
Exploración del potencial de agentes LLM en la asistencia al onboarding de proyectos software

Martín López de Ipiña Muñoz



Dirección

Galdos Otermin, Aritz • Pereira Varela, Juanan • Azanza Sesé, Maider

11 de junio de 2025

Resumen

El proceso de incorporación de nuevos desarrolladores a proyectos software presenta desafíos debido a la sobrecarga informativa que supone sintetizar información dispersa en múltiples fuentes como código fuente, documentación y sistemas de gestión. Este trabajo implementa y evalúa un sistema multiagente basado en grandes modelos de lenguaje (LLM) para asistir el proceso de onboarding, compuesto por cinco agentes especializados con acceso a fuentes específicas mediante técnicas RAG, protocolo MCP y mecanismos de orquestación con planificadores y coordinadores. El sistema incorpora estrategias de prompting, memoria persistente y diseño adaptativo, evaluándose mediante métricas automatizadas sobre un dataset de 46 preguntas derivadas de una elicitation de requisitos con profesionales de LKS Next. Los resultados demuestran una precisión superior al 80 % en consultas de onboarding, destacando la efectividad de las arquitecturas con planificación unificada y la eficacia del protocolo MCP para la integración de sistemas externos. Las evidencias obtenidas posicionan a los agentes LLM como una solución prometedora para optimizar el onboarding en entornos productivos, promoviendo la autonomía de las personas incorporadas y reduciendo la carga sobre el equipo, siempre que se implemente una supervisión apropiada del sistema.

Objetivos de Desarrollo Sostenible

Este proyecto se relaciona principalmente con el ODS 4 (Educación de Calidad), facilitando formación más asequible, y contribuye al ODS 10 (Reducción de las Desigualdades).

Los sistemas de incorporación tradicionales dependen de mentores humanos. Esta asistencia consume recursos valiosos que no siempre es viable invertir, complicando los procesos de integración. Mediante la exploración realizada en este proyecto, se contribuye al desarrollo de sistemas de incorporación semiautomatizados que demandan menos recursos humanos especializados. De este modo, la calidad de la formación en proyectos de software se independiza parcialmente de los recursos disponibles, proporcionando un marco educativo más igualitario alineado con el ODS 4 y reduciendo las desigualdades de acceso a la formación técnica cualificada conforme al ODS 10.

No obstante, al tratarse los agentes LLM de tecnologías emergentes, los sