas como redacción automática de informes, generación de resúmenes y asistencia en la escritura creativa. Además, algunos modelos de IAG están entrenados para escribir código de programación, generar funciones completas o ayudar en tareas de desarrollo de software.
- **Modelos generativos de imágenes.** Utilizando modelos como las Red Generativa Adversativa (GAN) (Goodfellow et al., 2014) o los Modelo de Difusión (DM) (Rombach et al., 2022), la IA puede generar imágenes completamente nuevas. Esto se aplica en áreas como el arte digital, el diseño de productos, la moda, o la creación de paisajes y personajes en videojuegos.
- **Modelos de generación de audio y música.** Algunos sistemas están entrenados para componer música o generar voces sintéticas realistas. Estos modelos

pueden, por ejemplo, crear una canción original a partir de un estilo musical o leer textos en voz alta imitando voces humanas. Son muy utilizados en videojuegos, cine o aplicaciones educativas.

- **Modelos generativos de vídeo.** Aunque más recientes y complejos, existen modelos que pueden crear vídeos cortos animados o editar vídeos automáticamente a partir de texto. Se usan para generar contenido publicitario, clips animados o visualizaciones científicas.
- **Modelos multimodales.** Estos modelos combinan distintos tipos de contenido (texto, imagen, audio, etc.) y pueden interpretar y generar información en múltiples formatos. Por ejemplo, pueden describir una imagen en palabras o generar una imagen a partir de un texto. Este tipo de IA es muy potente y tiene aplicaciones muy diversas, desde la accesibilidad hasta la generación de contenido educativo.

Aunque el potencial de la IA generativa es extraordinario, también existen retos importantes asociados a su uso. Uno de los principales desafíos es el fenómeno conocido como *alucinaciones*, en el que el modelo genera contenido incorrecto o inventado que aparenta ser veraz (Ji, Lee, Fries et al., 2023b; Maynez et al., 2020). Además, estos sistemas pueden reproducir sesgos presentes en los datos de entrenamiento, generando respuestas discriminatorias, poco éticas o culturalmente insensibles si no se aplican mecanismos de control adecuados. También hay preocupaciones sobre el uso malintencionado de estas tecnologías, por ejemplo, en la generación de desinformación o suplantaciones digitales. Por otro lado, el costo computacional de entrenar y ejecutar modelos generativos a gran escala sigue siendo elevado (Strubell et al., 2019), lo que limita su accesibilidad.

En conjunto, la IA generativa amplía las capacidades creativas de las máquinas, y ya está teniendo un impacto profundo en sectores como la educación, el diseño, la salud, la comunicación y la industria del entretenimiento. En el contexto de este trabajo, su integración con sistemas de recuperación de información (como RAG) y agentes inteligentes permite automatizar tareas complejas como la planificación de viajes, combinando creatividad, precisión y acceso a datos actualizados. Estos modelos se adaptan particularmente bien a tareas como la generación de itinerarios personalizados, respuestas contextualizadas y recomendaciones prácticas a partir de la información disponible en tiempo real.

3.1.2 › Modelos de Lenguaje Grande

Un LLM es una categoría específica dentro de la inteligencia artificial generativa, enfocada en comprender y generar lenguaje natural. Estos modelos están diseñados para producir texto de forma coherente, contextualizada y fluida, imitando el estilo y estructura del lenguaje humano. Para ello, se entrenan con grandes volúmenes de texto, que incluyen libros, artículos, páginas web, conversaciones y otros documentos, lo que les permite aprender patrones lingüísticos, semánticos y sintácticos (Brown et al., 2020).

Un modelo de lenguaje se considera “grande” por varias razones:

- **Cantidad de parámetros:** Los LLMs contienen millones o incluso billones de

parámetros ajustables durante el entrenamiento. Estos parámetros codifican el conocimiento del modelo y permiten representar relaciones complejas entre palabras, conceptos y estructuras de pensamiento (Bommasani et al., 2021).

- **Tamaño del corpus de entrenamiento:** El entrenamiento se realiza con conjuntos masivos de datos textuales, extraídos de fuentes diversas. Esta variedad permite a los modelos desarrollar un conocimiento generalizado y versátil sobre numerosos temas.
- **Capacidad multitarea:** A diferencia de modelos entrenados para una única tarea, los LLMs pueden realizar múltiples funciones, como traducir idiomas, resumir documentos, generar texto creativo, responder preguntas o escribir código, sin requerir entrenamiento adicional para cada una.

Desde el punto de vista arquitectónico, la mayoría de los LLMs actuales se basan en la arquitectura *Transformer*, introducida por Vaswani et al. Esta arquitectura permite procesar secuencias completas de texto en paralelo, mejorando la eficiencia y la capacidad de capturar dependencias de largo alcance. El mecanismo central del *Transformer* es la **atención**, que permite al modelo identificar qué partes del texto son relevantes para generar la siguiente palabra. Este enfoque ha reemplazado a arquitecturas anteriores como el de la Red neuronal recurrente (RNN) o la Red de memoria a corto-largo plazo (LSTM), y ha sido adoptado por modelos como *GPT*, *BERT*, *T5* o *LLaMA*.

Otra distinción importante es la diferencia entre modelos propietarios y modelos de código abierto:

- **Modelos cerrados**, como *GPT-4* (OpenAI, 2023c), *Claude* (Anthropic, 2023) o *Gemini* (DeepMind, 2023), se entrenan con grandes recursos computacionales y suelen ofrecerse como servicios comerciales en plataformas en la nube. Estos modelos son de alto rendimiento, pero sus parámetros y datos de entrenamiento no son públicos.
- **Modelos de código abierto**, como *LLaMA* (Touvron et al., 2023), *Mistral* (Jiang & AI, 2023) o *Falcon* (Penedo et al., 2023), son publicados por comunidades académicas o empresas que promueven la transparencia. Aunque pueden tener un tamaño menor, permiten ser adaptados e integrados directamente en aplicaciones propias, como es el caso de este proyecto.

Los LLMs han transformado múltiples sectores:

- En **educación**, sirven como tutores virtuales, asistentes de redacción o herramientas de autoevaluación.
- En **atención al cliente**, se usan para crear chatbots capaces de mantener conversaciones naturales y resolver problemas complejos.
- En **comunicación y medios**, automatizan la redacción de noticias, informes y contenido publicitario.
- En **programación**, actúan como asistentes inteligentes que ayudan a escribir y depurar código.
- En **investigación científica**, ayudan a sintetizar literatura técnica y explorar nuevas hipótesis.

Sin embargo, su uso también conlleva ciertos riesgos. Uno de ellos es la generación de respuestas plausibles pero incorrectas (**alucinaciones**) (Ji, Lee, Fries et al., 2023b), que pueden inducir a error si no se validan con fuentes fiables. Otro riesgo es la perpetuación de sesgos presentes en los datos de entrenamiento, lo que puede afectar la equidad y neutralidad de las respuestas. Por ello, en sistemas críticos, se recomienda combinar los LLMs con mecanismos de control o verificación externa, como la recuperación de información basada en contexto (Lewis et al., 2020).

En este trabajo, los LLMs desempeñan un papel central: son el componente generativo que transforma las consultas del usuario, junto con el contexto recuperado por los agentes, en respuestas completas y personalizadas. A través de técnicas como RAG, estos modelos se integran con bases de conocimiento externas para proporcionar respuestas más precisas, actuales y adaptadas a cada caso de uso. Su capacidad para adaptarse dinámicamente al contexto es clave para automatizar de forma inteligente la planificación de viajes y ofrecer una experiencia más útil y eficiente al usuario.

3.1.3 › Orquestación de agentes de IA

Los agentes de inteligencia artificial son componentes autónomos que perciben su entorno, procesan información y ejecutan acciones dirigidas a alcanzar un objetivo. Cada agente puede estar especializado en una tarea concreta, como recuperar datos, analizar texto o generar lenguaje natural. En lugar de depender de un único modelo monolítico, la orquestación de agentes permite diseñar sistemas compuestos por múltiples agentes que trabajan de forma colaborativa y estructurada (Wooldridge, 2009).

La orquestación de agentes de IA consiste en coordinar estratégicamente un conjunto de agentes especializados para que colaboren en la resolución de un problema complejo. Esta coordinación no solo implica que los agentes interactúen entre sí, sino que lo hagan según un flujo lógico y organizado que optimice la eficiencia y modularidad del sistema (Russell & Norvig, 2021). Este enfoque favorece el principio de responsabilidad única: cada agente ejecuta una función específica y delega el resto a sus compañeros.

Tipos de agentes

La clasificación de agentes suele hacerse según su complejidad y nivel de autonomía:

- **Agentes reactivos:** Responden a estímulos inmediatos sin memoria ni planificación.
- **Agentes con estado interno:** Tienen una memoria interna que les permite tomar decisiones en función del contexto.
- **Agentes basados en objetivos:** Planifican acciones para alcanzar una meta.
- **Agentes de aprendizaje:** Modifican su comportamiento en función de la experiencia o los datos recibidos.

Estrategias de coordinación

La orquestación puede organizarse siguiendo distintos patrones:

- **Secuencial:** Los agentes se activan en un orden fijo, como etapas en una cadena de procesamiento.
- **Paralela:** Múltiples agentes operan simultáneamente y combinan sus resultados.
- **Jerárquica:** Un agente principal (o coordinador) dirige al resto, decide el orden de ejecución y centraliza el control del flujo.

Esta última forma, conocida como arquitectura controlador-subordinado o *hub-and-spoke*, es particularmente útil cuando se requiere trazabilidad, control de errores o decisiones condicionales (Docs, 2022).

Casos de uso

La orquestación de agentes ya se aplica en diversas áreas:

- **Asistentes conversacionales complejos**, donde un agente principal delega tareas a otros especializados en recuperar información, interpretar emociones o consultar bases de datos.
- **Sistemas autónomos de diagnóstico médico**, que combinan agentes de análisis de imágenes, procesamiento de texto clínico y generación de informes.
- **Automatización de procesos empresariales**, donde distintos agentes interactúan con distintos tipos de Interfaz de Programación de Aplicaciones (API), extraen datos, procesan documentos y notifican a usuarios humanos.

Aplicación en este proyecto

En el caso de KoalaRoute, la orquestación permite descomponer la tarea compleja de planificar un viaje en subprocesos manejados por agentes especializados:

- El **agente de recuperación** se encarga de buscar información relevante sobre actividades, alojamiento y transporte en una base documental.
- El **agente generativo**, basado en un LLM, sintetiza y presenta la información al usuario.
- Un **agente controlador** gestiona la transición entre fases, supervisando el flujo general del sistema según el estado de la interacción.

Esta arquitectura modular aporta ventajas clave: permite escalar el sistema fácilmente con nuevos agentes (por ejemplo, uno de clima o recomendaciones gastronómicas), facilita el mantenimiento del código y mejora la robustez frente a errores. Además, al estar cada agente desacoplado, pueden entrenarse, modificarse o desplegarse de forma independiente, aumentando la flexibilidad general del sistema.

3.1.4 › Generación Aumentada por Recuperación

Uno de los principales retos de los modelos de lenguaje grande (LLMs) es su dependencia del conocimiento estático adquirido durante el entrenamiento. Aunque son capaces de generar texto con gran fluidez y precisión, sus respuestas se basan únicamente en la información contenida en sus pesos, lo que puede provocar errores, desactualización o falta de contexto específico en dominios concretos (Ji, Lee, Fries et al., 2023a). Frente a este problema, surge la técnica de RAG (Generación Aumentada por Recuperación), una solución híbrida que combina modelos generativos con mecanismos de recuperación de información externos (Lewis et al., 2020).

La esencia del enfoque RAG reside en proporcionar al LLM acceso a una fuente de conocimiento adicional y dinámica, como una base documental o un motor de búsqueda. Así, cuando el usuario realiza una consulta, el sistema primero recupera fragmentos de información relevantes (pasajes, documentos, párrafos, etc.), y luego utiliza esos fragmentos como contexto para que el modelo genere una respuesta más precisa y fundamentada (Karpukhin et al., 2020). Esto elimina la necesidad de reentrenar el modelo cada vez que se requiere conocimiento actualizado o especializado (Izacard & Grave, 2020).

Fases del proceso RAG

El pipeline clásico de RAG consta de dos etapas principales:

- **Recuperación de información (Retrieval):** El sistema utiliza técnicas de búsqueda para localizar documentos relevantes en función de la entrada del usuario. Esta búsqueda puede realizarse mediante métodos tradicionales (*sparse retrieval*, como *BM25*) o mediante representaciones vectoriales (*dense retrieval*) basadas en *embeddings* semánticos (Lin et al., 2021).
- **Generación (Generation):** El LLM toma la información recuperada y genera una respuesta final, incorporando los fragmentos relevantes como contexto de entrada.

Además, existen variantes avanzadas de RAG:

- **RAG Pipeline vs RAG End-to-End:** En el modelo por *pipeline*, la recuperación y la generación se implementan como módulos independientes; en el enfoque *end-to-end*, ambos componentes se integran en un único proceso entrenable (Asai et al., 2023).
- **Dense vs Sparse Retrieval:** Los métodos *sparse* se basan en coincidencias literales de palabras clave, mientras que los *dense* codifican textos en espacios vectoriales, permitiendo búsquedas semánticas más robustas (Karpukhin et al., 2020; Xiong et al., 2021).
- **Fusion-in-Decoder (FiD):** Una técnica en la que múltiples documentos recuperados se inyectan directamente como entradas paralelas al decodificador del LLM, permitiendo razonar sobre varias fuentes simultáneamente (Izacard & Grave, 2020).

Formato de entrada estructurado

Para maximizar la efectividad del proceso generativo, es habitual utilizar *prompts estructurados*, que organizan de forma explícita el contenido que recibe el modelo. En este trabajo, se emplea un formato que distingue claramente entre el rol del sistema, el contexto recuperado y la pregunta del usuario, facilitando una inferencia más precisa. Una descripción detallada de estos formatos puede consultarse en el Anexo 3.

Implementación en el sistema KoalaRoute

En el contexto de este TFM, RAG constituye el núcleo funcional del sistema de planificación de viajes. La implementación se basa en los siguientes componentes:

- **Vector store:** Se utiliza **Azure AI Search** como motor de recuperación semántica, indexando documentos en formato `.md` relacionados con ciudades australianas. Cada documento contiene secciones etiquetadas (clima, transporte, alojamiento, etc.), lo que facilita la búsqueda contextual.
- **Modelo de *embeddings*:** Para convertir texto en representaciones vectoriales se utiliza un modelo de *embeddings* desplegado en *Azure OpenAI* (OpenAI, 2023b), configurado mediante la clase `AzureOpenAIEmbeddings` de *LangChain*. Este modelo permite generar vectores compatibles con la búsqueda semántica de **Azure AI Search**, garantizando una integración nativa y eficiente con el servicio.
- **Prompt contextual:** Una vez recuperados los fragmentos relevantes, se construye un mensaje estructurado que combina la entrada del usuario con el contenido recuperado, siguiendo una plantilla específica adaptada al formato *ChatML* (OpenAI, 2023a).
- **Modelo generativo:** La respuesta final es generada mediante la API de `gpt-35-turbo` de *Azure OpenAI*, configurada como un agente generativo dentro del sistema. Este modelo ofrece una alta calidad en la generación de lenguaje natural y una latencia reducida, siendo especialmente adecuado para tareas interactivas en tiempo real.

Este enfoque permite al sistema responder a preguntas complejas sobre planificación de viajes, por ejemplo, “Quiero ir a Perth y moverme por la ciudad con opciones económicas” o “Quiero ir a un festival de Melbourne en la primera semana o tercera semana de julio”, combinando la potencia de los LLMs con acceso en tiempo real a información estructurada y actualizada. De esta forma, RAG actúa como un puente entre la generación flexible y el conocimiento factual, garantizando respuestas más útiles, relevantes y verificables para el usuario final.

3.2 › Estado del Arte

La integración de modelos generativos con mecanismos de recuperación de información (RAG) y arquitecturas multiagente representa una evolución significativa en

el diseño de sistemas inteligentes. Esta convergencia permite la creación de agentes especializados capaces de colaborar dinámicamente, razonar de forma distribuida y acceder a fuentes heterogéneas de información en tiempo real. El desarrollo de estas arquitecturas responde a desafíos fundamentales como la mitigación de alucinaciones, la necesidad de razonamiento contextual y la escalabilidad en entornos abiertos. En las secciones siguientes se revisan los principales enfoques del estado del arte, organizados según su contribución al entendimiento de los desafíos, fundamentos teóricos y soluciones aplicadas en la intersección entre LLM, RAG y orquestación multiagente.

3.2.1 › Desafíos en LLMs: alucinaciones y fiabilidad

El trabajo de Huang et al. (2025) ofrece una revisión exhaustiva sobre el fenómeno de las alucinaciones en los LLMs, abordando su clasificación, causas, impacto y estrategias de mitigación. Esta contribución resulta especialmente relevante en el contexto de este proyecto, ya que la presencia de alucinaciones representa uno de los principales desafíos a la hora de integrar LLMs en arquitecturas multiagente asistidas por RAG. La revisión proporcionada por los autores justifica la necesidad de incorporar mecanismos complementarios de verificación y recuperación de información, con el objetivo de mejorar la fiabilidad y factualidad de las respuestas generadas por los agentes. En este sentido, el presente trabajo se alinea con las recomendaciones del estado del arte al adoptar una arquitectura que mitiga este problema mediante la integración de componentes RAG dentro de la orquestación de agentes.

3.2.2 › Fundamentos para agentes autónomos basados en LLMs

Superado el análisis de los principales riesgos asociados al uso de LLMs, la literatura también ha explorado los fundamentos necesarios para su evolución hacia agentes autónomos. En esta línea, el grupo de Ferrag et al. (2025) realiza una revisión exhaustiva sobre la transición desde el razonamiento con LLMs hacia la construcción de agentes autónomos de IA. Aporta una categorización de las capacidades de razonamiento de los LLMs y las relaciona con el diseño de agentes que interactúan con entornos complejos, subrayando la necesidad de mecanismos de memoria, percepción, planificación y toma de decisiones. En el contexto de este trabajo, su relevancia radica en la sistematización de los componentes funcionales que deben integrar los agentes basados en LLM para alcanzar comportamientos autónomos, lo cual complementa el enfoque propuesto en nuestro trabajo, centrado en una orquestación distribuida y colaborativa con RAG como soporte a la toma de decisiones. La perspectiva funcional que ofrece esta revisión contribuye a contextualizar las decisiones de diseño adoptadas en la arquitectura multiagente desarrollada.

De forma complementaria, Barua (2024) amplía esta perspectiva con un análisis del ecosistema de agentes autónomos contruidos sobre modelos de lenguaje, destacando cómo la memoria, la planificación y la ejecución de acciones permiten a estos agentes desenvolverse de manera autónoma. Se examinan arquitecturas cognitivas emergentes, mecanismos de razonamiento como *Chain-of-Thought*, y el uso de herramientas externas. Además, se analizan marcos de evaluación como *AgentBench*, *WebArena* y *ToolLLM*, que permiten medir el rendimiento en tareas interactivas.

Aunque el enfoque no se centra específicamente en la integración con RAG, este trabajo sienta una base conceptual esencial para el diseño y análisis de sistemas multiagente con recuperación de información, facilitando su aplicación en dominios complejos y abiertos como el que aborda este proyecto.

En un plano más aplicado, Bousetouane (2025) introduce una propuesta integral para el diseño y despliegue de sistemas agénticos verticales, presentando una arquitectura modular basada en agentes LLM compuesta por componentes de memoria, habilidades cognitivas, herramientas y motores de razonamiento. Frente a la rigidez de plataformas Software como Servicio (SaaS) tradicionales, plantea un nuevo paradigma en el que agentes con inteligencia contextual y especializada permiten una automatización adaptativa y escalable de flujos de trabajo. Aunque su enfoque no se centra en la orquestación multiagente ni en RAG, proporciona una base conceptual sólida para entender cómo estructurar agentes especializados que pueden integrarse en ecosistemas más complejos, como los abordados en este trabajo.

3.2.3 › Hacia arquitecturas multiagente con RAG

Propuestas conceptuales y arquitecturas emergentes

A partir de los fundamentos previos, varios trabajos recientes han explorado directamente la intersección entre agentes y recuperación aumentada. Un ejemplo representativo es el trabajo de Singh et al. (2025), que presenta una revisión sistemática de enfoques que combinan RAG con sistemas de agentes. Analiza cómo la integración de múltiples agentes puede mejorar la recuperación, el razonamiento y la generación en tareas complejas mediante cadenas colaborativas de pensamiento (*chain-of-thought*) distribuidas entre agentes. Se describen arquitecturas comunes, como los agentes de recuperación, agentes de decisión y agentes generativos, así como las ventajas del enfoque distribuido frente a los sistemas monolíticos. Esta perspectiva resulta especialmente relevante para nuestro trabajo, al proporcionar una visión consolidada de los desafíos, estructuras y beneficios inherentes a la orquestación de sistemas RAG multiagente, alineándose estrechamente con el diseño modular implementado en KoalaRoute.

Profundizando en arquitecturas específicas, Zhu et al. (2024) proponen un sistema multiagente robusto para RAG denominado *ATM*, que mejora la precisión y fiabilidad del proceso generativo mediante una arquitectura adversarial. Este enfoque introduce agentes especializados para generación, evaluación y retroalimentación, organizados en un ciclo cerrado que permite refinar iterativamente las respuestas. Destaca por su capacidad para mitigar errores y alucinaciones típicos de los LLMs, y por su énfasis en una coordinación adaptativa entre agentes. Esta propuesta resulta relevante para este proyecto, ya que valida el uso de orquestación multiagente como mecanismo para mejorar la robustez y veracidad de los sistemas RAG.

En una línea similar pero con una aproximación distinta, Chen et al. (2025) presentan un enfoque novedoso que aplica aprendizaje por refuerzo multiagente para mejorar el rendimiento de los sistemas RAG. El modelo propuesto introduce una arquitectura en la que los agentes especializados cooperan mediante una política de coordinación optimizada, con el objetivo de seleccionar y generar respuestas más

precisas. Esta propuesta es especialmente relevante para este proyecto, ya que demuestra cómo la orquestación basada en múltiples agentes puede potenciar la generación contextual a partir de fuentes recuperadas, reforzando así la pertinencia y fiabilidad de las respuestas.

Implementaciones avanzadas y soluciones aplicadas

A nivel de implementación, Seabra et al. (2024) desarrollan un sistema de orquestación dinámica multiagente diseñado para resolver consultas complejas combinando fuentes estructuradas y no estructuradas. La propuesta introduce agentes especializados, incluyendo agentes Lenguaje de Consulta Estructurada (SQL), agentes RAG y agentes de enrutamiento, que son coordinados automáticamente según la naturaleza de la consulta. Uno de los principales avances consiste en la implementación de una ingeniería de prompts dinámica, que permite ajustar las instrucciones a cada contexto. Aplicado al dominio de gestión de contratos, el sistema demuestra mejoras sustanciales en precisión, relevancia y adaptabilidad frente a sistemas monolíticos.

Con un enfoque centrado en el razonamiento distribuido, Nguyen et al. (2025) introducen un novedoso enfoque multiagente, denominado *MA-RAG*, para mejorar el rendimiento de los sistemas RAG mediante el razonamiento colaborativo basado en cadenas de pensamiento. Propone una arquitectura en la que múltiples agentes especializados (por ejemplo, en recuperación, evaluación o síntesis) cooperan para generar respuestas más precisas y fundamentadas. Cada agente contribuye desde su rol con un razonamiento intermedio que es coordinado y fusionado a lo largo del proceso. El artículo demuestra que este razonamiento colaborativo entre agentes permite una mejor interpretación del contexto, reduciendo errores y aumentando la fiabilidad de las respuestas generadas. La validación experimental incluye benchmarks en tareas de QA complejas, donde se observa una mejora significativa frente a enfoques monolíticos. Este trabajo refuerza la viabilidad del diseño modular e inteligente de sistemas RAG mediante la orquestación estructurada de múltiples agentes.

Finalmente, de Aquino e Aquino et al. (2025) ofrecen una visión evolutiva del enfoque RAG, partiendo de sus formas tradicionales hacia arquitecturas más sofisticadas como el *Graph-RAG* y sistemas multiagente. Se describen los beneficios que estos enfoques avanzados aportan, como la especialización de tareas, la reducción de errores y la escalabilidad del sistema, al combinar agentes independientes coordinados con mecanismos de recuperación más estructurados. Asimismo, se destacan herramientas como *LangGraph* (L. Inc., 2023), que facilitan la implementación de flujos de trabajo agénticos con recuperación contextual, proponiendo así un marco conceptual unificado para futuras soluciones basadas en LLM.

La Tabla 3.1 resume los avances en la literatura de la orquestación de agentes de IA con el uso de RAG.

Tabla 3.1: Estado del arte en orquestación de agentes de IA con RAG y LLM

Paper	Enfoque principal	Aportes clave
LLMs: desafíos y fundamentos		
Huang et al. (2025)	Alucinaciones en LLMs	Revisión exhaustiva sobre el fenómeno de las alucinaciones, sus causas, tipos y técnicas de mitigación. Justifica el uso de RAG como mecanismo correctivo.
Ferrag et al. (2025)	Capacidades cognitivas en LLMs	Clasificación de tipos de razonamiento y análisis funcional de componentes requeridos para construir agentes autónomos a partir de LLMs.
Barua (2024)	Revisión de agentes autónomos basados en LLMs	Marco conceptual sobre arquitecturas cognitivas, uso de memoria, planificación, herramientas y benchmarks para evaluación.
Bousetouane (2025)	Agentes verticales con LLMs	Propuesta modular con memoria, herramientas, habilidades y razonamiento para construir agentes especializados adaptables a flujos de trabajo sectoriales.
Arquitecturas multiagente con RAG: propuestas y marcos generales		
Singh et al. (2025)	Revisión de sistemas agentic-RAG	Clasificación de arquitecturas con agentes especializados para recuperación, decisión y generación. Propone cadenas de razonamiento distribuidas.
de Aquino e Aquino et al. (2025)	Evolución de RAG hacia modelos multiagente	Comparativa entre enfoques clásicos y avanzados: RAG clásico, <i>Graph-RAG</i> y multiagente. Incluye herramientas como <i>LangGraph</i> para implementar estas arquitecturas.
Arquitecturas multiagente con RAG: soluciones específicas y aplicadas		
Zhu et al. (2024)	Sistema adversarial multiagente para RAG	Arquitectura <i>ATM</i> con agentes generadores, evaluadores y de feedback en bucle cerrado. Mejora la precisión y reduce alucinaciones mediante ajuste adversarial.
Chen et al. (2025)	Aprendizaje por refuerzo multiagente	Cooperación entre agentes mediante política de coordinación entrenada para optimizar la recuperación y generación de respuestas.
Seabra et al. (2024)	Orquestación dinámica multiagente	Integración de agentes RAG, SQL y de enrutamiento con adaptación dinámica de prompts según el contexto. Aplicado a gestión de contratos.
Nguyen et al. (2025)	Razonamiento colaborativo multiagente	Propuesta de <i>MA-RAG</i> con agentes especializados que cooperan mediante cadena de razonamiento compartida. Mejora la precisión en tareas de QA complejas.

Marco Metodológico

Este capítulo describe con detalle el proceso de desarrollo e implementación del sistema propuesto. Se abordan tres bloques fundamentales: la construcción de los agentes que componen el sistema, la definición del grafo que orquesta su ejecución, y el diseño de la interfaz web que permite la interacción con el usuario final. Cada apartado incluye las decisiones técnicas, las bibliotecas utilizadas y la arquitectura aplicada.

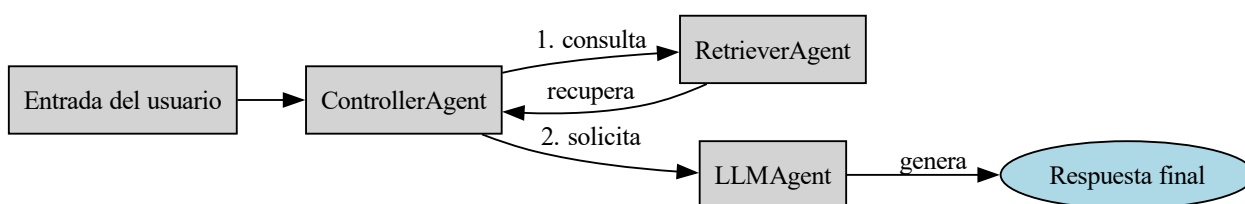


Figura 4.1: Flujo de ejecución del sistema con agentes orquestados en LangGraph

El sistema ha sido concebido como una aplicación web interactiva orientada a la generación automatizada de itinerarios turísticos personalizados para viajes en Australia. El flujo de funcionamiento parte de una consulta del usuario y termina con la visualización de una propuesta de itinerario construida por un LLM controlado mediante arquitectura RAG y orquestación multiagente. El recorrido se puede observar de manera esquemática en la Figura 4.1.

1. **Entrada del usuario:** a través de un formulario implementado con *Streamlit* (S. Inc., 2024), el usuario introduce sus preferencias de viaje: destino deseado, duración, presupuesto estimado, tipo de viaje (por ejemplo, en pareja o en grupo) e intereses temáticos (naturaleza, gastronomía, cultura, etc.).
2. **Construcción del prompt:** con esta información, se genera un prompt estructurado en formato *ChatML* (OpenAI, 2023a) que incluye un mensaje de sistema (definido por defecto o modificado según el caso), el contexto recuperado y la consulta del usuario. Este mensaje se construye dinámicamente con plantillas almacenadas en un archivo YAML.
3. **Ejecución del grafo de agentes:** el prompt se envía a una función que inicializa el estado y lanza la ejecución del grafo definido con *LangGraph* (L. Inc., 2023). Este grafo orquesta tres agentes principales: uno de control, uno de recuperación y uno generador. Dependiendo de si se ha consultado por contexto relevante, el grafo redirige la ejecución hacia el agente adecuado.
4. **Generación de la respuesta:** finalmente, el agente generador se comunica con un modelo de lenguaje desplegado en *Azure OpenAI* (Azure, 2023b) para producir una respuesta natural que contenga un itinerario día a día adaptado al perfil del usuario. Este resultado se muestra directamente en la interfaz.

Este enfoque permite una generación más precisa, controlada y personalizada, incorporando tanto capacidades generativas como acceso a información actualizada y filtrada según los intereses definidos.

El código fuente completo de la aplicación, incluyendo la definición del grafo, los agentes, las plantillas de prompts y la interfaz de usuario, se encuentra disponible públicamente en el repositorio. Este repositorio permite explorar el flujo de trabajo, el diseño de la aplicación y los módulos empleados en el desarrollo de la solución (Utica Babyak, 2025).

4.1 › Construcción de los agentes

El sistema propuesto se articula en torno a una arquitectura multiagente, donde cada componente tiene una responsabilidad claramente definida dentro del flujo conversacional. Esta división de responsabilidades permite no solo modularidad y escalabilidad, sino también una mayor trazabilidad en la lógica del sistema. Los agentes fueron implementados como clases en *Python* con un esquema común, orientado a facilitar su integración en el grafo de ejecución definido mediante *LangGraph*.

Cada agente recibe y devuelve un estado compartido, representado mediante la clase *AgentState*, que encapsula las claves *input*, *response* y *last_node*. Este esquema garantiza la coherencia entre interacciones y permite encadenar los resultados de cada agente de forma controlada.

4.1.1 › ControllerAgent: coordinación del flujo RAG

El *ControllerAgent* actúa como el cerebro del sistema. Su función principal es construir el mensaje que se enviará a los siguientes agentes del flujo, componiendo

un prompt estructurado a partir del mensaje del usuario y del contexto disponible, si lo hubiera. Para ello, se apoya en funciones auxiliares definidas en el módulo `prompt_utils` (véase el Listado 12), que permiten cargar plantillas de prompt desde archivos YAML y generar mensajes en formato *ChatML* compatibles con la API de *Azure OpenAI*. Este agente se encarga también de adaptar el contenido del mensaje del sistema para cada situación, incluyendo instrucciones específicas que orientan el comportamiento del modelo de lenguaje. La lógica de transición entre agentes, es decir, la decisión de si se debe recuperar contexto o generar una respuesta, se describe más adelante al detallar la definición del grafo.

Además, el **ControllerAgent** contempla el uso de un *prompt de fallback* cuando la recuperación no produce resultados útiles. En ese caso, el prompt del sistema se complementa con instrucciones adicionales para que el modelo genere una respuesta basada en conocimiento general, manteniendo el tono y el formato establecidos. Su implementación completa puede consultarse en el Listado 6.

Prompt Utils: construcción dinámica de prompts

El módulo `prompt_utils.py` contiene funciones utilitarias que permiten construir mensajes en formato *ChatML* a partir de plantillas predefinidas. Estas plantillas están almacenadas en un archivo YAML externo, lo que permite modificar los mensajes del sistema sin necesidad de alterar el código fuente.

Entre las funciones más relevantes se encuentran:

- **build_chatml_messages**: Construye una lista de mensajes en formato **system** y **user**, a partir de la entrada del usuario, el contexto recuperado y un prompt del sistema (por defecto o modificado).
- **load_prompt** y **load_formatted_prompt**: Permiten cargar una plantilla por clave desde el archivo YAML, aplicando si es necesario formateo dinámico con variables como `days`, `budget` o `interests`.
- **extract_user_interests_from_prompt**: Esta función permite extraer la lista de intereses del usuario que fueron incluidos en el prompt final, comparándolos con la plantilla y extrayendo dinámicamente el fragmento correspondiente.

Gracias a este módulo, los prompts enviados al modelo se mantienen consistentes, estructurados y personalizables, facilitando la adaptación del sistema a distintos dominios o estilos conversacionales. El módulo se puede consultar en el Listado 12.

4.1.2 › RetrieverAgent: recuperación semántica contextualizada

El **RetrieverAgent** es el encargado de realizar la recuperación de documentos relevantes desde un almacén vectorial implementado sobre *Azure Cognitive Search* (Azure, 2023a). Este componente utiliza un modelo de *embeddings* proporcionado por *Azure OpenAI*, cuyo despliegue y configuración están definidos en el archivo de entorno y gestionados desde el archivo de configuración principal, accesible en el Listado 2. Este modelo garantiza la compatibilidad total con el índice vectorial definido en Azure Search.

El flujo de recuperación incluye los siguientes pasos:

1. **Extracción de intereses temáticos del usuario:** se utiliza la función `extract_user_interests_from_prompt`, definida en el módulo `prompt_utils.py` (véase el Listado 12), para extraer automáticamente los intereses que el usuario ha seleccionado al rellenar el formulario web. Esta función compara el prompt dinámicamente instanciado con su plantilla base y localiza la parte concreta que corresponde a la variable `{interests}`, devolviendo una lista de términos clave (por ejemplo, `[“naturaleza”, “cultura”, “aventura”]`) normalizados en minúscula. Estos intereses son los que se utilizarán como filtro temático para refinar la búsqueda semántica.
2. **Consulta al índice vectorial:** con la consulta original del usuario se realiza una búsqueda por similitud utilizando la función `similarity_search` del almacén vectorial. Esta operación devuelve un conjunto de documentos representados como objetos enriquecidos con metadatos, incluyendo título y sección temática.
3. **Filtrado temático:** los documentos recuperados se filtran conservando únicamente aquellos cuya temática (almacenada en el metadato `category`) contiene alguna coincidencia con los intereses del usuario extraídos previamente. Este paso reduce el ruido y mejora la precisión de la información suministrada al modelo generador.
4. **Evaluación por similitud vectorial:** sobre los documentos que han superado el filtro temático, se calcula la similitud de coseno entre el *embedding* del texto del documento y el *embedding* de la consulta del usuario. Solo se seleccionan para el contexto aquellos documentos que superen un umbral de similitud predefinido, asegurando relevancia semántica.
5. **Composición del contexto:** los documentos seleccionados se formatean como bloques Markdown y se concatenan en una cadena final que será insertada como contexto durante la fase de generación.

Listing 1: Estructura simplificada del índice vectorial de Azure Cognitive Search

```

1 {
2   "fields": [
3     { "name": "id", "type": "Edm.String", "key": true },
4     { "name": "title", "type": "Edm.String", "searchable": true },
5     { "name": "section", "type": "Edm.String", "searchable": true },
6     { "name": "category", "type": "Collection(Edm.String)", "searchable": true
7       ↪ },
8     { "name": "content", "type": "Edm.String", "searchable": true },
9     { "name": "content_vector", "type": "Collection(Edm.Single)", "dimensions":
10       ↪ 3072 }
11   ]
12 }
```

Los documentos utilizados en esta recuperación están definidos en un índice vectorial configurado específicamente en *Azure Cognitive Search*. En el Listado 1 se muestra

una versión simplificada de la estructura del índice, centrada en los campos relevantes que se recuperan durante una consulta semántica.

Estos campos permiten almacenar metadatos útiles (como el título del documento o su sección), representar el contenido textual y realizar consultas vectoriales eficientes mediante el campo `content_vector`. El campo `category`, opcionalmente asignado durante la carga, permite enriquecer las capacidades de filtrado si se desea utilizar anotaciones adicionales.

Este agente devuelve un bloque de texto estructurado que puede ser incorporado directamente como contexto dentro del mensaje del usuario, o bien dejarse vacío si no se encuentra información útil (activando el modo de *fallback* en el siguiente agente). Su implementación completa puede consultarse en el Listado 7.

Vector: Inicialización del almacén vectorial

El módulo *vector.py* se encarga de inicializar los dos componentes clave para la recuperación semántica del sistema: el modelo de *embeddings* y el almacén vectorial basado en *Azure Cognitive Search*.

En primer lugar, se inicializa una función de *embeddings* proporcionada por *Azure OpenAI*, mediante el uso de la clase `AzureOpenAIEmbeddings`. Esta función se configura a través de variables definidas en el archivo `.env`, incluyendo el endpoint de Azure OpenAI, la clave API correspondiente y el nombre del despliegue asociado. El modelo seleccionado (por ejemplo, `text-embedding-3-large`) convierte tanto consultas como fragmentos documentales en vectores de alta dimensionalidad, optimizados para la búsqueda semántica.

Posteriormente, se inicializa el almacén `AzureSearch`, especificando:

- El endpoint del servicio de búsqueda de Azure.
- La clave de acceso al servicio.
- El nombre del índice vectorial configurado.
- La función de *embeddings* gestionada por Azure OpenAI.

El objeto resultante *vector_store* es utilizado directamente por el `RetrieverAgent` para consultar y filtrar documentos relevantes en función de su similitud con la consulta del usuario. El módulo completo puede consultarse en el Listado 13.

4.1.3 › LLMAgent: generación de respuesta con LLM

El `LLMAgent` es responsable de comunicarse con el modelo de lenguaje desplegado en *Azure OpenAI* (Azure, 2023b). Utiliza la API del servicio para enviar mensajes en formato *ChatML*, previamente contruidos por el controlador. Estos mensajes contienen un rol *system* (que impone las reglas del sistema), un posible bloque de contexto (cuando se ha recuperado información relevante), y el mensaje del usuario.

El agente utiliza una función utilitaria para llamar a la API y obtiene una respuesta textual que se almacena nuevamente en el estado, junto con una marca que indica que

el nodo ejecutado ha sido el generador. Este resultado se mostrará posteriormente en la interfaz de usuario. Su implementación puede consultarse en el Listado 8.

LLM: Conexión con Azure OpenAI

El módulo *llm.py* encapsula la lógica necesaria para interactuar con el modelo de lenguaje desplegado en la plataforma *Azure OpenAI*. Este archivo inicializa el cliente de la API utilizando las credenciales y parámetros definidos en la configuración del proyecto, como el nombre del despliegue, el endpoint o el número máximo de tokens por respuesta.

El principal método expuesto es:

```
36 def call_openai_chat(prompt_messages: list[dict]) -> str:
```

Esta función recibe una lista de mensajes en formato *ChatML*, que incluyen los roles *system* y *user*, y devuelve el contenido textual generado por el modelo. La encapsulación de esta llamada en una función independiente permite mantener el resto del sistema desacoplado de detalles técnicos como el cliente, el endpoint o el nombre del modelo. La función es invocada directamente por el agente generador (LLMAgent) para producir las respuestas personalizadas. Finalmente, el módulo se puede consultar en el Listado 11.

Cada agente ha sido implementado siguiendo principios de claridad, separación de responsabilidades y compatibilidad con sistemas de ejecución distribuidos, permitiendo su extensión a nuevos dominios o adaptaciones futuras.

4.2 › Definición del grafo de ejecución

La coordinación entre agentes se articula mediante un grafo de ejecución definido con la biblioteca *LangGraph*, una herramienta que permite representar flujos conversacionales como grafos dirigidos, en los que cada nodo corresponde a un agente funcional y las aristas modelan transiciones condicionales sobre el estado compartido.

4.2.1 › Modelo de estado compartido: AgentState

Antes de construir el grafo, se define un modelo de estado compartido que permite intercambiar información entre los distintos nodos. Para ello, se crea el tipo *AgentState*, implementado como un *TypedDict* en Python (véase el Listado 10). Este estado contiene cuatro claves principales:

- **input**: el mensaje original introducido por el usuario.
- **response**: el resultado parcial o final generado por los agentes.

- **last_node**: el nombre del último nodo ejecutado, necesario para la lógica del controlador.
- **retrieved_docs**: lista de metadatos de los documentos recuperados durante la fase de recuperación.

Este diseño permite mantener un flujo coherente y transparente entre interacciones, facilitando la trazabilidad, el control y la depuración del sistema multiagente. La inclusión de los documentos recuperados en el estado también permite realizar evaluaciones posteriores y explicar el razonamiento del sistema de forma más clara.

4.2.2 › Construcción del grafo: `build_langgraph_controller_flow`

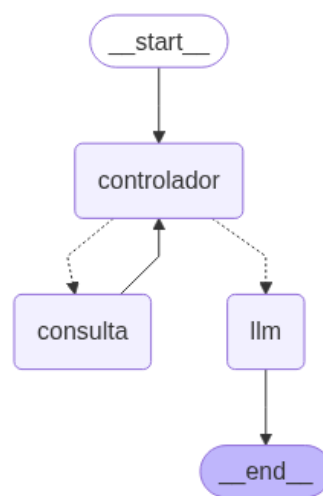


Figura 4.2: Flujo de ejecución entre agentes orquestados en LangGraph

La construcción del grafo se lleva a cabo en el módulo `graph.py`, donde se define la función `build_langgraph_controller_flow`, encargada de ensamblar los nodos del flujo (véase Listado 9).

Cada nodo del grafo está vinculado a un agente previamente implementado:

- **controlador**: nodo principal asociado al `ControllerAgent`, que determina la transición siguiente.
- **consulta**: nodo vinculado al `RetrieverAgent`, encargado de recuperar contexto externo.
- **llm**: nodo final que invoca al `LLMAgent` para generar la respuesta.

El flujo se inicia siempre en el nodo `controlador`, que, en función del valor de `last_node` presente en el estado, decide si debe dirigirse al recuperador o pasar directamente al generador. Esta lógica se encapsula en la función auxiliar `next_node`. Si se trata de la primera invocación del controlador, el flujo se redirige al recuperador para obtener contexto relevante. En cambio, si ya se ha vuelto del recuperador, el controlador deriva la ejecución hacia el generador, permitiendo así una planificación flexible en función de la información disponible.

Este diseño permite capturar reglas conversacionales de forma declarativa, facilita la incorporación de nuevos nodos o rutas condicionales, y ofrece una base robusta y extensible para la orquestación multiagente.

Gracias a esta estructura, el sistema no solo controla la ejecución secuencial de los agentes, sino que también habilita reinicios controlados del flujo, decisiones dinámicas y una trazabilidad clara del estado en cada paso de la conversación.

4.3 › Diseño de la interfaz web

La interacción entre el sistema y el usuario final se implementa mediante una interfaz web desarrollada con la biblioteca *Streamlit*. Este marco permite construir aplicaciones web en *Python* de forma rápida y declarativa, sin necesidad de escribir código en *HTML*, *CSS* o *JavaScript*. En el contexto de este proyecto, la interfaz ofrece una experiencia simple e intuitiva para solicitar itinerarios personalizados y visualizar las respuestas generadas por el sistema multiagente.

4.3.1 › Estructura del formulario de entrada

La interfaz se organiza en bloques que permiten al usuario introducir información relevante para la generación del itinerario:

- **Consulta libre:** un campo de texto donde el usuario puede describir con sus propias palabras qué le gustaría hacer, qué lugares desea visitar o qué tipo de experiencia busca.
- **Parámetros estructurados:** se incluyen selectores y listas para especificar:
 - Duración del viaje (en días).
 - Presupuesto estimado (gama económica, media o alta).
 - Tipo de viaje (por ejemplo, individual, en pareja, en familia, etc.).
 - Intereses temáticos (naturaleza, cultura, gastronomía, aventura, etc.).

Estos valores se utilizan para construir dinámicamente un **prompt estructurado** que se adapta al formato requerido por el modelo de lenguaje. El prompt se genera a partir de una plantilla definida en YAML ubicada en `prompt.yaml` (véase el Listado 3), en la que se interpolan las preferencias del usuario.

```

27 prompt_base: | # 57~ tokens
28     Quiero un itinerario día a día para un viaje de {days} días,
29     con presupuesto {budget}, viajando en formato {travel_type},
30     centrado en los siguientes intereses: {interests}.
31     Detalles adicionales:
  
```

4.3.2 › Visualización de tokens y validación

Uno de los aspectos clave para el correcto funcionamiento del sistema es el control del número de **tokens** que puede manejar el modelo de lenguaje. Para ello, la interfaz:

- Calcula los tokens utilizados por la consulta del usuario.
- Visualiza una barra de progreso que indica cuánto espacio queda disponible respecto al límite máximo de tokens.
- Muestra alertas si el usuario excede el número permitido o si el campo de entrada está vacío.

Esta validación anticipada mejora la experiencia del usuario y evita errores durante la fase de ejecución del grafo.

4.3.3 › Envío del prompt y presentación del resultado

Una vez validado el input, el usuario puede pulsar el botón **Generar itinerario**. En ese momento:

1. Se construye el prompt final concatenando la plantilla formateada con la consulta libre del usuario.
2. Este prompt se envía al grafo multiagente a través de la función `run_prompt`, definida en el módulo `runner.py` (véase el Listado 18).
3. El resultado generado por el modelo se muestra directamente en la interfaz web, en formato Markdown enriquecido, que puede incluir encabezados, listas o enlaces.

4.3.4 › Gestión de configuración desde archivos YAML

Tanto los textos de los menús desplegables como los valores de los campos estructurados se definen en un archivo de configuración `ui_options.yaml` (véase el Listado 5). Esto permite mantener una separación entre la lógica de presentación y los datos, facilitando la traducción, la modificación de la interfaz o la adaptación a otros contextos sin alterar el código fuente.

4.3.5 › Ventajas del enfoque con Streamlit

La elección de *Streamlit* como framework de desarrollo presenta varias ventajas clave:

- **Despliegue sencillo:** las aplicaciones pueden ejecutarse localmente o desplegarse en servicios cloud con un solo comando.
- **Interactividad reactiva:** los cambios en los formularios o parámetros actualizan el estado de la aplicación sin necesidad de recargar la página.
- **Compatibilidad con Markdown y HTML:** permite mostrar las respuestas generadas en un formato legible y visualmente atractivo.

- **Integración con Python nativo:** facilita la comunicación con módulos internos, como la construcción de prompts o la ejecución del grafo.

4.3.6 › Referencias cruzadas con el código fuente

La implementación completa de la interfaz se encuentra en el archivo `app.py` (véase Listado 17), donde se puede observar la lógica de construcción del formulario, validación de tokens y visualización de resultados. Este módulo actúa como punto de entrada de la aplicación, integrando todos los componentes desarrollados.

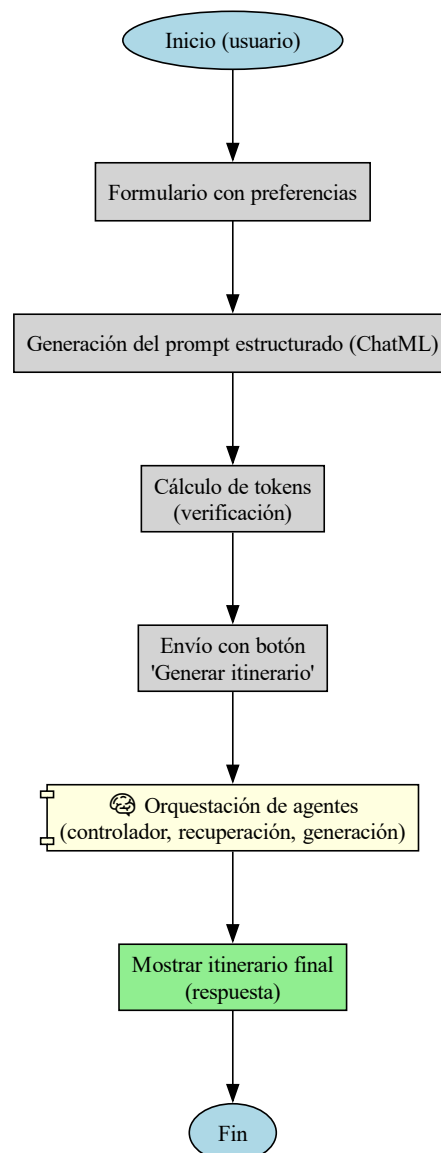


Figura 4.3: Flujo de interacción entre el usuario y el sistema desde la interfaz web

Como se muestra en la Figura 4.3, el flujo general sigue los siguientes pasos:

1. El usuario introduce sus preferencias mediante la interfaz.
2. Se construye dinámicamente un prompt estructurado a partir de las entradas.
3. El prompt es enviado al grafo de agentes implementado con *LangGraph*.
4. El sistema orquesta la recuperación de contexto y la generación de respuesta.
5. La respuesta final es mostrada en la interfaz como itinerario detallado.

Presentación de análisis, resultados y discusión

Este capítulo presenta una evaluación crítica del sistema desarrollado, combinando tanto aspectos cuantitativos como cualitativos. El objetivo principal es determinar en qué medida la solución propuesta cumple con los objetivos planteados, y valorar su desempeño en contextos realistas de uso desde una perspectiva técnica y funcional.

En primer lugar, se detalla la metodología empleada para evaluar la eficacia del sistema, que incluye métricas cuantitativas orientadas a la recuperación de información relevante (como el *Recall adaptativo* o la *Cobertura Temática*) y a la calidad semántica de las respuestas generadas por el modelo de lenguaje. Además, se complementa con una evaluación cualitativa centrada en analizar la coherencia narrativa de los itinerarios, su relevancia respecto a los intereses del usuario y el grado de personalización contextual alcanzado.

Posteriormente, se presentan los resultados obtenidos a partir de distintos escenarios de prueba, organizados según criterios como la precisión del contexto recuperado, y la calidad y adecuación del itinerario generado.

Este enfoque mixto permite validar el comportamiento del sistema de forma objetiva, reproducible y también interpretativa, identificando sus fortalezas, limitaciones y oportunidades de mejora futura sobre una base empírica y observacional.

Los resultados aquí expuestos constituyen una base sólida para argumentar la viabilidad del enfoque basado en orquestación de agentes con RAG en aplicaciones prácticas, y sirven como punto de partida para trabajos posteriores centrados en dominios similares o ampliaciones del sistema actual.

5.1 › Metodología de evaluación

La evaluación del sistema propuesto se ha abordado desde una perspectiva técnica, centrada en el análisis de métricas cuantitativas orientadas a validar el rendimiento funcional de los componentes clave, complementadas con métricas cualitativas que permiten valorar aspectos como la coherencia, relevancia y adecuación contextual de las respuestas generadas.

Dado el carácter modular del sistema, estructurado en torno a una arquitectura de agentes orquestados mediante un grafo, la evaluación se ha dividido en dos bloques principales:

- Evaluación de la **eficacia en la recuperación de contexto**, centrada en el rendimiento del **RetrieverAgent**, medida a través de la cobertura temática y la relevancia semántica de los documentos seleccionados.
- Evaluación de la **calidad estructural y adecuación de los itinerarios generados**, vinculada al **LLMAgent**, mediante verificaciones automáticas del formato, la inclusión de los intereses del usuario y la coherencia general del contenido generado.

Este enfoque cuantitativo permite una validación sistemática del comportamiento del sistema en condiciones controladas, sin depender de valoraciones subjetivas.

Para la evaluación de los escenarios, se han suministrado documentos específicos sobre las diez ciudades más pobladas de Australia, que conforman la base de conocimiento utilizada por el sistema de recuperación. El contenido completo de estos documentos está disponible en el repositorio asociado al proyecto (Utica Babyak, 2025).

5.1.1 › Evaluación de la recuperación de contexto

La primera dimensión evaluada es la capacidad del sistema para recuperar fragmentos de información relevantes y temáticamente coherentes con la consulta del usuario. Para ello, se han utilizado métricas estándar en tareas de recuperación semántica:

- **Precisión@k (P@k)**: fracción de documentos relevantes entre los k primeros resultados retornados por el **RetrieverAgent**.
- **Cobertura temáticamente alineada**: porcentaje de secciones recuperadas que coinciden con los intereses extraídos desde el prompt dinámico.
- **Similitud media de coseno**: medida del alineamiento semántico entre la consulta del usuario y los textos recuperados, basada en los vectores de embedding generados.

Los datos evaluados se han obtenido a partir de consultas simuladas con distintos perfiles de viaje, analizando el conjunto de documentos devueltos antes de la fase de generación.

5.1.2 › Evaluación de la calidad de las respuestas generadas

La segunda dimensión evaluada se refiere a la calidad, coherencia y personalización de los itinerarios generados por el LLM_{Agent}. Dado que esta tarea implica generación de lenguaje natural, se han utilizado métricas cualitativas complementadas con observación experta:

- **Coherencia narrativa:** grado en que el itinerario generado sigue una lógica temporal, geográfica y semántica adecuada.
- **Relevancia del contenido:** inclusión de recomendaciones útiles y ajustadas a los intereses y restricciones indicadas por el usuario.
- **Adaptación al contexto:** uso adecuado del contexto proporcionado por el Retriever_{Agent} (cuando se ha recuperado) en la construcción de la respuesta final.

Estas propiedades se han valorado utilizando escalas de 1 a 5 en distintos escenarios simulados, con observación directa y anotaciones de calidad.

En conjunto, esta metodología de evaluación busca capturar tanto el rendimiento técnico de los componentes fundamentales como la percepción global de utilidad y calidad, clave para validar la propuesta en escenarios de uso real.

5.1.3 › Diseño de los escenarios de prueba

Para la evaluación del sistema se definieron distintos perfiles de usuario simulando casos de uso realistas. Estos perfiles se construyeron a partir de combinaciones válidas de los parámetros recogidos por la interfaz: duración del viaje, presupuesto, tipo de viaje e intereses temáticos. Además, se especificó el input del usuario para formular su consulta con mayor especificidad, permitiendo evaluar la capacidad del sistema para adaptar las recomendaciones a indicaciones particulares.

Tabla 5.1: Escenarios de prueba definidos para el sistema

ID	Duración (días)	Intereses	Presupuesto y tipo de viaje	Ciudad	Entrada
1	3	Naturaleza, Clima agradable	Económico, pareja	Brisbane o alrededores	Queremos visitar zonas verdes, parques naturales o jardines botánicos cerca de Brisbane.
2	7	Cultura, Historia y curiosidades, Turismo urbano	Medio, individual	Melbourne y similares	Quiero explorar ciudades históricas con museos y sitios de interés cultural, especialmente en Melbourne.

ID	Duración (días)	Intereses	Presupuesto y tipo de viaje	Ciudad	Entrada
3	6	Gastronomía, Playas, Clima agradable	Medio, pareja	Gold Coast o al- rededores	Queremos disfrutar de buena comida lo- cal y relajarnos en la playa. Preferiblemen- te en Gold Coast o alrededores.
4	7	Festivales y eventos, Cultura, Turismo urbano	Alto, grupo	Sídney, Melbourne y similares	Planeamos asistir a eventos culturales im- portantes en ciudades como Sídney o Mel- bourne.
5	4	Naturaleza, Aventura, Viaje económico	Económico, individual	-	Me gustaría explorar rutas naturales menos concurridas, con acti- vidades al aire libre, sin gastar mucho.
6	5	Turismo urbano, Cultura, Playas, Naturaleza	Medio, pareja	-	Queremos una combi- nación equilibrada de ciudad, cultura y na- turaleza. Nos interesa incluir algo de playa también.
7	5	Aventura, Transporte público, Naturaleza	Alto, familia	-	Buscamos actividades divertidas y seguras para niños en contac- to con la naturale- za. Nos movemos en transporte público.
8	4	-	Económico, mochilero	-	Solo quiero un itine- rario barato para re- correr varias ciudades por mi cuenta.

Los escenarios, que se pueden encontrar en la Tabla 5.1, fueron ejecutados cada uno como una simulación independiente utilizando los formularios definidos en la aplicación web. Los escenarios fueron ejecutados a través de la interfaz web de evaluación del sistema, recopilando tanto el contexto recuperado como el itinerario propuesto, junto con las métricas asociadas al desempeño del sistema. De este modo, se logró una evaluación integral, homogénea y reproducible, que combinó análisis cuantitativos con apreciaciones cualitativas.

5.2 › Eficacia en la recuperación de contexto

La eficacia del **RetrieverAgent** se evaluó a partir de tres métricas clave, diseñadas para capturar diferentes aspectos del proceso de recuperación: la exhaustividad (*Recall@k*), la alineación temática con los intereses del usuario (Cobertura temática) y la proximidad semántica entre la consulta y los documentos seleccionados (Precisión semántica). A continuación, se detalla la definición de cada métrica y se presentan los resultados obtenidos.

5.2.1 › Métricas de evaluación

- **Recall@k**: mide la proporción de documentos relevantes efectivamente recuperados entre los k primeros resultados. En este caso, se considera que un documento es relevante si aborda explícitamente alguno de los temas solicitados por el usuario.

$$\text{Recall@k} = \frac{\text{número de documentos relevantes recuperados}}{\text{número total de documentos relevantes existentes}} \quad (5.1)$$

- **Cobertura temática**: refleja el grado de correspondencia entre los intereses del usuario y los temas cubiertos por los documentos seleccionados. Esta métrica se calcula como la fracción de intereses detectados en el prompt que están representados en al menos un documento recuperado.

$$\text{Cobertura temática} = \frac{\text{intereses cubiertos en la recuperación}}{\text{intereses totales del usuario}} \quad (5.2)$$

- **Precisión semántica**: cuantifica la similitud entre la consulta del usuario y los documentos recuperados utilizando la similitud de coseno entre los vectores de embedding. Se considera que un documento es semánticamente adecuado si su similitud con la consulta supera un umbral, que en el caso de este proyecto se escogió como 0,4.

$$\text{sim}(\vec{q}, \vec{d}) = \frac{\vec{q} \cdot \vec{d}}{\|\vec{q}\| \cdot \|\vec{d}\|} \quad (5.3)$$

5.2.2 › Resultados

Los resultados por escenario se resumen en las tablas 5.2 a 5.9, y muestran una variabilidad significativa en el rendimiento del **RetrieverAgent** según el tipo de perfil de usuario y los intereses solicitados. A continuación, se analizan en detalle los valores obtenidos para cada métrica.

Tabla 5.2: Resultados de evaluación del escenario 1

ID	Recall adaptativo	Cobertura Temática	Coherencia Semántica
1	1.00	1.00	0.49

En el escenario de la Tabla 5.2, el sistema alcanzó un desempeño óptimo. Todos los documentos relevantes fueron recuperados (**recall adaptativo** = **1.00**), cubriendo completamente los intereses temáticos del usuario (**cobertura temática** = **1.00**) y con una **coherencia semántica elevada (0.49)**. Esto indica una recuperación eficaz para una consulta bien delimitada en temática y localización.

Tabla 5.3: Resultados de evaluación del escenario 2

ID	Recall adaptativo	Cobertura Temática	Coherencia Semántica
2	0.50	1.00	0.48

Aunque se logró cubrir todos los intereses del usuario en los resultados del escenario de la Tabla 5.3 (**cobertura temática** = **1.00**), el sistema solo recuperó la mitad de los documentos relevantes esperados (**recall adaptativo** = **0.50**). La **coherencia semántica se mantuvo alta (0.48)**, lo que sugiere que los documentos seleccionados fueron pertinentes aunque insuficientes en cantidad.

Tabla 5.4: Resultados de evaluación del escenario 3

ID	Recall adaptativo	Cobertura Temática	Coherencia Semántica
3	0.67	1.00	0.5

En el escenario de la Tabla 5.4, el sistema recuperó una parte importante de la información esperada (**recall adaptativo** = **0.67**) y cubrió la totalidad de los intereses (**cobertura temática** = **1.00**), con una **coherencia semántica de 0.50**. Este escenario muestra un rendimiento sólido en consultas temáticas amplias pero bien formuladas.

Tabla 5.5: Resultados de evaluación del escenario 4

ID	Recall adaptativo	Cobertura Temática	Coherencia Semántica
4	0.33	1.00	0.50

Dados los resultados del escenario de la Tabla 5.5, pese a una caída en el **recall adaptativo (0.33)**, el sistema logró una **cobertura temática completa (1.00)** y mantuvo una **coherencia semántica alta (0.50)**. La diversidad de localizaciones e intereses en la consulta puede haber dificultado una recuperación más completa.

Tabla 5.6: Resultados de evaluación del escenario 5

ID	Cobertura Temática	Coherencia Semántica
5	0.33	0.41

A partir de aquí, se deja de evaluar con la métrica de **recall adaptativo** puesto que los escenarios dejan de especificar ciudades en el input del usuario, y es muy complicado determinar qué documentos se espera que el sistema recupere.

Viendo los resultados de la Tabla 5.6, en este caso, el sistema solo cubrió una parte de los intereses indicados (**cobertura temática** = **0.33**) y presentó una **coherencia semántica moderada (0.41)**. La falta de especificidad geográfica y el umbral especificado en la similitud del coseno parecen haber dificultado una recuperación precisa.

Tabla 5.7: Resultados de evaluación del escenario 6

ID	Cobertura Temática	Coherencia Semántica
6	0.75	0.4

Tal y como se observa en la Tabla 5.7, el sistema recuperó fragmentos que cubrían el 75 % de los intereses planteados, aunque con una **coherencia semántica algo limitada (0.40)**. La consulta combinaba múltiples dimensiones temáticas, lo que pudo dispersar la relevancia de los documentos seleccionados.

Tabla 5.8: Resultados de evaluación del escenario 7

ID	Cobertura Temática	Coherencia Semántica
7	0.00	0.00

El escenario de la Tabla 5.8 fue el escenario con peor desempeño: el sistema no logró recuperar ningún contenido relevante ni temáticamente (**cobertura** = **0.00**) ni semánticamente (**similitud** = **0.00**). La ausencia de localización explícita y la combinación inusual de intereses dificultaron el proceso de recuperación.

Tabla 5.9: Resultados de evaluación del escenario 8

ID	Coherencia Semántica
8	0.00

Finalmente, viendo los resultados de la Tabla 5.9, al no haber especificado el usuario una lista de intereses, la consulta se volvió extremadamente genérica e impidió una recuperación significativa, resultando en una **coherencia semántica nula (0.00)**. Este caso refleja la necesidad de entradas más concretas para guiar adecuadamente la recuperación contextual. Nótese que al no haber especificado una lista de intereses, no se ha evaluado con la métrica de **cobertura temática** puesto que no tenemos intereses.

5.3 › Evaluación de la calidad de las respuestas generadas

Además de la eficacia en la recuperación de contexto, se evaluó la calidad de los itinerarios generados por el **LLMAgent**, con el objetivo de valorar la utilidad, coherencia y adecuación de las respuestas desde el punto de vista del usuario final. Dado que se trata de una tarea de generación de lenguaje natural, la evaluación se centró en criterios cualitativos, complementados con observación directa sobre escenarios simulados.

5.3.1 › Métricas de evaluación

La calidad de las respuestas se analizó atendiendo a tres dimensiones fundamentales, cada una de ellas evaluada de forma independiente mediante una escala ordinal de 1 (muy deficiente) a 5 (excelente). Estas métricas permitieron capturar aspectos clave de la generación de itinerarios personalizados.

- **Coherencia narrativa:** esta métrica evalúa si el itinerario sigue una estructura lógica en términos temporales, geográficos y temáticos. Se valora que las actividades estén distribuidas de forma consistente a lo largo de los días propuestos, que las transiciones entre ubicaciones sean viables y que las recomendaciones guarden una conexión razonable entre sí.
- **Relevancia del contenido:** se refiere al grado en que las recomendaciones incluidas en el itinerario se alinean con los intereses, restricciones y preferencias indicadas por el usuario en el prompt. Una puntuación alta implica que el agente ha interpretado correctamente los intereses temáticos y los ha reflejado de manera explícita y útil.
- **Adaptación al contexto:** esta métrica mide hasta qué punto el **LLMAgent** ha integrado correctamente el contenido proporcionado por el **RetrieverAgent** (cuando ha estado disponible). Se valora la incorporación explícita de fragmentos del contexto recuperado, así como su uso adecuado para enriquecer o justificar las propuestas del itinerario.

Estas dimensiones permiten evaluar no solo la calidad lingüística de las respuestas, sino también su alineación con el input del usuario y su dependencia del conocimiento extraído, lo cual es especialmente relevante en arquitecturas basadas en RAG.

5.3.2 › Resultados

Escenario 1

En el escenario de la Tabla A.1, el sistema recuperó los siguientes documentos: **Brisbane#Descripción General**, **Brisbane#Clima**, **Brisbane#Principales Atracciones**, **Sunshine Coast#Principales Atracciones**

Estos documentos proporcionaron información clave sobre la ciudad de Brisbane, su clima, sus principales puntos de interés turístico, así como atracciones relevantes

en Sunshine Coast, una región cercana muy alineada con los intereses del usuario (naturaleza y entorno agradable).

La salida generada en este escenario presenta una propuesta estructurada y razonablemente adaptada a la petición del usuario. A continuación, se valoran las métricas cualitativas definidas.

■ **Coherencia narrativa: 5/5**

El itinerario está claramente dividido en tres días, con una secuencia lógica y realista de actividades que respeta un ritmo equilibrado de exploración. Las actividades de cada día comienzan por la mañana, evolucionan por la tarde y culminan con sugerencias de cena por la noche. Los desplazamientos están bien justificados (por ejemplo, se sugiere alquilar coche para visitar la Sunshine Coast), lo cual refuerza la viabilidad temporal y geográfica de la propuesta.

■ **Relevancia del contenido: 5/5**

Las actividades propuestas se alinean perfectamente con los intereses del usuario: se incluyen visitas a *zonas verdes* como *South Bank Parklands*, *Brisbane Botanic Gardens* y *Mount Coot-tha*, así como experiencias de *naturaleza* en Sunshine Coast (Glass House Mountains, Noosa Heads). El clima también se menciona implícitamente en la elección de actividades al aire libre, y la propuesta es adecuada para un viaje de pareja con presupuesto económico.

■ **Adaptación al contexto: 4.5/5**

El itinerario integra de forma efectiva muchos de los elementos presentes en el contexto recuperado. Lugares como *South Bank Parklands*, *Riverwalk*, *Mount Coot-tha*, *Lone Pine Koala Sanctuary* y *Brisbane Botanic Gardens* están correctamente utilizados. También se hace referencia a la Sunshine Coast y sus atracciones clave. No obstante, se podría haber citado explícitamente alguna característica del clima subtropical o la playa artificial de South Bank (Streets Beach), que también figuraban en el contexto. Por esta razón, la puntuación no es completamente perfecta.

Escenario 2

En el escenario de la Tabla A.2, el sistema recuperó los siguientes documentos: **Melbourne#Descripción General**, **Melbourne#Principales Atracciones**, **Canberra#Principales Atracciones**, **Sídney#Principales Atracciones**.

Estos documentos proporcionaron información relevante sobre las tres ciudades incluidas en el itinerario: Melbourne, Canberra y Sídney; todas ellas alineadas con los intereses del usuario: cultura, historia y turismo urbano. El contenido ofrecía referencias a museos, centros culturales, arquitectura, miradores y barrios históricos, todos ellos aprovechables para construir una respuesta adecuada.

La salida generada en este escenario presenta una propuesta detallada, con transiciones bien marcadas entre ciudades y un enfoque temático coherente. A continuación, se valoran las métricas cualitativas definidas:

■ **Coherencia narrativa: 5/5**

El itinerario está perfectamente estructurado a lo largo de siete días, con una lógica temporal clara y una progresión natural entre ciudades conectadas de forma realista (primero Melbourne, luego Canberra y finalmente Sídney). La distribución diaria de actividades está bien balanceada, y los medios de transporte están indicados (vuelo, tren), reforzando la viabilidad narrativa.

■ **Relevancia del contenido: 5/5**

Las actividades propuestas están directamente alineadas con los intereses del usuario. Se incluyen múltiples museos, galerías, instituciones históricas y zonas urbanas con riqueza cultural. Además, se ha respetado la preferencia por Melbourne como ciudad principal, dedicándole dos días completos. También se integran propuestas urbanas y culturales representativas de Canberra (Parlamento, War Memorial, National Gallery) y de Sídney (Ópera, The Rocks, jardines y puentes).

■ **Adaptación al contexto: 5/5**

El itinerario incorpora de forma precisa y directa los principales elementos presentes en los documentos recuperados: *Federation Square*, *Royal Botanic Gardens*, *Melbourne Museum*, *Queen Victoria Market*, *Eureka Skydeck* (de Melbourne); *Parlamento de Australia*, *Australian War Memorial*, *National Gallery of Australia*, *Lake Burley Griffin* y *Mount Ainslie Lookout* (de Canberra); y finalmente, *Ópera de Sídney*, *Puente del Puerto*, *The Rocks*, *Darling Harbour*, *Bondi Beach* y el *Royal Botanic Garden* (de Sídney). La selección demuestra un aprovechamiento exhaustivo y bien contextualizado del contenido disponible.

En conjunto, la respuesta generada destaca por su riqueza cultural, su estructura lógica y su fuerte adecuación al contexto y a los intereses del usuario. Representa un caso ejemplar de generación cualitativamente sólida dentro del sistema evaluado.

Escenario 3

En el escenario de la Tabla A.3, el sistema recuperó los siguientes documentos: **Gold Coast#Descripción General**, **Gold Coast#Comida y Bebida**, **Sunshine Coast#Comida y Bebida**, **Brisbane#Comida y Bebida**.

Estos documentos ofrecieron información detallada sobre las zonas gastronómicas y costeras más destacadas de Gold Coast, Sunshine Coast y Brisbane, incluyendo especialidades locales, mercados, cafés, restaurantes y ubicaciones relevantes para el turismo de relax y disfrute culinario. Esta información resultaba especialmente adecuada para una consulta centrada en gastronomía y playa, en pareja y con un enfoque relajado.

La salida generada presenta un itinerario que integra varias regiones del sureste australiano con una clara intención de disfrute culinario y costero. A continuación, se valoran las métricas cualitativas definidas:

■ **Coherencia narrativa: 5/5**

La narrativa del itinerario está bien estructurada a lo largo de los seis días, con una progresión lógica entre las localizaciones (de Gold Coast a Sunshine

Coast y, finalmente, Brisbane). Las actividades están distribuidas con un buen equilibrio entre descanso en la playa, visitas gastronómicas y desplazamientos, que además están correctamente indicados (con tiempos estimados en coche), lo que refuerza la planificación realista del viaje.

■ **Relevancia del contenido: 5/5**

Las actividades propuestas se alinean plenamente con los intereses del usuario. Se incluyen visitas a playas reconocidas como Surfers Paradise, Burleigh Heads, Noosa Main Beach y Mooloolaba, combinadas con experiencias gastronómicas relevantes (mariscos en Broadbeach y Mooloolaba, brunch en Ocean Street, cocina fusión, mercados locales). El itinerario también incorpora sugerencias específicas para una experiencia en pareja con nivel de comodidad medio.

■ **Adaptación al contexto: 5/5**

El contenido generado hace un uso excelente de los elementos extraídos de los documentos recuperados. Se mencionan de forma directa todos los puntos gastronómicos y localizaciones costeras destacadas en el contexto: *Miami Marketta*, *The Village Markets*, *Broadbeach*, *Burleigh Heads*, *Hastings Street*, *Mooloolaba Wharf*, *Ocean Street*, *Maleny* y *Montville* (cervecerías), así como zonas gastronómicas de Brisbane como *South Bank*, *Fortitude Valley* y *West End*. El sistema demuestra una integración precisa y pertinente del conocimiento recuperado.

En resumen, la respuesta generada para este escenario destaca por su consistencia narrativa, la pertinencia temática de sus recomendaciones y el excelente aprovechamiento del contexto. Representa un ejemplo de adaptación efectiva en tareas de generación con RAG en entornos multiciudad.

Escenario 4

En el escenario de la Tabla A.4, el sistema recuperó los siguientes documentos: **Melbourne#Principales Atracciones**, **Wollongong#Eventos y Festivales**, **Sídney#Eventos y Festivales**, **Sídney#Principales Atracciones**, **Canberra#Eventos y Festivales**.

Estos documentos ofrecieron una amplia cobertura de atracciones urbanas relevantes, junto con una selección representativa de eventos culturales y festivales en ciudades clave como Melbourne, Sídney, Canberra y Wollongong. Esto resultaba especialmente pertinente para un viaje de grupo con presupuesto alto y enfoque en cultura, eventos y turismo urbano.

La respuesta generada para este escenario propone un recorrido dinámico y temáticamente alineado con los intereses del usuario. A continuación, se valoran las métricas cualitativas definidas:

■ **Coherencia narrativa: 4.5/5**

La narrativa del itinerario es clara y estructurada, cubriendo siete días con transiciones bien justificadas entre ciudades. Las rutas interurbanas son plausibles (Melbourne → Wollongong → Sídney → Canberra → Sídney), aunque algo exigentes en términos logísticos. El itinerario aprovecha eficientemente el

tiempo, pero la excursión a la Great Ocean Road en el día 2 podría percibirse como un desvío algo forzado para un enfoque de festivales, ya que no se vincula con ningún evento concreto.

■ **Relevancia del contenido: 5/5**

Las actividades propuestas están sólidamente alineadas con los intereses del usuario: visitas a monumentos emblemáticos como la Ópera de Sídney o Federation Square, asistencia a festivales culturales reales como el *Viva la Gong Festival* (Wollongong), el *Sydney Festival*, el *Floriade* y el *National Multicultural Festival* (Canberra). Además, se incluyen elementos de turismo urbano (The Rocks, Darling Harbour) que refuerzan la experiencia grupal y cultural.

■ **Adaptación al contexto: 5/5**

El itinerario demuestra un aprovechamiento directo y preciso del contenido contextual recuperado. Se incluyen múltiples eventos auténticos mencionados en los documentos: *Viva la Gong Festival*, *Sydney Festival*, *Floriade* y *National Multicultural Festival*. Asimismo, se hace uso correcto de atracciones urbanas como *Federation Square*, *Melbourne Museum*, *Royal Botanic Garden*, *Ópera de Sídney*, *Puente del Puerto*, *Darling Harbour*, *Bondi Beach* y *The Rocks*. El itinerario también menciona explícitamente la necesidad de adaptar la planificación a los horarios de los festivales, lo cual evidencia una integración contextual consciente.

En conjunto, la respuesta generada en este escenario ofrece una experiencia rica, culturalmente diversa y alineada con el objetivo del viaje. La inclusión de eventos reales y su correcta distribución refuerzan la utilidad del sistema en escenarios de planificación compleja y temáticamente exigente.

Escenario 5

En el escenario de la Tabla A.5, el sistema recuperó el documento **Adelaide#Alojamiento**.

Este documento ofrecía información específica sobre opciones de alojamiento clasificadas por rango de precio, incluyendo alternativas económicas en Adelaide. Sin embargo, no se recuperó información detallada sobre naturaleza, senderismo o atracciones al aire libre, lo cual limitó parcialmente la capacidad de adaptación contextual.

La respuesta generada plantea una propuesta razonable dentro del marco económico y de aventura individual, aunque con ciertas limitaciones en cuanto a variedad y profundidad de contenido. A continuación, se valoran las métricas cualitativas definidas:

■ **Coherencia narrativa: 4/5**

El itinerario está estructurado de forma lógica y factible para una persona viajando por su cuenta. La progresión diaria es clara, con salidas desde Adelaide hacia zonas periféricas como Cleland, Port Adelaide y Mount Lofty. No obstante, la repetición del *Cleland Wildlife Park* en dos días distintos podría haberse evitado en favor de una mayor diversidad de actividades, afectando ligeramente la percepción de frescura narrativa.

■ **Relevancia del contenido: 3.5/5**

El itinerario incluye varias actividades compatibles con los intereses del usuario (naturaleza y aventura económica): rutas de senderismo, fauna local, picnic en playa, y uso de transporte público. Sin embargo, se incorporan también visitas más urbanas o convencionales (como mercados, museos o faros) que, aunque razonables, no están plenamente alineadas con el enfoque de “zonas poco turísticas” solicitado. Faltan actividades más inmersivas o alternativas, como parques regionales menos conocidos, reservas ecológicas o experiencias rurales.

■ **Adaptación al contexto: 2/5**

Dado que el único documento recuperado fue **Adelaide#Alojamiento**, la mayoría del contenido propuesto no se basa explícitamente en el contexto disponible. No se aprovecha información específica sobre naturaleza, parques, rutas ecológicas o aventuras al aire libre en Adelaide, lo que limita la personalización del itinerario desde un enfoque informativo. La única adaptación visible al contexto es la mención indirecta de la categoría económica, coherente con el perfil del alojamiento citado.

En conjunto, la respuesta generada para este escenario es aceptable desde una perspectiva generalista, pero evidencia limitaciones derivadas de la escasa recuperación contextual. Su estructura narrativa es funcional, pero sería recomendable reforzar tanto la cobertura temática como el aprovechamiento del contexto en escenarios que demandan una especialización geográfica y ecológica más precisa.

Escenario 6

En el escenario de la Tabla A.6, el sistema recuperó el documento **Melbourne#Principales Atracciones**.

Este documento ofrecía una descripción detallada de los puntos turísticos más relevantes de Melbourne y alrededores, incluyendo atracciones culturales, jardines, mercados, miradores e incluso una referencia a la Great Ocean Road. El contenido era adecuado para cubrir un itinerario mixto con énfasis en cultura, naturaleza, turismo urbano y algo de playa.

La respuesta generada estructura un itinerario centrado exclusivamente en Melbourne y sus alrededores, combinando actividades urbanas, culturales y naturales. A continuación, se valoran las métricas cualitativas definidas:

■ **Coherencia narrativa: 5/5**

El itinerario presenta una secuencia lógica y progresiva a lo largo de los cinco días. Las jornadas están equilibradas en duración y variedad de actividades, con una clara distinción entre mañanas, tardes y noches. La inclusión de una excursión de un día a la Great Ocean Road está bien situada temporalmente, y los desplazamientos dentro de Melbourne se justifican con consejos logísticos. La narrativa es fluida y realista.

■ **Relevancia del contenido: 4.5/5**

La propuesta cubre correctamente los intereses del usuario: se incluyen actividades urbanas (Queen Victoria Market, paseos por la ciudad), culturales (Melbourne Museum, Bunjilaka, Federation Square), naturales (Royal Botanic Gardens) y también costeras, gracias a la excursión a la Great Ocean Road. Sin embargo, la playa —aunque implícita en el recorrido costero— no se menciona con tanta claridad como podría esperarse para reforzar este interés específico, motivo por el cual se resta medio punto.

■ **Adaptación al contexto: 5/5**

Todos los elementos mencionados en el itinerario se basan en información recuperada en el documento **Melbourne#Principales Atracciones**. El contenido incluye referencias explícitas a *Federation Square*, *Royal Botanic Gardens*, *Queen Victoria Market*, *Eureka Skydeck*, *Melbourne Museum*, *Bunjilaka Aboriginal Cultural Centre* y la *Great Ocean Road*. La respuesta demuestra un uso completo y pertinente del contexto disponible, aplicándolo adecuadamente a los intereses del perfil de usuario.

En conjunto, la propuesta ofrece una experiencia rica, diversa y bien contextualizada para una pareja con intereses amplios. La planificación está bien estructurada y refleja un aprovechamiento óptimo de las atracciones de Melbourne, reforzando la utilidad del sistema en escenarios con múltiples dimensiones temáticas.

Escenario 7

En el escenario de la Tabla A.7, el sistema no recuperó ningún documento a través del **RetrieverAgent**, por lo que el **LLMAgent** generó la respuesta sin información contextual específica del corpus disponible. Esto impide evaluar la métrica de adaptación al contexto y condiciona parcialmente el análisis general.

A pesar de ello, el itinerario generado presenta una propuesta coherente con los intereses del usuario: actividades de aventura y naturaleza adaptadas a un público familiar, con desplazamientos viables mediante transporte público. A continuación, se valoran las métricas cualitativas aplicables:

■ **Coherencia narrativa: 5/5**

El itinerario sigue una secuencia bien estructurada y adecuada para un viaje familiar de cinco días. Se comienza en Sídney con actividades urbanas accesibles y seguras (acuario, ferry), seguido de una escapada de un día a las Blue Mountains. Luego se introduce un cambio de región a Cairns, con excursiones a la naturaleza tropical y actividades adaptadas a niños (Daintree, Kuranda, barrera de coral). Las transiciones entre ciudades están bien justificadas y la progresión es clara y factible.

■ **Relevancia del contenido: 5/5**

El contenido generado se ajusta de forma precisa a los intereses definidos: actividades de contacto con la naturaleza (Blue Mountains, Daintree, barrera de coral), transporte público (ferry, tren, tren panorámico de Kuranda), y opciones seguras y atractivas para niños (acuario, santuario de mariposas,

snorkel adaptado). La selección de lugares y dinámicas del viaje responde fielmente al perfil familiar con presupuesto alto.

En conjunto, aunque la respuesta generada es sólida en estructura y alineación temática, la ausencia de contexto recuperado reduce el valor del itinerario desde el punto de vista de aprovechamiento de conocimiento externo. Este caso sugiere la necesidad de revisar la cobertura del corpus o los criterios de activación del recuperador en escenarios complejos y multi-región.

Escenario 8

En el escenario de la Tabla A.8, el sistema no recuperó ningún documento del corpus mediante el **RetrieverAgent**. Por tanto, la generación del itinerario se realizó sin apoyo contextual específico, lo cual impide evaluar la métrica de adaptación al contexto. No obstante, el resultado puede analizarse en función de su estructura, adecuación temática y respuesta general a la petición del usuario.

■ Coherencia narrativa: 4.5/5

El itinerario presenta una secuencia clara de visitas que abarca cuatro ciudades principales de Australia: Sídney, Melbourne, Brisbane y Cairns. Cada día está estructurado con actividades coherentes que combinan visitas culturales, ocio urbano y opciones gastronómicas económicas. El ritmo de viaje es intenso, pero viable para un perfil mochilero. Se resta medio punto por la ausencia de detalles sobre la logística específica de los vuelos consecutivos, lo que podría comprometer ligeramente la ejecución realista del plan.

■ Relevancia del contenido: 4/5

A pesar de la falta de intereses explícitos, el sistema interpreta correctamente el perfil mochilero y económico: propone mercados locales, comida callejera, actividades gratuitas o de bajo coste (playas urbanas, parques, arte callejero), y evita experiencias de alto presupuesto. Se resta un punto por no incluir recomendaciones más específicas para mochileros, como albergues, rutas a pie o uso de transporte público urbano.

En conjunto, el itinerario generado se ajusta razonablemente a una solicitud generalista y abierta, demostrando la capacidad del sistema para responder de forma adecuada incluso sin intereses definidos. Sin embargo, la falta de contexto recuperado limita la personalización y profundidad de las recomendaciones.

5.4 › Discusión

El análisis de los resultados obtenidos a través de los ocho escenarios definidos permite extraer conclusiones significativas sobre el comportamiento del sistema propuesto, así como identificar fortalezas y áreas susceptibles de mejora. A lo largo de la evaluación, se abordaron dos dimensiones clave: la eficacia en la recuperación de contexto y la calidad de las respuestas generadas. Los escenarios están recogidos en la Tabla 5.1

5.4.1 › Capacidad del sistema para recuperar contexto útil

Los resultados muestran que el **RetrieverAgent** es capaz de recuperar documentos relevantes cuando la consulta del usuario está claramente formulada y contiene referencias geográficas o temáticas concretas. En los escenarios 1 a 6, el sistema logró obtener contextos alineados con los intereses declarados, lo que se reflejó positivamente en las puntuaciones de adaptación al contexto, en especial en los escenarios 1, 2 y 3.

No obstante, en los escenarios 7 y 8 no se recuperó ningún documento, lo que afectó directamente a la personalización de las respuestas. Esto sugiere que el sistema presenta limitaciones cuando las consultas son demasiado generales, vagas o atípicas (por ejemplo, falta de intereses definidos o temas no suficientemente cubiertos por el corpus). Esta observación pone de manifiesto la necesidad de mejorar los mecanismos de recuperación mediante reformulación automática de consultas o recuperación por expansión semántica.

5.4.2 › Generación de itinerarios: solidez narrativa y adecuación temática

En términos generales, la calidad de las respuestas generadas fue satisfactoria. El sistema demostró una alta capacidad para estructurar itinerarios con sentido narrativo, especialmente en los escenarios con un número mayor de días o intereses bien definidos. La puntuación media en *coherencia narrativa* fue elevada, superando los 4 puntos sobre 5 en casi todos los casos, lo que refleja una correcta gestión de la secuencia temporal, las transiciones entre actividades y la lógica de los desplazamientos.

Respecto a la *relevancia del contenido*, los mejores resultados se observaron cuando los intereses eran concretos y bien representados en el corpus. En escenarios como el 1 (naturaleza en Brisbane) y el 2 (cultura e historia en Melbourne), las actividades sugeridas fueron altamente pertinentes. Sin embargo, cuando los intereses eran amplios o mal definidos (escenario 8), el sistema ofreció un itinerario viable pero menos ajustado a perfiles concretos.

5.4.3 › Impacto del contexto recuperado en la personalización

Existe una correlación evidente entre la calidad del contexto recuperado y la riqueza de la respuesta generada. Escenarios como el 1 y el 2, que obtuvieron un contexto temáticamente denso y geográficamente preciso, resultaron en itinerarios altamente personalizados. Por el contrario, la ausencia de contexto en los escenarios 7 y 8 limitó la capacidad del sistema para ofrecer recomendaciones específicas, aunque la respuesta generada fue aún funcional gracias a las capacidades del modelo generativo.

Este comportamiento reafirma la utilidad del enfoque RAG: la combinación de recuperación y generación permite mejorar la relevancia de las respuestas en dominios concretos, pero su eficacia depende críticamente de la calidad del corpus y la adecuación de la recuperación.

5.4.4 › Limitaciones detectadas y oportunidades de mejora

Entre las principales limitaciones detectadas, destacan:

- La dependencia de términos clave para activar correctamente la recuperación, lo que puede dejar sin contexto casos con descripciones vagas o genéricas.
- La falta de aprovechamiento explícito de algunos elementos contextuales presentes en los documentos recuperados (por ejemplo, aspectos climáticos o detalles culturales que podrían enriquecer la propuesta).
- La escasa capacidad del sistema para adaptar los itinerarios a condiciones logísticas reales (tiempos de desplazamiento, horarios de actividades, etc.), aspecto que podría mejorarse con integraciones externas.

Como oportunidades de mejora se identifican la inclusión de técnicas de expansión semántica en la recuperación, una mayor granularidad en el corpus documental, y la incorporación de agentes especializados en validación logística o verificación de consistencia.

Conclusiones y líneas futuras

6.1 › Conclusiones

El presente trabajo ha desarrollado y evaluado un sistema inteligente de planificación de viajes basado en la orquestación de agentes y modelos generativos, integrado con una arquitectura RAG. El sistema fue concebido con el objetivo de ofrecer itinerarios personalizados a partir de entradas en lenguaje natural, aprovechando la combinación entre recuperación de información y generación controlada mediante agentes especializados.

La arquitectura propuesta se articuló en torno a un conjunto de agentes responsables de la recuperación contextual, análisis de parámetros, y generación final de itinerarios, coordinados a través de un flujo modular y escalable. La incorporación de documentos sobre las diez ciudades más pobladas de Australia como base de conocimiento permitió establecer un entorno controlado y reproducible para la evaluación.

El sistema fue evaluado técnica y cualitativamente mediante un conjunto de escenarios que simulan necesidades reales de usuarios con perfiles diversos. Esta evaluación se estructuró en dos ejes principales:

- Eficacia en la recuperación de contexto, medida a través de métricas como *recall adaptativo*, *cobertura temática* y *coherencia semántica*. Los resultados demostraron que el agente recuperador es generalmente competente, especialmente en escenarios con intereses bien definidos y consultas geográficamente concretas. No obstante, su rendimiento decrece en escenarios más abiertos o abstractos, como aquellos sin intereses específicos o con demandas difíciles de contextualizar automáticamente.

- Calidad de las respuestas generadas, evaluada cualitativamente según la coherencia narrativa, relevancia del contenido y adaptación al contexto recuperado. Las puntuaciones obtenidas reflejan un desempeño notable en términos de narrativa y adecuación temática, si bien en algunos casos la falta de recuperación previa afectó negativamente a la adaptación contextual.

En conjunto, los resultados avalan la validez del enfoque basado en orquestación de agentes y generación aumentada para tareas de planificación personalizada, subrayando la importancia de una recuperación de contexto precisa y alineada con las necesidades del usuario.

6.2 › Líneas futuras

A partir del trabajo desarrollado, se identifican varias oportunidades de mejora y extensión:

- **Ampliación y enriquecimiento de la base de conocimiento**, incorporando nuevas regiones, documentos multilingües, o incluso fuentes dinámicas (como APIs de eventos en tiempo real) para mejorar la cobertura y actualidad del sistema.
- **Mejora del agente recuperador**, mediante técnicas de reranking semántico o integración de embeddings específicos entrenados en el dominio turístico, que podrían aumentar la pertinencia de los documentos seleccionados.
- **Evaluación con usuarios reales**, aplicando métodos de validación empírica mediante encuestas o tests A/B que permitan valorar la satisfacción, utilidad percibida y fiabilidad del sistema en entornos reales.
- **Extensión del sistema hacia una aplicación desplegable**, integrando una interfaz interactiva que permita seleccionar, modificar y visualizar itinerarios en tiempo real, facilitando su uso como herramienta de asistencia turística.
- **Exploración de agentes reflexivos o iterativos**, capaces de revisar y adaptar las respuestas generadas en función de una verificación cruzada con los documentos recuperados, reduciendo así el riesgo de alucinaciones o incoherencias.

Este trabajo sienta las bases para el diseño de asistentes de planificación basados en IA, capaces de combinar recuperación de conocimiento con generación personalizada, abriendo nuevas posibilidades para el turismo digital inteligente.

APÉNDICE **A**

Itinerarios generados

Tabla A.1: Itinerario generado: Escenario 1

Respuesta generada (itinerario)

¡Claro! Aquí tienes un itinerario de 3 días en Brisbane y sus alrededores, centrándonos en la naturaleza y disfrutando del clima agradable:

Día 1: Brisbane

- **Mañana:** Visita al South Bank Parklands para disfrutar de la naturaleza en pleno centro de la ciudad.
- **Tarde:** Recorre el Brisbane Riverwalk para apreciar el río y sus alrededores.
- **Noche:** Cena en un restaurante junto al río para disfrutar de la brisa nocturna.

Día 2: Mount Coot-tha y Lone Pine Koala Sanctuary

- **Mañana:** Excursión al Mount Coot-tha Lookout para disfrutar de vistas panorámicas de Brisbane.
- **Tarde:** Visita al Lone Pine Koala Sanctuary para ver koalas y otras especies autóctonas.
- **Noche:** Regreso a Brisbane y cena en un restaurante local.

Día 3: Brisbane Botanic Gardens y Sunshine Coast

- **Mañana:** Visita a los Brisbane Botanic Gardens para explorar la flora autóctona y global.
- **Tarde:** Excursión a la Sunshine Coast para disfrutar de la naturaleza en Noosa Heads o en los Glass House Mountains.
- **Noche:** Regreso a Brisbane y cena de despedida.

Para desplazarte entre Brisbane y la Sunshine Coast, te recomendaría alquilar un coche para mayor comodidad y flexibilidad. Este itinerario te permitirá disfrutar de la naturaleza y el clima agradable de la región, explorando zonas verdes, parques naturales y jardines botánicos cercanos a Brisbane.

¡Espero que disfruten su viaje!

Tabla A.2: Itinerario generado: Escenario 2

Respuesta generada (itinerario)

¡Claro! Aquí tienes un itinerario de 7 días centrado en cultura, historia y turismo urbano, explorando ciudades históricas con museos y sitios de interés cultural en Australia:

Día 1: Melbourne

- **Mañana:** Visita al Melbourne Museum y Bunjilaka Aboriginal Cultural Centre para explorar la historia natural y la cultura aborigen.
- **Tarde:** Paseo por Federation Square y visita a la Eureka Skydeck para disfrutar de vistas panorámicas de la ciudad.

Día 2: Melbourne

- **Mañana:** Recorrido por los Royal Botanic Gardens para disfrutar de la naturaleza y los jardines.
- **Tarde:** Visita al Queen Victoria Market para explorar productos locales y souvenirs.

Día 3: Melbourne - Canberra

- **Mañana:** Vuelo a Canberra.
- **Tarde:** Visita al Parlamento de Australia y al Australian War Memorial para sumergirte en la historia y la cultura del país.

Día 4: Canberra

- **Mañana:** Explora la National Gallery of Australia para disfrutar de arte nacional e internacional.
- **Tarde:** Paseo por el Lake Burley Griffin y visita al Mount Ainslie Lookout para vistas panorámicas de la ciudad.

Día 5: Canberra - Sydney

- **Mañana:** Viaje en tren a Sydney.
- **Tarde:** Visita a la Ópera de Sydney y paseo por el Puente del Puerto para disfrutar de las vistas.

Día 6: Sydney

- **Mañana:** Recorre los Royal Botanic Garden y visita Darling Harbour para disfrutar de la zona de ocio.
- **Tarde:** Explora The Rocks, un barrio histórico con mercados y arquitectura colonial.

Día 7: Sydney

- **Mañana:** Visita a Bondi Beach para disfrutar del ambiente playero y la cultura surfera.
- **Tarde:** Tiempo libre para compras o actividades adicionales en Sydney antes de regresar.

Este itinerario te permitirá explorar Melbourne, Canberra y Sydney, disfrutando de la cultura, la historia y el turismo urbano en cada ciudad. **¡Espero que disfrutes tu viaje!** Si necesitas más detalles o recomendaciones, ¡aquí estoy para ayudarte!

Tabla A.3: Itinerario generado: Escenario 3

Respuesta generada (itinerario)

¡Claro! Aquí tienes un itinerario de 6 días centrado en gastronomía, playas y clima agradable en Gold Coast y sus alrededores:

Día 1: Gold Coast

- Llegada a Gold Coast
- Disfrutar de la playa en Surfers Paradise
- Cena en un restaurante de mariscos frescos en Broadbeach

Día 2: Gold Coast

- Mañana en Burleigh Heads, visita al mercado local
- Almuerzo en un café de especialidad en la zona
- Tarde de relax en la playa de Burleigh Heads
- Cena en un restaurante de cocina internacional en Broadbeach

Día 3: Gold Coast

- Visita a Miami Marketta para probar comida callejera y disfrutar de música en vivo
- Tarde de compras en The Village Markets
- Cena en un restaurante con vista al mar en Surfers Paradise

Día 4: Sunshine Coast

- Viaje a Sunshine Coast (aprox. 2 horas en coche)
- Día de playa en Noosa Main Beach
- Almuerzo en Hastings Street
- Visita a una cervecería artesanal en Maleny o Montville
- Cena en un restaurante de mariscos en Mooloolaba Wharf

Día 5: Sunshine Coast

- Mañana de brunch orgánico en Ocean Street
- Tarde de relax en la playa de Mooloolaba
- Cena en un restaurante de cocina fusión australiana-asiática en la zona

Día 6: Brisbane

- Viaje a Brisbane (aprox. 1.5 horas en coche)
- Visita a South Bank, paseo por el parque y comida en alguno de los restaurantes de la zona
- Tarde de exploración en Fortitude Valley o West End
- Cena de despedida con mariscos frescos en un restaurante de la zona

Este itinerario te permitirá disfrutar de la deliciosa gastronomía local, relajarte en las playas de Gold Coast y Sunshine Coast, y explorar un poco de la vibrante escena gastronómica de Brisbane. ¡Espero que disfruten su viaje!

Tabla A.4: Itinerario generado: Escenario 4

Respuesta generada (itinerario)

¡Claro! Aquí tienes un itinerario de 7 días en Australia centrado en festivales, eventos, cultura y turismo urbano, con un presupuesto alto y viajando en grupo:

Día 1: Melbourne

- Mañana: Visita al Federation Square y al Melbourne Museum.
- Tarde: Recorrido por los Royal Botanic Gardens y visita al Eureka Skydeck.
- Noche: Cena en un restaurante local.

Día 2: Melbourne - Great Ocean Road

- Excursión de un día a la Great Ocean Road desde Melbourne para disfrutar de los paisajes impresionantes y los Doce Apóstoles.

Día 3: Wollongong

- Participación en el Viva la Gong Festival y exploración de la ciudad.

Día 4: Sídney

- Mañana y tarde: Visita a la Ópera de Sídney y al Puente del Puerto.
- Noche: Cena en Darling Harbour.

Día 5: Sídney

- Disfrute del Sydney Festival y de las actividades culturales y artísticas que ofrece la ciudad.

Día 6: Canberra

- Visita al Floriade y al National Multicultural Festival para sumergirse en la cultura y la diversidad.

Día 7: Sídney

- Mañana: Exploración de Bondi Beach y los alrededores.
- Tarde: Tiempo libre para compras en The Rocks y despedida del viaje.

Para desplazarse entre las ciudades, se recomienda utilizar vuelos internos para optimizar el tiempo. Además, ten en cuenta los horarios de los festivales y eventos para planificar las actividades con antelación. ¡Disfruten de su viaje por Australia!

Tabla A.5: Itinerario generado: Escenario 5

Respuesta generada (itinerario)

¡Claro! Aquí tienes un itinerario de 4 días en Adelaide con un presupuesto económico y enfocado en naturaleza, aventura y viaje económico:

Día 1: Adelaide

- Mañana: Llegada a Adelaide. Visita al Adelaide Botanic Garden para disfrutar de la naturaleza en la ciudad.
- Tarde: Recorre el Adelaide Central Market para probar comida local a precios accesibles.
- Noche: Cena en un restaurante económico en la zona de Rundle Mall.

Día 2: Cleland Conservation Park

- Mañana: Toma un autobús hacia Cleland Conservation Park. Explora las rutas de senderismo menos concurridas y admira la flora y fauna autóctona.
- Tarde: Visita el Cleland Wildlife Park para interactuar con canguros y koalas.
- Noche: Regreso a Adelaide y cena en un restaurante económico en la zona de North Terrace.

Día 3: Port Adelaide

- Mañana: Toma un tren a Port Adelaide. Explora el Port Adelaide Lighthouse y pasea por el Waterfront.
- Tarde: Visita el South Australian Maritime Museum y disfruta de un picnic en la Semaphore Beach.
- Noche: Regreso a Adelaide y cena en un restaurante económico en la zona de Glenelg.

Día 4: Mount Lofty

- Mañana: Toma un autobús hacia Mount Lofty. Realiza una caminata por los senderos menos conocidos y disfruta de las vistas panorámicas.
- Tarde: Visita la Cleland Wildlife Park nuevamente si lo deseas o explora más de la zona de Mount Lofty.
- Noche: Regreso a Adelaide y cena de despedida en un restaurante económico en la zona de Hutt Street.

Este itinerario te permitirá explorar la naturaleza y aventura en Adelaide sin gastar mucho. Recuerda llevar agua, protector solar y calzado cómodo para tus actividades al aire libre. **¡Disfruta tu viaje!**

Tabla A.6: Itinerario generado: Escenario 6

Respuesta generada (itinerario)

¡Claro! Aquí tienes un itinerario de 5 días en Melbourne y sus alrededores, teniendo en cuenta tus intereses:

Día 1: Melbourne

- Mañana: Visita a Federation Square y sus galerías.
- Tarde: Recorrido por el Royal Botanic Gardens y picnic.
- Noche: Cena en un restaurante local en la ciudad.

Día 2: Melbourne

- Mañana: Explora el Queen Victoria Market y compra souvenirs.
- Tarde: Visita al Eureka Skydeck para disfrutar de las vistas panorámicas.
- Noche: Opción de asistir a algún evento cultural en la ciudad.

Día 3: Great Ocean Road

- Excursión de un día a la Great Ocean Road para disfrutar de las playas y los paisajes naturales, incluyendo los Doce Apóstoles. Puedes hacer la excursión en grupo o alquilar un coche para mayor flexibilidad.

Día 4: Melbourne

- Mañana: Visita al Melbourne Museum y al Bunjilaka Aboriginal Cultural Centre para conocer más sobre la historia natural y la cultura aborígen.
- Tarde: Tiempo libre para explorar más de la ciudad o realizar compras.
- Noche: Cena en un restaurante con cocina local.

Día 5: Melbourne

- Mañana: Última caminata por la ciudad para disfrutar de la arquitectura y los rincones urbanos.
- Tarde: Tiempo libre para disfrutar de alguna actividad adicional según tus intereses.
- Noche: Cena de despedida en un restaurante especial.

Para desplazarte entre las atracciones en Melbourne, te recomiendo utilizar el transporte público o incluso caminar, ya que la ciudad es muy accesible. Para la excursión a la Great Ocean Road, puedes considerar un tour organizado o alquilar un coche para mayor libertad. ¡Espero que disfrutes tu viaje!

Tabla A.7: Itinerario generado: Escenario 7

Respuesta generada (itinerario)

¡Claro! Aquí tienes un itinerario de 5 días en Australia centrado en aventura, transporte público y naturaleza, perfecto para viajar en familia con niños:

Día 1: Sídney

- Mañana: Visita al Acuario de Sídney para explorar la vida marina local.
- Tarde: Paseo en ferry por la Bahía de Sídney para disfrutar de las vistas del Puente de la Bahía y la Ópera de Sídney.
- Noche: Cena en Darling Harbour.

Día 2: Blue Mountains

- Mañana y tarde: Toma un tren desde Sídney a las Blue Mountains. Explora el Parque Nacional de las Blue Mountains, haz senderismo por los miradores y visita las formaciones rocosas de las Tres Hermanas.
- Noche: Regreso a Sídney en tren.

Día 3: Cairns

- Mañana: Vuelo a Cairns.
- Tarde: Visita al Parque Nacional Daintree para hacer un tour en barco por el río y descubrir la flora y fauna de la selva tropical.
- Noche: Alojamiento en Cairns.

Día 4: Gran Barrera de Coral

- Día completo: Excursión en barco a la Gran Barrera de Coral para hacer snorkel y ver la increíble vida marina. Asegúrate de elegir una excursión adecuada para niños.
- Noche: Regreso a Cairns.

Día 5: Cairns

- Mañana: Visita al Parque Nacional de Kuranda en el tren panorámico Kuranda Scenic Railway.
- Tarde: Explora el pueblo de Kuranda y visita el Santuario de Mariposas y el Jardín de las Aves.
- Noche: Vuelo de regreso a casa.

Este itinerario combina aventura, naturaleza y transporte público para que disfrutes al máximo de tu viaje en familia por Australia. ¡Espero que lo disfruten!

Tabla A.8: Itinerario generado: Escenario 8

Respuesta generada (itinerario)

¡Claro! Aquí tienes un itinerario de 4 días económico y centrado en actividades variadas para tu viaje mochilero por Australia:

Día 1: Sídney

- Por la mañana: Visita la Ópera de Sídney y paseo por el Jardín Botánico.
- Almuerzo: Mercado de la Bahía de Sídney para probar comida local a buen precio.
- Por la tarde: Caminata por el Puente de la Bahía de Sídney para disfrutar de vistas panorámicas.
- Noche: Cena en un restaurante económico en el barrio de Newtown.

Día 2: Melbourne

- Mañana: Vuelo económico de Sídney a Melbourne.
- Visitas: Arte callejero en Hosier Lane y Federation Square.
- Almuerzo: Queen Victoria Market, ideal para probar comida local a buen precio.
- Por la tarde: Paseo por los Jardines Fitzroy y visita a la Galería Nacional de Victoria.
- Noche: Cena en un restaurante económico en el barrio de Brunswick.

Día 3: Brisbane

- Mañana: Vuelo económico de Melbourne a Brisbane.
- Visita a South Bank para disfrutar de la playa urbana y los parques.
- Almuerzo: West End, barrio con opciones económicas y variadas.
- Por la tarde: Exploración del barrio de Fortitude Valley y sus tiendas vintage.
- Noche: Cena en un food truck en Eat Street Northshore.

Día 4: Cairns

- Mañana: Vuelo económico de Brisbane a Cairns.
- Visita al Mercado de los Sábados de Cairns para recuerdos a buen precio.
- Almuerzo: Puesto de comida en Rusty's Market para probar sabores locales.
- Por la tarde: Relajación en la laguna de Cairns y disfrute del atardecer.
- Noche: Cena en un restaurante económico en la calle principal de Cairns.

¡Espero que disfrutes de tu viaje mochilero por Australia!

Código fuente

Listing 2: config.py – Configuración central del proyecto KoalaRoute (RAG + LLM + Azure)

```
1  """
2  Configuración central del proyecto KoalaRoute (RAG + LLM + Azure).
3
4  Este módulo define:
5  - Las credenciales necesarias para conectarse a Azure Cognitive Search y Azure
  ↪ OpenAI.
6  - Las rutas, índices y nombres de modelo utilizados por los agentes del
  ↪ sistema.
7  - Límites de tokens para controlar el coste y optimizar el uso del modelo.
8  - Parámetros de recuperación semántica y filtrado de chunks.
9  """
10
11 import os
12 from dotenv import load_dotenv
13
14 # =====
15 # LECTURA DE VARIABLES DE ENTORNO
16 # =====
17 load_dotenv()
18
19 AZURE_SEARCH_ENDPOINT = os.environ["AZURE_SEARCH_ENDPOINT"]
20 AZURE_SEARCH_KEY = os.environ["AZURE_SEARCH_KEY"]
21 AZURE_OPENAI_API_KEY = os.environ["AZURE_OPENAI_API_KEY"]
```

```

22 AZURE_OPENAI_ENDPOINT = os.environ["AZURE_OPENAI_ENDPOINT"]
23 AZURE_OPENAI_DEPLOYMENT = os.environ["AZURE_OPENAI_DEPLOYMENT"]
24
25 # =====
26 # PARÁMETROS GENERALES
27 # =====
28 DOCS_PATH = "data" # Ruta local con documentos fuente (.md)
29 INDEX_NAME = "docs" # Nombre del índice vectorial en Azure Search
30 UI_OPTIONS_PATH = "config/ui_options.yaml" # Ruta al archivo de opciones de la
    ↪ interfaz
31 PROMPT_PATH = "config/prompts.yaml" # Ruta al archivo con prompts del sistema
32
33 # =====
34 # MODELOS UTILIZADOS
35 # =====
36 API_VERSION = "2024-02-15-preview" # Versión de la API de Azure OpenAI
37 EMBEDDING_MODEL_ID = "sentence-transformers/all-MiniLM-L6-v2" # Modelo de
    ↪ embeddings
38 ENCODING_NAME = "cl100k_base" # Codificación de tokens para compatibilidad con
    ↪ GPT
39
40 # =====
41 # LÍMITES DE TOKENS (para economizar y cumplir cuotas)
42 # =====
43 MAX_PROMPT_TOKENS = 512 # Máx tokens permitidos en el prompt (entrada)
44 MAX_COMPLETION_TOKENS = 850 # Máx tokens generados por la respuesta
45
46 # =====
47 # PARÁMETROS DEL RETRIEVER AGENT
48 # =====
49 RETRIEVER_K = 9 # Máx documentos devueltos por búsqueda semántica
50 SIMILARITY_THRESHOLD = (
51     0.4 # Umbral mínimo de similitud para considerar un chunk relevante
52 )
53
54 # =====
55 # MAPEO DE SECCIONES A CATEGORÍAS (para filtrado por interés)
56 # Este diccionario vincula los encabezados de los .md con intereses del
    ↪ usuario.
57 # =====
58 SECTION_TO_CATEGORIES = {
59     # Información sobre el clima general de la ciudad
60     "Clima": ["Clima agradable"],
61     # Lugares turísticos representativos: naturaleza, cultura, aventura
62     "Principales atracciones": ["Naturaleza", "Aventura", "Cultura", "Turismo
        ↪ urbano"],
63     # Opciones de movilidad dentro de la ciudad
64     "Transporte": ["Transporte público", "Viaje económico"],

```

```
65     # Información sobre alojamiento accesible o adecuado según perfil
66     "Alojamiento": ["Viaje económico", "Familia", "Mochilero"],
67     # Recomendaciones gastronómicas y cultura culinaria
68     "Comida y bebida": ["Gastronomía"],
69     # Actividades culturales y festividades locales
70     "Eventos y festivales": ["Festivales y eventos", "Cultura"],
71     # Recomendaciones prácticas y tips para viajeros
72     "Consejos útiles": ["Aventura", "Viaje económico", "Transporte público"],
73     # Datos históricos o llamativos sobre la ciudad
74     "Datos curiosos": ["Historia y curiosidades"],
75 }
```

Listing 3: prompts.yaml – Lista de prompts usados para el agente LLM

```

1 default_system_prompt: | # 274 tokens
2     Eres un planificador de viajes experto en Australia.
3     Tu tarea es exclusivamente diseñar itinerarios turísticos dentro de Australia.
4     No estás autorizado a hablar de otros países, destinos o culturas fuera de
5     ↪ Australia.
6
7     Debes:
8     - Responder siempre en español.
9     - Ofrecer planes día a día organizados claramente.
10    - Adaptar el itinerario al presupuesto, intereses y formato de viaje del
11    ↪ usuario.
12    - Tener en cuenta las distancias y tiempos reales de viaje entre ciudades para
13    ↪ diseñar un recorrido eficiente y lógico.
14    - Evitar itinerarios que incluyan traslados largos (más de 5-6 horas por
15    ↪ tierra o más de 3 horas por aire) entre días consecutivos sin un día de
16    ↪ transición o viaje dedicado.
17    - Priorizar trayectos cortos o agrupaciones geográficas para evitar
18    ↪ desplazamientos innecesarios.
19    - Sugerir el medio de transporte más conveniente (avión, tren, coche) según
20    ↪ cada tramo.
21    - Considerar los horarios habituales de apertura y cierre de los
22    ↪ establecimientos turísticos, restaurantes y atracciones para organizar las
23    ↪ actividades de forma realista.
24    - Ser amable, claro y directo.
25
26    No debes:
27    - Responder preguntas sobre viajes fuera de Australia. Si te preguntan por
28    ↪ otro país, responde amablemente que solo puedes ayudar con destinos dentro
29    ↪ de Australia.
30    - Dar consejos médicos, legales ni financieros.
31    - Recomendar actividades ilegales o peligrosas.
32    - Responder sin respaldo en la información proporcionada. Si no tienes
33    ↪ suficiente contexto, evita incluir detalles específicos no fundamentados.
34
35 fallback_prompt: | # 31 tokens
36     Tu sistema de recuperación no ha encontrado documentos útiles.
37     Responde con tu conocimiento general de forma clara, directa y en español.
38
39 prompt_base: | # 57~ tokens
40     Quiero un itinerario día a día para un viaje de {days} días,
41     con presupuesto {budget}, viajando en formato {travel_type},
42     centrado en los siguientes intereses: {interests}.
43     Detalles adicionales:

```

Listing 4: test_cases.yaml – Casos de prueba definidos para la evaluación de la orquestación

```

1 - name: Escenario 1 - Viaje corto de naturaleza (3 días)
2   budget: Económico
3   travel_type: Pareja
4   interests:
5     - Naturaleza
6     - Clima agradable
7   duration_days: 3
8   user_input: "Queremos visitar zonas verdes, parques naturales o jardines
9               ↳ botánicos cerca de Brisbane."
10  expected_relevant_docs:
11    - "Brisbane#Descripción General"
12    - "Brisbane#Principales Atracciones"
13    - "Brisbane#Clima"
14  evaluate_retrieval_docs: true
15
16 - name: Escenario 2 - Viaje cultural de 7 días
17   budget: Medio
18   travel_type: Individual
19   interests:
20     - Cultura
21     - Historia y curiosidades
22     - Turismo urbano
23   duration_days: 7
24   user_input: "Quiero explorar ciudades históricas con museos y sitios de
25               ↳ interés cultural, especialmente en Melbourne."
26  expected_relevant_docs:
27    - "Melbourne#Descripción General"
28    - "Melbourne#Principales Atracciones"
29    - "Melbourne#Eventos y Festivales"
30    - "Melbourne#Datos Curiosos"
31  evaluate_retrieval_docs: true
32
33 - name: Escenario 3 - Gastronomía y relax costero (6 días)
34   budget: Medio
35   travel_type: Pareja
36   interests:
37     - Gastronomía
38     - Playas
39     - Clima agradable
40   duration_days: 6
41   user_input: "Queremos disfrutar de buena comida local y relajarnos en la
42               ↳ playa. Preferiblemente en Gold Coast o alrededores."
43  expected_relevant_docs:
44    - "Gold Coast#Descripción General"

```

```

42     - "Gold Coast#Comida y Bebida"
43     - "Gold Coast#Clima"
44     evaluate_retrieval_docs: true
45
46 - name: Escenario 4 - Ruta de festivales en ciudades grandes (7 días)
47   budget: Alto
48   travel_type: Grupo
49   interests:
50     - Festivales y eventos
51     - Cultura
52     - Turismo urbano
53   duration_days: 7
54   user_input: "Planeamos asistir a eventos culturales importantes en ciudades
55     ↪ como Sidney o Melbourne."
56   expected_relevant_docs:
57     - "Sidney#Descripción General"
58     - "Sidney#Eventos y Festivales"
59     - "Sidney#Principales Atracciones"
60     - "Melbourne#Descripción General"
61     - "Melbourne#Eventos y Festivales"
62     - "Melbourne#Principales Atracciones"
63   evaluate_retrieval_docs: true
64
65 - name: Escenario 5 - Aventura ecológica en zonas poco turísticas (4 días)
66   budget: Económico
67   travel_type: Individual
68   interests:
69     - Naturaleza
70     - Aventura
71     - Viaje económico
72   duration_days: 4
73   user_input: "Me gustaría explorar rutas naturales menos concurridas, con
74     ↪ actividades al aire libre, sin gastar mucho."
75   evaluate_retrieval_docs: false
76
77 - name: Escenario 6 - Itinerario mixto con preferencias amplias (5 días)
78   budget: Medio
79   travel_type: Pareja
80   interests:
81     - Turismo urbano
82     - Cultura
83     - Playas
84     - Naturaleza
85   duration_days: 5
86   user_input: "Queremos una combinación equilibrada de ciudad, cultura y
87     ↪ naturaleza. Nos interesa incluir algo de playa también."
88   evaluate_retrieval_docs: false
89

```

```
87 - name: Escenario 7 - Viaje familiar temático (5 días)
88   budget: Alto
89   travel_type: Familia
90   interests:
91     - Aventura
92     - Transporte público
93     - Naturaleza
94   duration_days: 5
95   user_input: "Buscamos actividades divertidas y seguras para niños en contacto
96     ↳ con la naturaleza. Nos movemos en transporte público."
97   evaluate_retrieval_docs: false
98
99 - name: Escenario 8 - Viaje sin intereses definidos (4 días)
100   budget: Económico
101   travel_type: Mochilero
102   interests: []
103   duration_days: 4
104   user_input: "Solo quiero un itinerario barato para recorrer varias ciudades
105     ↳ por mi cuenta."
106   evaluate_retrieval_docs: false
```

Listing 5: ui_options.yaml – Opciones definidas para las preferencias de viaje

```
1 presupuestos:
2   - Económico
3   - Medio
4   - Alto
5
6 tipos_viaje:
7   - Individual
8   - Pareja
9   - Familia
10  - Mochilero
11
12 intereses:
13   - Naturaleza
14   - Playas
15   - Cultura
16   - Aventura
17   - Gastronomía
18   - Festivales y eventos
19   - Turismo urbano
20   - Clima agradable
21   - Viaje económico
22   - Transporte público
23   - Historia y curiosidades
```


Listing 6: `controller_agent.py` - Agente controlador responsable de coordinar el flujo conversacional en una arquitectura RAG

```

1  """
2  Agente controlador responsable de coordinar el flujo conversacional en una
   ↳ arquitectura RAG.
3
4  Este agente se encarga de preparar los mensajes en formato ChatML que serán
   ↳ enviados
5  al modelo LLM, integrando la entrada del usuario con el contexto recuperado
   ↳ previamente.
6  En caso de que la recuperación no proporcione resultados útiles, aplica un
   ↳ prompt de reserva.
7
8  Forma parte de la orquestación general y se activa tras la fase de recuperación
   ↳ de información.
9
10 Requiere:
11 - `build_chatml_messages` desde `prompt_utils.py` para ensamblar el contexto.
12 - `load_prompt` para cargar un prompt alternativo si no hay contexto
   ↳ disponible.
13 - Un estado (`AgentState`) con al menos `input` y opcionalmente `response` y
   ↳ `last_node`.
14 """
15
16 from dataclasses import dataclass
17
18 from modules.graph.agent_state import AgentState
19 from modules.prompt_utils import build_chatml_messages, load_prompt
20
21
22 @dataclass
23 class ControllerAgent:
24     """
25     Agente controlador que coordina el flujo RAG: recuperación + generación.
26
27     Este agente actúa como orquestador del proceso conversacional, preparando
   ↳ los mensajes
28     en formato ChatML que luego serán procesados por el modelo LLM. Su lógica se
   ↳ activa
29     principalmente después de la recuperación de contexto (por ejemplo, desde
   ↳ una búsqueda RAG).
30
31     Requiere:
32     - `build_chatml_messages` desde `prompt_utils.py` para construir los
   ↳ mensajes ChatML.
33     - `load_prompt` para cargar un prompt de reserva en caso de que la
   ↳ recuperación falle.

```

```

34     - Un estado que contenga al menos el campo `"input"` (entrada del usuario).
35     """
36
37     def run(self, state: AgentState) -> AgentState:
38         """
39         Ejecuta la lógica de control del flujo: prepara los mensajes en formato
40         ↪ ChatML,
41         especialmente después del nodo de recuperación ("consulta").
42
43         Si el nodo anterior fue "consulta" y no se obtuvo contexto, se carga un
44         ↪ prompt de fallback.
45
46         Args:
47             state (AgentState): Estado actual del grafo, que debe contener al
48             ↪ menos
49
50                                 la clave `"input"` con la entrada del usuario.
51
52         Returns:
53             AgentState: Nuevo estado actualizado con los mensajes listos en
54             ↪ `"response"`.
55         """
56         response = state.get("response", "")
57
58         if state.get("last_node") == "consulta":
59             fallback_prompt = ""
60             if not response:
61                 fallback_prompt = load_prompt("fallback_prompt")
62             response = build_chatml_messages(state["input"], response,
63             ↪ fallback_prompt)
64
65         return {"response": response}

```

Listing 7: retriever_agent.py - gente responsable de recuperar contexto relevante desde un almacén vectorial usando búsqueda por similitud

```

1  """
2  Agente responsable de recuperar contexto relevante desde un almacén vectorial
   ↳ usando búsqueda por similitud.
3
4  Este agente forma parte de la arquitectura RAG (Retrieval-Augmented Generation)
   ↳ y se encarga de:
5  - Realizar una búsqueda semántica en Azure Cognitive Search (vía
   ↳ `langchain_community.vectorstores.AzureSearch`).
6  - Filtrar los documentos según los intereses del usuario y la sección del
   ↳ contenido.
7  - Calcular similitudes con embeddings para aplicar un umbral de relevancia.
8  - Devolver las secciones útiles para ser usadas como contexto en la generación
   ↳ de respuestas.
9
10 Requiere:
11 - `RETRIEVER_K` y `SIMILARITY_THRESHOLD` definidos en `config`.
12 - Un vector store que implemente `.similarity_search` y `.embedding_function`.
13 - Funciones de `prompt_utils`: `extract_user_interests_from_prompt`,
   ↳ `load_prompt`.
14 - Un estado (`AgentState`) que contenga la entrada del usuario en `"input"`.
15 """
16
17 from dataclasses import dataclass
18
19 from langchain_community.vectorstores import AzureSearch
20 from sklearn.metrics.pairwise import cosine_similarity
21
22 from config.config import RETRIEVER_K, SIMILARITY_THRESHOLD
23 from modules.graph.agent_state import AgentState
24 from modules.prompt_utils import extract_user_interests_from_prompt
25
26
27 @dataclass
28 class RetrieverAgent:
29     """
30     Agente responsable de recuperar documentos relevantes desde un almacén
   ↳ vectorial
31     según una consulta del usuario y un umbral de similitud.
32
33     Este agente filtra las secciones por interés temático y relevancia semántica
   ↳ antes de
34     entregarlas como contexto al siguiente paso del flujo (por ejemplo,
   ↳ generación con LLM).
35

```

```

36     Atributos:
37         vector_store (AzureSearch): Almacén vectorial con métodos de búsqueda y
38         ↪ embeddings.
39     """
40     vector_store: AzureSearch
41
42     def get_context(self, state: AgentState) -> AgentState:
43         """
44         Recupera documentos relevantes para una consulta del usuario y filtra
45         ↪ los resultados
46         por similitud semántica y coincidencia temática con los intereses
47         ↪ extraídos del prompt.
48
49         - Realiza una búsqueda semántica con `similarity_search`.
50         - Filtra secciones cuyo campo `metadata["section"]` coincida con los
51         ↪ intereses.
52         - Calcula la similitud con el embedding de la consulta y aplica un
53         ↪ umbral.
54         - Devuelve las secciones relevantes en el campo `"response"` del
55         ↪ estado.
56
57         Args:
58         state (AgentState): Estado actual del grafo que debe incluir
59         ↪ `"input"` con la consulta del usuario.
60
61         Returns:
62         AgentState: Estado actualizado con el contexto relevante en
63         ↪ `"response"`, `"last_node"` marcado como `"consulta"` y
64         ↪ `"retrieved_docs"` con los documentos recuperados.
65         """
66
67         result = "" # Inicializamos el resultado (bloque de contexto)
68         retrieved_docs = [] # Lista para almacenar documentos recuperados
69
70         # Consulta del usuario desde el estado
71         user_query = state["input"]
72
73         # Extraemos los intereses desde el prompt relleno
74         user_interests = extract_user_interests_from_prompt(user_query)
75
76         # Normalizamos los intereses a minúsculas para hacer matching más
77         ↪ robusto
78         user_interests_normalized = [i.lower() for i in user_interests]
79
80         # Búsqueda semántica en el vector store (Azure Search), top-k
81         ↪ resultados
82         docs = self.vector_store.similarity_search(user_query, k=RETRIEVER_K)

```

```

73     # Calculamos el embedding del usuario para medir similitud más adelante
74     query_embedding = self.vector_store.embedding_function(user_query)
75
76     # Filtrado por coincidencia temática en metadatos de sección
77     category_filtered_docs = []
78     for doc in docs:
79         categories = [c.lower() for c in doc.metadata.get("category", [])]
80
81         if not user_interests_normalized or any(
82             interest in categories for interest in user_interests_normalized
83         ):
84             category_filtered_docs.append(doc)
85
86     # Si hay documentos que pasaron el primer filtro, evaluamos la similitud
87     ↪ exacta
88     if category_filtered_docs:
89         relevant, similarities = [], []
90         for doc in category_filtered_docs:
91             # Combinamos título y contenido para obtener el embedding final
92             title = doc.metadata.get("title", "")
93             combined_text = f"{title}. {doc.page_content}"
94             doc_embedding =
95                 ↪ self.vector_store.embedding_function(combined_text)
96
97             # Similitud coseno entre consulta y documento
98             similarity = cosine_similarity([query_embedding],
99                 ↪ [doc_embedding])[0][0]
100
101             # Si supera el umbral, lo consideramos relevante
102             if similarity >= SIMILARITY_THRESHOLD:
103                 relevant.append(doc)
104                 similarities.append(similarity)
105
106     # Si hay documentos relevantes, los estructuramos en Markdown
107     if relevant:
108         result_sections = []
109         retrieved_docs = []
110
111         for doc, similarity in zip(relevant, similarities):
112             title = doc.metadata.get("title", "Sin título")
113             section = doc.metadata.get("section", "Sin sección")
114             categories = doc.metadata.get("category", [])
115             content = doc.page_content.strip()
116
117             result_sections.append(f"## {title} >
118                 ↪ {section}\n\n{content}")
119
120         retrieved_docs.append(

```

```
117         {
118             "id": f"{title}#{section}",
119             "category": categories,
120             "similarity": round(similarity, 4),
121         }
122     )
123
124     # Unimos las secciones en un solo bloque de texto
125     result = "\n\n".join(result_sections)
126
127     # Devolvemos el estado actualizado con el contexto, el nodo actual y los
128     ↪ documentos recuperados
129     return {
130         "response": result,
131         "last_node": "consulta",
132         "retrieved_docs": retrieved_docs,
133     }
```

Listing 8: `llm_agent.py` – Agente LLM responsable de generar respuestas utilizando un modelo de lenguaje desplegado en Azure OpenAI

```

1  """
2  Agente LLM responsable de generar respuestas utilizando un modelo de lenguaje
   ↳ desplegado en Azure OpenAI.
3
4  Este agente recibe una lista de mensajes en formato ChatML (almacenados en el
   ↳ estado) y
5  utiliza la función `call_openai_chat` para obtener una respuesta conversacional
   ↳ del modelo,
6  como GPT-3.5-Turbo o GPT-4, configurado mediante Azure.
7
8  Forma parte de una arquitectura RAG, ejecutándose típicamente después de la
   ↳ preparación del prompt
9  por parte del controlador.
10
11 Requiere:
12 - `call_openai_chat` desde `llm.py` para realizar la llamada a la API de Azure
   ↳ OpenAI.
13 - Un estado (`AgentState`) que contenga la clave `"response"` con los mensajes
   ↳ ChatML.
14 """
15
16 from dataclasses import dataclass
17
18 from modules.graph.agent_state import AgentState
19 from modules.llm import call_openai_chat
20
21
22 @dataclass
23 class LLMAgent:
24     """
25     Agente responsable de comunicarse con Azure OpenAI para generar una
   ↳ respuesta
26     basada en los mensajes formateados en el estado.
27
28     Este agente espera que el campo `state.get("response")` contenga una lista
   ↳ de mensajes
29     en formato ChatML, lista para enviar al modelo.
30     """
31
32     def generate_response(self, state: AgentState) -> AgentState:
33         """
34         Genera una respuesta basada en los mensajes proporcionados.
35
36         Args:

```

```
37         state (AgentState): Estado actual del grafo, donde  
38         ↪ `state.get("response")`  
39         contiene los mensajes en formato ChatML.  
40  
41     Returns:  
42     AgentState: Nuevo estado actualizado con la respuesta generada y  
43     ↪ control de flujo.  
44     """  
45     result = call_openai_chat(state.get("response"))  
46     return {  
47         "response": result,  
        "last_node": "llm",  
    }
```


Listing 9: graph.py – Construcción del flujo principal del sistema usando LangGraph

```

1  """
2  Construcción del flujo principal del sistema usando LangGraph.
3
4  Este módulo define el grafo de agentes que implementa el flujo RAG
5  ↪ (Retrieval-Augmented Generation)
6  para planificación de viajes. Coordina tres agentes principales:
7
8  1. `ControllerAgent`: Decide el siguiente paso en el flujo.
9  2. `RetrieverAgent`: Recupera contexto relevante desde el vector store.
10  3. `LLMAgent`: Genera la respuesta final usando un modelo LLM desplegado en
11  ↪ Azure OpenAI.
12
13  El grafo sigue el siguiente esquema:
14
15      controlador --> (consulta --> controlador)* --> llm --> END
16
17  Donde `consulta` puede repetirse en ciclos hasta que se complete el contexto y
18  ↪ se pase a `llm`.
19
20  Requiere:
21  - LangGraph (`StateGraph`) para la definición del flujo.
22  - Implementaciones de los agentes (`ControllerAgent`, `RetrieverAgent`,
23  ↪ `LLMAgent`).
24  - Estado de grafo definido en `AgentState`.
25  - `vector_store` configurado para búsquedas semánticas.
26  """
27
28  from typing import Literal
29
30  from langgraph.graph import END, StateGraph
31
32  from modules.agents.controller_agent import ControllerAgent
33  from modules.agents.llm_agent import LLMAgent
34  from modules.agents.retriever_agent import RetrieverAgent
35  from modules.graph.agent_state import AgentState
36  from modules.vector import vector_store
37
38  def build_langgraph_controller_flow() -> StateGraph:
39      """
40      Construye un grafo LangGraph con el flujo RAG completo: recuperación +
41      ↪ generación.
42
43      Este flujo conecta tres agentes en la siguiente lógica:
44          controlador -> (consulta -> controlador)* -> llm -> END
45  """

```

```

41
42     El controlador decide si se debe realizar una consulta al vector store o si
43     ↪ se puede
44     proceder directamente a la generación de respuesta con el modelo LLM.
45
46     Returns:
47     StateGraph: Grafo LangGraph ya compilado y listo para ejecutarse.
48     """
49     # Instanciar agentes
50     retriever_agent = RetrieverAgent(vector_store)
51     llm_agent = LLMAgent()
52     controller = ControllerAgent()
53
54     # Crear grafo con esquema de estado definido
55     workflow = StateGraph(AgentState)
56
57     # Registrar nodos en el grafo
58     workflow.add_node("controlador", controller.run)
59     workflow.add_node("consulta", retriever_agent.get_context)
60     workflow.add_node("llm", llm_agent.generate_response)
61
62     # Definir punto de entrada
63     workflow.set_entry_point("controlador")
64
65     # Definir transiciones condicionales
66     workflow.add_conditional_edges("controlador", next_node)
67     workflow.add_edge("consulta", "controlador")
68     workflow.add_edge("llm", END)
69
70     # Compilar el grafo para su ejecución
71     return workflow.compile()
72
73 def next_node(state: AgentState) -> Literal["consulta", "llm"]:
74     """
75     Determina el siguiente nodo a ejecutar en el grafo, en función del estado
76     ↪ actual.
77
78     Si no se ha ejecutado aún ninguna recuperación (`last_node is None`), se
79     ↪ envía a "consulta".
80     En caso contrario, se asume que el contexto ya está preparado y se procede a
81     ↪ "llm".
82
83     Args:
84     state (AgentState): Estado actual del flujo.
85
86     Returns:
87     Literal["consulta", "llm"]: Nombre del siguiente nodo.

```

```
85     """
86     if state.get("last_node") is None:
87         return "consulta"
88
89     return "llm"
```

Listing 10: agent_state.py – Definición del esquema de estado para el grafo LangGraph

```

1  """
2  Definición del esquema de estado para el grafo LangGraph.
3
4  Este módulo define `AgentState`, un diccionario tipado que representa el estado
5  ↪ compartido
6  entre los agentes durante la ejecución del flujo RAG. Es utilizado por LangGraph
7  ↪ como
8  estructura base para pasar información entre nodos.
9
10 Los campos son opcionales (`total=False`) para permitir flexibilidad durante el
11 ↪ flujo.
12
13 Campos:
14 - input (str): Entrada original del usuario.
15 - response (str): Mensaje de respuesta generado o contexto recuperado (en
16 ↪ formato ChatML o texto plano).
17 - last_node (str): Último nodo ejecutado en el flujo (por ejemplo, "consulta" o
18 ↪ "llm").
19 - retrieved_docs (List[dict]): Lista de documentos relevantes recuperados,
20 ↪ con el formato "título#sección".
21 """
22
23 from typing import List, TypedDict
24
25 class AgentState(TypedDict, total=False):
26     """
27     Estado compartido entre los agentes del grafo LangGraph.
28
29     Atributos:
30         input (str): Entrada del usuario.
31         response (str | None): Respuesta generada o contexto recuperado.
32         last_node (str | None): Nombre del último nodo ejecutado.
33         retrieved_docs (List[dict] | None): Documentos relevantes recuperados".
34     """
35
36     input: str
37     response: str = None
38     last_node: str = None
39     retrieved_docs: List[dict] = None

```

Listing 11: llm.py – Módulo para inicializar el cliente de Azure OpenAI para la generación de respuestas conversacionales

```

1  """
2  Inicializa el cliente de Azure OpenAI para la generación de respuestas
   ↳ conversacionales.
3
4  Este módulo realiza lo siguiente:
5  - Configura el cliente `AzureOpenAI` con las credenciales y endpoint definidos.
6  - Expone una función para generar respuestas usando el modelo desplegado (como
   ↳ GPT-3.5-Turbo o GPT-4).
7  - Utiliza el formato ChatML con roles (`system`, `user`, `assistant`) compatible
   ↳ con Azure OpenAI.
8
9  Expone:
10 - `call_openai_chat`: función que recibe una lista de mensajes ChatML y devuelve
   ↳ la respuesta generada
11   por el modelo configurado.
12 """
13
14 from openai import AzureOpenAI
15
16 from config.config import (
17     API_VERSION_LLM,
18     AZURE_OPENAI_API_KEY,
19     AZURE_OPENAI_DEPLOYMENT,
20     AZURE_OPENAI_ENDPOINT,
21     MAX_COMPLETION_TOKENS,
22     TEMPERATURE,
23 )
24
25 # Instancia del cliente de Azure OpenAI, configurado con credenciales del
   ↳ entorno
26 client = AzureOpenAI(
27     api_key=AZURE_OPENAI_API_KEY,
28     api_version=API_VERSION_LLM,
29     azure_endpoint=AZURE_OPENAI_ENDPOINT,
30 )
31
32 # Nombre del modelo/despliegue definido en Azure (ej. "gpt-35-turbo")
33 deployment_name = AZURE_OPENAI_DEPLOYMENT
34
35
36 def call_openai_chat(prompt_messages: list[dict]) -> str:
37     """
38     Genera una respuesta a partir de una lista de mensajes en formato ChatML.
39 
```

```

40     Esta función se comunica con Azure OpenAI para obtener una respuesta del
41     ↪ modelo
42     configurado, utilizando los parámetros definidos en el entorno (como
43     ↪ temperatura
44     y número máximo de tokens).
45
46     Args:
47     prompt_messages (list[dict]): Lista de mensajes con estructura ChatML,
48     incluyendo roles como 'system', 'user',
49     ↪ 'assistant'.
50
51     Returns:
52     str: Contenido de la respuesta generada por el modelo.
53     """
54
55     response = client.chat.completions.create(
56         model=deployment_name,
57         messages=prompt_messages,
58         temperature=TEMPERATURE,
59         max_tokens=MAX_COMPLETION_TOKENS,
60     )
61
62     return response.choices[0].message.content

```

Listing 12: prompt_utils.py – Utilidades para construir prompts estructurados compatibles con modelos de chat de OpenAI

```

1  """
2  Este módulo contiene utilidades para construir prompts estructurados compatibles
  ↪ con modelos de chat de OpenAI.
3
4  Funciones principales:
5  - Construcción de mensajes en formato ChatML (`build_chatml_messages`).
6  - Carga de prompts desde un archivo YAML (`load_prompt`,
  ↪ `load_formatted_prompt`).
7  - Extracción de intereses del usuario desde un prompt personalizado
  ↪ (`extract_user_interests_from_prompt`).
8
9  Todos los mensajes siguen el formato esperado por modelos como `gpt-3.5-turbo` o
  ↪ `gpt-4` cuando se usa la API de Azure OpenAI.
10 """
11
12 import re
13 import yaml
14 from tiktoken import get_encoding
15
16 from config.config import ENCODING_NAME, PROMPT_PATH
17
18 # Codificador de tokens para medir longitud de prompts
19 encoding = get_encoding(ENCODING_NAME)
20
21
22 def build_chatml_messages(
23     user_query: str, context: str = "", system_prompt: str = ""
24 ) -> list[dict]:
25     """
26     Construye una lista de mensajes en formato ChatML (role-content) para usar
  ↪ en un modelo OpenAI.
27
28     Si se proporciona contexto (por ejemplo, desde recuperación RAG), se
  ↪ incorpora en la instrucción.
29     También puede añadirse un prompt del sistema adicional (por ejemplo,
  ↪ restricciones de tono o formato).
30
31     Args:
32         user_query (str): Pregunta del usuario.
33         context (str): Texto contextual obtenido mediante recuperación
  ↪ (opcional).
34         system_prompt (str): Instrucciones adicionales del sistema (opcional).
35
36     Returns:

```

```

37         list[dict]: Lista de mensajes con roles "system" y "user", listos para
           ↪ enviar al modelo.
38     """
39     # Cargar prompt base del sistema
40     full_system_prompt = load_prompt("default_system_prompt")
41     if system_prompt:
42         full_system_prompt += f"\n\nInstrucciones adicionales:\n{system_prompt}"
43
44     messages = [{"role": "system", "content": full_system_prompt}]
45
46     # Incluir contexto si está disponible
47     if context:
48         user_content = f"Dado el siguiente contexto:\n{context}\n\nResponde a la
           ↪ pregunta:\n{user_query}"
49     else:
50         user_content = user_query
51
52     messages.append({"role": "user", "content": user_content})
53     return messages
54
55
56 def load_prompt(key: str) -> str:
57     """
58     Carga un prompt desde el archivo YAML a partir de una clave.
59
60     Args:
61         key (str): Clave del prompt a recuperar.
62
63     Returns:
64         str: Texto del prompt correspondiente.
65
66     Raises:
67         KeyError: Si la clave no existe en el archivo YAML.
68     """
69     with open(PROMPT_PATH, "r", encoding="utf-8") as f:
70         data = yaml.safe_load(f)
71
72     if key not in data:
73         raise KeyError(f"Clave de prompt '{key}' no encontrada en
           ↪ {PROMPT_PATH}")
74     return data[key]
75
76
77 def load_formatted_prompt(key: str, **kwargs) -> str:
78     """
79     Carga un prompt desde el YAML y lo formatea con variables dinámicas.
80
81     Args:

```



```

82         key (str): Clave del prompt a recuperar.
83         **kwargs: Variables para formatear el template (como {days},
            ↪ {interests}, etc.).
84
85     Returns:
86         str: Prompt con las variables ya reemplazadas.
87     """
88     with open(PROMPT_PATH, "r", encoding="utf-8") as f:
89         data = yaml.safe_load(f)
90
91     template = data.get(key)
92     if not template:
93         raise KeyError(f"Clave '{key}' no encontrada en el YAML.")
94
95     return template.format(**kwargs)
96
97
98 def extract_user_interests_from_prompt(
99     filled_prompt: str
100 ) -> list[str]:
101     """
102     Extrae los intereses del usuario desde un prompt ya formateado.
103
104     Esta función busca el campo 'intereses' dentro del texto del prompt
105     y extrae su contenido hasta el primer punto, sin depender de una plantilla
106     o de sufijos como "Detalles adicionales".
107
108     Args:
109         filled_prompt (str): Texto completo del prompt ya instanciado.
110
111     Returns:
112         list[str]: Lista de intereses normalizados (en minúsculas y sin espacios
            ↪ sobrantes).
113     """
114     # Captura intereses hasta el primer punto tras ellos
115     pattern = r"intereses:\s*(?P<interests>.+?)\."
116
117     match = re.search(pattern, filled_prompt, re.IGNORECASE | re.DOTALL)
118     if match:
119         interests_str = match.group("interests").strip()
120         return [i.strip().lower() for i in interests_str.split(",") if
            ↪ i.strip()]
121     return []

```

Listing 13: `vector.py` – Módulo para inicializar los embeddings y el vector store de Azure Cognitive Search

```

1  """
2  Módulo para inicializar los embeddings y el vector store de Azure Cognitive
   ↳ Search.
3
4  Este módulo configura dos componentes esenciales para la fase de recuperación
   ↳ semántica (RAG):
5
6  1. `embeddings`: Modelo de embeddings de Azure OpenAI, encargado de convertir
   ↳ textos en vectores
7     utilizando un despliegue configurado del modelo `text-embedding-3-large` (u
   ↳ otro compatible).
8  2. `vector_store`: Objeto `AzureSearch` de LangChain configurado para realizar
   ↳ búsquedas híbridas
9     (semánticas + léxicas) sobre un índice existente en Azure Cognitive Search.
10
11  Exporta:
12  - `vector_store`: Instancia lista para ser utilizada por el agente de
   ↳ recuperación (`RetrieverAgent`).
13  """
14
15  from langchain_openai import AzureOpenAIEmbeddings
16  from langchain_community.vectorstores import AzureSearch
17
18  from config.config import (
19      AZURE_OPENAI_EMBEDDINGS_API_KEY,
20      AZURE_OPENAI_EMBEDDINGS_DEPLOYMENT,
21      AZURE_OPENAI_EMBEDDINGS_ENDPOINT,
22      AZURE_SEARCH_ENDPOINT,
23      AZURE_SEARCH_KEY,
24      INDEX_NAME,
25      API_VERSION_EMBEDDINGS,
26  )
27
28  # ----- EMBEDDINGS -----
29  # Inicializa el modelo de embeddings usando Azure OpenAI.
30  # Este modelo convierte las consultas y los documentos en vectores numéricos
   ↳ (embeddings),
31  # que posteriormente se utilizan para realizar búsquedas semánticas en Azure
   ↳ Cognitive Search.
32  # El despliegue del modelo debe haberse realizado previamente en Azure OpenAI y
   ↳ configurado en el sistema.
33  embeddings = AzureOpenAIEmbeddings(
34      azure_deployment=AZURE_OPENAI_EMBEDDINGS_DEPLOYMENT,
35      azure_endpoint=AZURE_OPENAI_EMBEDDINGS_ENDPOINT,

```

```
36     openai_api_key=AZURE_OPENAI_EMBEDDINGS_API_KEY,
37     openai_api_type="azure",
38     openai_api_version=API_VERSION_EMBEDDINGS,
39 )
40
41
42 # ----- VECTOR STORE -----
43 # Configura el almacén vectorial utilizando Azure Cognitive Search.
44 # Este objeto permite realizar búsquedas por similitud utilizando los vectores
45   ↳ generados por el modelo de embeddings.
46 vector_store = AzureSearch(
47     azure_search_endpoint=AZURE_SEARCH_ENDPOINT,
48     azure_search_key=AZURE_SEARCH_KEY,
49     index_name=INDEX_NAME,
50     embedding_function=embeddings.embed_query,
51 )
```

Listing 14: `app_test.py` – Interfaz de evaluación de la aplicación KoalaRoute utilizando Streamlit

```

1  """
2  Interfaz web para evaluación de escenarios RAG mediante Streamlit.
3
4  Este módulo define una aplicación Streamlit que permite seleccionar y ejecutar
5  ↪ escenarios de prueba
6  predefinidos. Cada escenario contiene una consulta del usuario, información
7  ↪ contextual y una respuesta esperada.
8  El flujo de evaluación consiste en:
9
10 1. Selección de un escenario desde un desplegable.
11 2. Construcción del prompt de entrada usando una plantilla y los parámetros del
12 ↪ escenario.
13 3. Ejecución del sistema completo RAG (recuperación + generación) mediante
14 ↪ `run_prompt`.
15 4. Evaluación de la respuesta generada frente a la esperada con métricas de
16 ↪ recuperación y coherencia semántica.
17 5. Visualización del resultado, incluyendo:
18     - Métricas de evaluación.
19     - Respuesta generada (itinerario propuesto).
20     - Documentos recuperados durante el proceso.
21
22 Componentes principales:
23 - `load_formatted_prompt`: Construye el prompt estructurado a partir de los
24 ↪ datos del escenario.
25 - `run_prompt`: Ejecuta el grafo de LangGraph y devuelve la respuesta generada y
26 ↪ los documentos utilizados.
27 - `Evaluator`: Calcula métricas como `Recall adaptativo` y `Coherencia
28 ↪ Semántica`.
29
30 Requiere archivos de escenario YAML ubicados en el directorio correspondiente,
31 ↪ accesibles mediante `get_available_scenarios`.
32
33 Uso:
34 Ejecutar `streamlit run webapp/app_test.py` desde la raíz del proyecto.
35 """
36
37 import streamlit as st
38
39 from modules.prompt_utils import load_formatted_prompt
40 from webapp.evaluation.evaluator import Evaluator
41 from webapp.evaluation.scenario_utils import (
42     get_available_scenarios,
43     load_scenario_by_name,
44 )

```

```

36 from webapp.runner import run_prompt
37
38 st.set_page_config(page_title="KoalaTest", page_icon="", layout="centered")
39 st.title("Evaluador de escenarios de prueba")
40
41 available_scenarios = get_available_scenarios()
42
43 if not available_scenarios:
44     st.warning("No se encontraron archivos de escenario.")
45 else:
46     selected_scenario = st.selectbox("Selecciona un escenario:",
47                                     ↪ available_scenarios)
48
49     if st.button("Ejecutar evaluación"):
50         with st.spinner("Procesando escenario..."):
51             # Cargar datos del escenario
52             scenario_data = load_scenario_by_name(selected_scenario)
53
54             # Construir prompt completo
55             interest_str = (
56                 ", ".join(scenario_data["interests"])
57                 if scenario_data.get("interests")
58                 else "cualquier tipo de actividad"
59             )
60             full_prompt = (
61                 load_formatted_prompt(
62                     "prompt_base",
63                     days=scenario_data["duration_days"],
64                     budget=scenario_data["budget"].lower(),
65                     travel_type=scenario_data["travel_type"].lower(),
66                     interests=interest_str,
67                 )
68                 + scenario_data["user_input"]
69             )
70
71             # Ejecutar el sistema
72             generated_output = run_prompt(full_prompt)
73
74             # Añadir resultados al escenario para evaluación
75             generated_response = generated_output.get("generated_response", "")
76             retrieved_docs = generated_output.get("retrieved_docs", [])
77
78             scenario_data["retrieved_docs"] = retrieved_docs
79
80             # Evaluar
81             evaluator = Evaluator()
82             result = evaluator.evaluate_scenario(scenario_data)

```

```
83     st.success("Evaluación completada.")
84     st.subheader("Resultado de la evaluación")
85     st.json(result)
86
87     st.subheader("Respuesta generada (itinerario)")
88     st.markdown(generated_response)
89
90     if retrieved_docs:
91         st.subheader("Documentos recuperados")
92         for doc in retrieved_docs:
93             st.markdown(f"- `{doc.get('id')}`")
```

Listing 15: evaluator.py – Módulo de evaluación de escenarios para sistemas RAG

```

1  """
2  Módulo de evaluación de escenarios para sistemas RAG.
3
4  Este módulo define la clase `Evaluator`, encargada de medir la calidad de las
   ↳ respuestas generadas
5  por el sistema, tanto a nivel de recuperación como de generación. Es utilizado
   ↳ para comparar
6  automáticamente las salidas del sistema con respuestas de referencia en
   ↳ escenarios de prueba definidos.
7
8  Funcionalidades:
9  - `recall_at_k`: Evalúa la calidad de la recuperación mediante la métrica de
   ↳ Recall adaptativo,
10     ajustando dinámicamente el valor de `k` según el número de documentos
   ↳ recuperados.
11  - `semantic_similarity`: Calcula la similitud semántica entre la consulta del
   ↳ usuario y los textos recuperados utilizando embeddings y similitud media del
   ↳ coseno.
12  - `evaluate_scenario`: Evalúa un escenario completo, devolviendo un diccionario
   ↳ con las métricas
13     obtenidas.
14
15  Dependencias:
16  - `embeddings`: Objeto de embeddings compartido, inicializado desde
   ↳ `modules.vector`.
17  - `K_EVAL_THRESHOLD`: Proporción configurable de documentos usados como top-K en
   ↳ el cálculo de recall.
18  """
19
20  from math import ceil
21  from typing import Dict, List, Set
22
23  from config.config import K_EVAL_THRESHOLD
24  from modules.vector import embeddings
25
26
27  class Evaluator:
28      def __init__(self):
29          """
30          Inicializa el evaluador utilizando el objeto de embeddings definido en
   ↳ `modules.vector`.
31          Este objeto permite convertir texto en vectores para calcular similitud
   ↳ semántica.
32          """
33      self.embeddings = embeddings

```

```

34
35 def recall_at_k(self, relevant_docs: Set[str], retrieved_docs: List[Dict])
    ↪ -> float:
36     """
37     Calcula el recall adaptativo entre los documentos relevantes esperados y
    ↪ los recuperados.
38
39     El valor de k se ajusta dinámicamente según la proporción
    ↪ `K_EVAL_THRESHOLD` y la longitud
40     de la lista de documentos recuperados.
41
42     Args:
43         relevant_docs (Set[str]): Conjunto de identificadores de documentos
    ↪ relevantes esperados.
44         retrieved_docs (List[Dict]): Lista de documentos recuperados por el
    ↪ sistema.
45
46     Returns:
47         float: Valor de recall@k, entre 0.0 y 1.0. Si no hay documentos
    ↪ relevantes, retorna 0.0.
48     """
49     k = max(1, ceil(K_EVAL_THRESHOLD * len(retrieved_docs)))
50     top_k_ids = [doc["id"] for doc in retrieved_docs[:k]]
51     relevant_retrieved = sum(1 for doc_id in top_k_ids if doc_id in
    ↪ relevant_docs)
52     total_relevant = len(relevant_docs)
53     return relevant_retrieved / total_relevant if total_relevant > 0 else
    ↪ 0.0
54
55 def thematic_coverage(
56     self, interests: List[str], retrieved_docs: List[Dict]
57 ) -> float:
58     """
59     Calcula la cobertura temática de los documentos recuperados respecto a
    ↪ los intereses del usuario.
60
61     Esta métrica evalúa qué proporción de los intereses proporcionados por
    ↪ el usuario
62     están presentes en las categorías de los documentos recuperados. Se
    ↪ normaliza todo el texto
63     a minúsculas para una comparación robusta.
64
65     Si no se especifican intereses, se asume cobertura total (1.0).
66
67     Args:
68         interests (List[str]): Lista de intereses definidos por el usuario.
69         retrieved_docs (List[Dict]): Lista de documentos recuperados, cada
    ↪ uno con una clave "category"

```



```

70         que contiene una lista de etiquetas
71         ↪ temáticas.
72
73     Returns:
74         float: Porcentaje de intereses cubiertos por las categorías de los
75         ↪ documentos (entre 0.0 y 1.0).
76     """
77
78     if not interests:
79         return 1.0
80
81     # Normalizar intereses y categorías
82     interests_normalized = {i.strip().lower() for i in interests}
83     retrieved_categories = set()
84
85     for doc in retrieved_docs:
86         for cat in doc.get("category", []):
87             retrieved_categories.add(cat.strip().lower())
88
89     matches = sum(1 for i in interests_normalized if i in
90     ↪ retrieved_categories)
91     return matches / len(interests_normalized) if interests_normalized else
92     ↪ 0.0
93
94 def semantic_similarity(self, retrieved_docs: List[Dict]) -> float:
95     """
96     Calcula la similitud semántica entre la consulta del usuario y los
97     ↪ documentos recuperados,
98     utilizando los valores de similitud precomputados.
99
100     Args:
101         retrieved_docs (List[Dict]): Lista de documentos recuperados, cada
102         ↪ uno con el campo 'similarity'.
103
104     Returns:
105         float: Similitud promedio entre la consulta del usuario y los
106         ↪ documentos recuperados.
107     """
108     similarities = [
109         doc.get("similarity") for doc in retrieved_docs if "similarity" in
110         ↪ doc
111     ]
112
113     return sum(similarities) / len(similarities) if similarities else 0.0
114
115 def evaluate_scenario(self, scenario: Dict) -> Dict:
116     """
117     Evalúa un escenario de prueba comparando los resultados generados con
118     ↪ los esperados.

```

```

109
110     Esta función calcula varias métricas para analizar la calidad de la
111     ↪ recuperación y la generación:
112     - Recall adaptativo: porcentaje de documentos esperados que han sido
113     ↪ recuperados, ajustado según un umbral proporcional a k.
114     - Cobertura Temática: proporción de intereses del usuario que están
115     ↪ representados en las categorías de los documentos recuperados.
116     - Coherencia Semántica: similitud entre la consulta del usuario y los
117     ↪ textos recuperados.
118
119     Args:
120     scenario (Dict): Diccionario con la información del escenario
121
122     Returns:
123     Dict: Diccionario con las métricas obtenidas para el escenario.
124     """
125
126     results = {"Escenario": scenario["name"]}
127
128     if scenario["evaluate_retrieval_docs"]:
129         # Métrica de recall adaptativo
130         recall = self.recall_at_k(
131             set(scenario["expected_relevant_docs"]),
132             ↪ scenario["retrieved_docs"]
133         )
134         results["Recall adaptativo"] = round(recall, 2)
135
136     if scenario["interests"]:
137         # Métrica de cobertura temática
138         coverage = self.thematic_coverage(
139             scenario["interests"], scenario["retrieved_docs"]
140         )
141         results["Cobertura Temática"] = round(coverage, 2)
142
143     # Métrica de coherencia semántica
144     coherence = self.semantic_similarity(scenario["retrieved_docs"])
145     results["Coherencia Semántica"] = round(coherence, 2)
146
147     return results

```

Listing 16: `scenario_utils.py` – Módulo de utilidades para la gestión de escenarios de prueba

```

1  """
2  Módulo de utilidades para la gestión de escenarios de prueba.
3
4  Este módulo proporciona funciones para interactuar con un único archivo YAML que
5  ↪ contiene
6  la definición de múltiples escenarios de evaluación. Cada escenario representa
7  ↪ un caso de uso
8  potencial para el sistema RAG (Recuperación Aumentada con Generación),
9  ↪ incluyendo campos como
10 nombre, duración, intereses, entrada del usuario, documentos esperados y
11 ↪ respuesta de referencia.
12
13 Funciones disponibles:
14 - `get_available_scenarios()`: Devuelve una lista con los nombres de todos los
15 ↪ escenarios definidos.
16 - `load_scenario_by_name(name)`: Carga y devuelve un escenario específico a
17 ↪ partir de su nombre.
18
19 El archivo YAML utilizado está ubicado en la ruta definida por `SCENARIOS_PATH`,
20 ↪ y es compartido
21 por la interfaz de Streamlit para evaluar casos de prueba.
22
23 Dependencias:
24 - PyYAML para parseo del archivo YAML.
25 - Streamlit para mostrar errores y advertencias al usuario en la interfaz
26 ↪ gráfica.
27 """
28
29 import os
30 from typing import List
31
32 import streamlit as st
33 import yaml
34
35 from config.config import SCENARIOS_PATH
36
37 def get_available_scenarios() -> List[str]:
38     """
39     Devuelve una lista con los nombres de los escenarios definidos
40     dentro del archivo único de escenarios YAML.
41
42     Returns:
43 
```

```

37         List[str]: Lista de nombres de escenarios.
38         """
39         if not os.path.exists(SCENARIOS_PATH):
40             st.warning("No se encontró el archivo de escenarios.")
41             return []
42
43         try:
44             with open(SCENARIOS_PATH, "r", encoding="utf-8") as f:
45                 data = yaml.safe_load(f)
46                 if isinstance(data, list):
47                     return [s["name"] for s in data if "name" in s]
48                 else:
49                     st.error("El archivo de escenarios no tiene el formato
50                               ↪ esperado.")
51                     return []
52         except Exception as e:
53             st.error(f"Error cargando los escenarios: {e}")
54             return []
55
56     def load_scenarioby_name(scenario_name: str) -> dict:
57         """
58         Busca y devuelve un escenario específico por su nombre dentro del archivo
59         ↪ único.
60
61         Args:
62             scenario_name (str): Nombre del escenario.
63
64         Returns:
65             dict: Escenario encontrado, o {} si no existe.
66         """
67         try:
68             with open(SCENARIOS_PATH, "r", encoding="utf-8") as f:
69                 escenarios = yaml.safe_load(f)
70                 for s in escenarios:
71                     if s.get("name") == scenario_name:
72                         return s
73             st.warning(f"No se encontró el escenario: {scenario_name}")
74             return {}
75         except Exception as e:
76             st.error(f"Error cargando el escenario: {e}")
77             return {}

```

Listing 17: app.py – Interfaz principal de la aplicación KoalaRoute utilizando Streamlit

```

1  """
2  Interfaz principal de la aplicación KoalaRoute utilizando Streamlit.
3
4  Esta aplicación permite a los usuarios generar itinerarios de viaje por
5  ↪ Australia de forma personalizada,
6  mediante un sistema basado en LLM y recuperación de contexto (RAG). El usuario
7  ↪ introduce sus preferencias
8  y el sistema construye un prompt estructurado para generar una respuesta
9  ↪ coherente y contextualizada.
10
11  Características:
12  - Entrada de texto libre para deseos del viaje.
13  - Selección de duración, presupuesto, tipo de viaje e intereses.
14  - Visualización del uso de tokens antes de enviar.
15  - Generación del itinerario mediante `run_prompt` (flujo LLM + RAG).
16
17  Uso:
18  Ejecutar `streamlit run webapp/app.py` desde la raíz del proyecto.
19  """
20
21  import streamlit as st
22  import yaml
23  from webapp.runner import run_prompt
24
25  from config.config import MAX_PROMPT_TOKENS, UI_OPTIONS_PATH
26  from modules.prompt_utils import encoding, load_formatted_prompt
27
28  # ----- CARGA DE OPCIONES DE UI DESDE YAML -----
29  with open(UI_OPTIONS_PATH, "r", encoding="utf-8") as f:
30      ui_options = yaml.safe_load(f)
31
32  # ----- CONFIGURACIÓN GENERAL DE LA PÁGINA -----
33  st.set_page_config(page_title="KoalaRoute", page_icon="", layout="centered")
34
35  st.title("KoalaRoute")
36  st.markdown("Planifica tu aventura perfecta por Australia con inteligencia y  
↪ estilo.")
37
38  # ----- ENTRADA DE DETALLES DEL VIAJE -----
39  st.markdown("## Detalles del viaje")
40  user_query = st.text_input("¿Qué te gustaría hacer o visitar?")
41
42  col1, col2 = st.columns(2)
43  with col1:

```

```

41     days = st.number_input("Duración (en días)", min_value=1, max_value=7,
42                             ↪ value=3)
43     with col2:
44         budget = st.selectbox("Presupuesto", ui_options["presupuestos"])
45     # ----- PREFERENCIAS ADICIONALES -----
46     st.markdown("## Preferencias del viaje")
47     col3, col4 = st.columns(2)
48     with col3:
49         travel_type = st.selectbox("Tipo de viaje", ui_options["tipos_viaje"])
50     with col4:
51         interests = st.multiselect(
52             "Intereses",
53             ui_options["intereses"],
54             default=["Naturaleza"],
55         )
56
57     # ----- CÁLCULO Y VISUALIZACIÓN DE TOKENS -----
58     user_token_count = len(encoding.encode(user_query))
59     tokens_remaining = max(MAX_PROMPT_TOKENS, 0)
60     progress_ratio = (
61         min(user_token_count / tokens_remaining, 1.0) if tokens_remaining > 0 else
62         ↪ 1.0
63     )
64
65     st.markdown("## Tokens disponibles para tu mensaje")
66     st.progress(
67         progress_ratio,
68         text=f"{user_token_count} / {tokens_remaining} tokens usados en tu mensaje",
69     )
70
71     # ----- CONSTRUCCIÓN DEL PROMPT -----
72     interest_str = ", ".join(interests) if interests else "cualquier tipo de
73     ↪ actividad"
74
75     # Prompt completo con sistema + entrada del usuario
76     full_prompt = (
77         load_formatted_prompt(
78             "prompt_base",
79             days=days,
80             budget=budget.lower(),
81             travel_type=travel_type.lower(),
82             interests=interest_str,
83         )
84         + user_query
85     )
86
87     # ----- BOTÓN Y LÓGICA DE GENERACIÓN -----

```

```

86 if st.button("Generar itinerario"):
87     if not user_query.strip():
88         st.warning("Por favor, describe tu viaje.")
89     elif not isinstance(days, int) or not 1 <= days <= 30:
90         st.error("La duración del viaje debe estar entre 1 y 30 días.")
91     elif user_token_count > tokens_remaining:
92         st.error(
93             f"Tu mensaje usa {user_token_count} tokens, pero solo puedes usar un
94             ↪ máximo de {tokens_remaining} "
95             "tokens debido al espacio reservado para instrucciones del sistema.
96             ↪ Reduce la longitud o complejidad del mensaje."
97         )
98     else:
99         with st.spinner("Trazando tu ruta ideal..."):
100             try:
101                 response = run_prompt(full_prompt)["generated_response"]
102                 st.success("Tu itinerario personalizado:")
103                 st.markdown(response, unsafe_allow_html=True)
104             except Exception as e:
105                 st.error(f"Error: {e}")

```

Listing 18: `runner.py` – Ejecutor principal del grafo LangGraph

```

1  """
2  Ejecutor principal del grafo LangGraph.
3
4  Este módulo sirve como punto de entrada para interactuar con el flujo de agentes
   ↳ definido
5  en el grafo de LangGraph. Permite ejecutar una única interacción completa con el
   ↳ sistema RAG,
6  recibiendo una entrada del usuario y devolviendo tanto la respuesta generada
   ↳ como los
7  documentos relevantes recuperados.
8
9  Responsabilidades:
10 - Instancia el grafo mediante `build_langgraph_controller_flow`.
11 - Expone la función `run_prompt`, que toma una consulta del usuario y devuelve
   ↳ un diccionario
12   con la respuesta final generada por el modelo y la lista de documentos
   ↳ utilizados.
13
14 La gestión de errores se realiza mediante trazas impresas, facilitando la
   ↳ depuración
15 en entornos de desarrollo local.
16 """
17
18
19 import traceback
20
21 from modules.graph.agent_state import AgentState
22 from modules.graph.graph import build_langgraph_controller_flow
23
24 # Construcción del grafo de agentes (controlador + retriever + LLM)
25 dialogue_manager = build_langgraph_controller_flow()
26
27
28 def run_prompt(user_query: str) -> str:
29     """
30     Ejecuta una única interacción con el grafo de agentes a partir de una
   ↳ consulta del usuario.
31
32     Este flujo sigue la lógica:
33         input del usuario → recuperación opcional → generación con LLM →
   ↳ respuesta final
34
35     Args:
36         user_query (str): Texto introducido por el usuario (consulta o petición
   ↳ de itinerario).

```



```

37
38     Returns:
39         dict: Diccionario con los siguientes campos:
40             - "generated_response": Respuesta generada por el modelo.
41             - "retrieved_docs": Lista de identificadores de documentos
42               ↪ recuperados (formato "título#sección").
43
44     Raises:
45         RuntimeError: Si ocurre algún error durante la ejecución del grafo.
46     """
47     try:
48         # Crear el estado inicial con la entrada del usuario
49         state = AgentState(input=user_query, response="")
50
51         # Ejecutar el grafo
52         result = dialogue_manager.invoke(state)
53
54         # Retornar la respuesta generada
55         return {
56             "generated_response": result.get("response", ""),
57             "retrieved_docs": result.get("retrieved_docs", []),
58         }
59
60     except Exception as e:
61         print("ERROR ejecutando el grafo:")
62         traceback.print_exc()
63         raise RuntimeError("Fallo en el grafo") from e

```

Índice de figuras

4.1. Flujo de ejecución del sistema con agentes orquestados en LangGraph	25
4.2. Flujo de ejecución entre agentes orquestados en LangGraph	31
4.3. Flujo de interacción entre el usuario y el sistema desde la interfaz web	34

Índice de tablas

3.1. Estado del arte en orquestación de agentes de IA con RAG y LLM . .	24
5.1. Escenarios de prueba definidos para el sistema	38
5.2. Resultados de evaluación del escenario 1	40
5.3. Resultados de evaluación del escenario 2	41
5.4. Resultados de evaluación del escenario 3	41
5.5. Resultados de evaluación del escenario 4	41
5.6. Resultados de evaluación del escenario 5	41
5.7. Resultados de evaluación del escenario 6	42
5.8. Resultados de evaluación del escenario 7	42
5.9. Resultados de evaluación del escenario 8	42
A.1. Itinerario generado: Escenario 1	56
A.2. Itinerario generado: Escenario 2	57
A.3. Itinerario generado: Escenario 3	58
A.4. Itinerario generado: Escenario 4	59
A.5. Itinerario generado: Escenario 5	60
A.6. Itinerario generado: Escenario 6	61
A.7. Itinerario generado: Escenario 7	62
A.8. Itinerario generado: Escenario 8	63

Siglas

API	Interfaz de Programación de Aplicaciones
DM	Modelo de Difusión
GAN	Red Generativa Adversativa
IA	Inteligencia Artificial
IAG	Inteligencia Artificial Generativa
LLM	Modelo de Lenguaje Grande
LSTM	Red de memoria a corto-largo plazo
RAG	Generación Aumentada por Recuperación
RNN	Red neuronal recurrente
SaaS	Software como Servicio
SQL	Lenguaje de Consulta Estructurada

Referencias

- Anthropic. (2023). Introducing Claude.
- Asai, A., Wang, X., Lin, J., Lewis, P., Riedel, S., & Hajishirzi, H. (2023). Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks: A Survey.
- Azure, M. (2023a). *Azure Cognitive Search Documentation* [Accedido en junio de 2025]. <https://learn.microsoft.com/en-us/azure/search/>
- Azure, M. (2023b). *Azure OpenAI Service Documentation* [Accedido en junio de 2025]. <https://learn.microsoft.com/en-us/azure/cognitive-services/openai/>
- Barua, S. (2024). Exploring Autonomous Agents through the Lens of Large Language Models: A Review.
- Bommasani, R., Hudson, J., Adeli, E., et al. (2021). On the Opportunities and Risks of Foundation Models. *arXiv preprint arXiv:2108.07258*.
- Bousetouane, F. (2025). Agentic Systems: A Guide to Transforming Industries with Vertical AI Agents.
- Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. (2020). Language Models are Few-Shot Learners. *Advances in Neural Information Processing Systems*, 33, 1877-1901.
- Chen, Y., Yan, L., Sun, W., Ma, X., Zhang, Y., Wang, S., Yin, D., Yang, Y., & Mao, J. (2025). Improving Retrieval-Augmented Generation through Multi-Agent Reinforcement Learning.
- de Aquino e Aquino, G., da Silva de Azevedo, N., Okimoto, L. Y. S., Camelo, L. Y. S., de Souza Bragança, H. L., Fernandes, R., Printes, A., Cardoso, F., Gomes, R., & Torné, I. G. (2025). From RAG to Multi-Agent Systems: A Survey of Modern Approaches in LLM Development. *Preprints*.
- DeepMind, G. (2023). Gemini: Our largest and most capable AI models.
- Docs, M. (2022). Hub-spoke network topology in Azure.
- Ferrag, M. A., Tihanyi, N., & Debbah, M. (2025). From LLM Reasoning to Autonomous AI Agents: A Comprehensive Review.
- Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., & Bengio, Y. (2014). Generative Adversarial Nets. *Advances in Neural Information Processing Systems*, 27.

- Huang, L., Yu, W., Ma, W., Zhong, W., Feng, Z., Wang, H., Chen, Q., Peng, W., Feng, X., Qin, B., & Liu, T. (2025). A Survey on Hallucination in Large Language Models: Principles, Taxonomy, Challenges, and Open Questions. *ACM Transactions on Information Systems*, 43(2), 1-55.
- Inc., L. (2023). LangGraph: Composable Multi-Actor RAG Workflows [Accessed: 2025-06-20].
- Inc., S. (2024). *Streamlit: The fastest way to build and share data apps* [Accessed: 2025-06-19].
- Izacard, G., & Grave, E. (2020). Leveraging Passage Retrieval with Generative Models for Open Domain Question Answering. *arXiv preprint arXiv:2007.01282*.
- Ji, Z., Lee, N., Fries, J., Bernstein, M., Zhang, T. B., & Zou, J. Y. (2023a). A Survey of Hallucination in Natural Language Generation [Preprint: <https://arxiv.org/abs/2303.02438>].
- Ji, Z., Lee, N., Fries, J., et al. (2023b). Survey of Hallucination in Natural Language Generation. *ACM Computing Surveys*, 55(12), 1-38.
- Jiang, Y., & AI, M. (2023). Mistral 7B Technical Report.
- Karpukhin, V., Oguz, B., Min, S., Lewis, P., Wu, L., Edunov, S., Chen, D., & Yih, W.-t. (2020). Dense Passage Retrieval for Open-Domain Question Answering. *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 6769-6781.
- Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W.-t., Rocktäschel, T., Riedel, S., & Kiela, D. (2020). Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems*, 33, 9459-9474.
- Lin, J., Ma, X., & Shen, Y. (2021). A Few Brief Notes on Deep Retrieval. *Proceedings of the 1st Workshop on Open-Domain Question Answering*.
- Maynez, J., Narayan, S., Bohnet, B., & McDonald, R. (2020). On Faithfulness and Factuality in Abstractive Summarization. *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, 1906-1919.
- Microsoft Corporation. (2024). *Azure AI Search* [Accessed: 2025-06-19]. <https://azure.microsoft.com/en-us/products/search>
- Nguyen, T., Chin, P., & Tai, Y.-W. (2025). MA-RAG: Multi-Agent Retrieval-Augmented Generation via Collaborative Chain-of-Thought Reasoning.
- OpenAI. (2022). ChatGPT: Optimizing Language Models for Dialogue [<https://openai.com/blog/chatgpt>].
- OpenAI. (2023a). *ChatML Format Specification* [Accedido en junio de 2025]. <https://platform.openai.com/docs/guides/gpt/chatml>
- OpenAI. (2023b). GPT-3.5-turbo model documentation [<https://platform.openai.com/docs/models/gpt-3-5>].
- OpenAI. (2023c). GPT-4 Technical Report.
- Penedo, G., et al. (2023). The RefinedWeb Dataset and Falcon LLMs.
- Rombach, R., Blattmann, A., Lorenz, D., Esser, P., & Ommer, B. (2022). High-Resolution Image Synthesis with Latent Diffusion Models. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 10684-10695.
- Russell, S., & Norvig, P. (2021). *Artificial Intelligence: A Modern Approach*. Pearson.
- Seabra, A., Cavalcante, C., Nepomuceno, J., Lago, L., Ruberg, N., & Lifschitz, S. (2024). Dynamic Multi-Agent Orchestration and Retrieval for Multi-Source Question-Answer Systems using Large Language Models.

- Singh, A., Ehtesham, A., Kumar, S., & Khoei, T. T. (2025). Agentic Retrieval-Augmented Generation: A Survey on Agentic RAG.
- Strubell, E., Ganesh, A., & McCallum, A. (2019). Energy and Policy Considerations for Deep Learning in NLP. *arXiv preprint arXiv:1906.02243*.
- Touvron, H., et al. (2023). LLaMA: Open and Efficient Foundation Language Models.
- Utica Babyak, M. (2025, junio). *Koala Route* (Ver. 1.0.0).
- Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is All you Need. En I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan & R. Garnett (Eds.), *Advances in Neural Information Processing Systems* (Vol. 30). Curran Associates, Inc.
- Wooldridge, M. (2009). *An Introduction to MultiAgent Systems*. John Wiley & Sons.
- Xiong, L., Xiong, C., Li, Y., Tang, K.-F., Liu, J., Bennett, P., Ahmed, J., & Bosselut, A. (2021). Approximate Nearest Neighbor Negative Contrastive Learning for Dense Text Retrieval [arXiv:2007.00808]. *International Conference on Learning Representations (ICLR)*.
- Zhu, J., Yan, L., Shi, H., Yin, D., & Sha, L. (2024). ATM: Adversarial Tuning Multi-agent System Makes a Robust Retrieval-Augmented Generator.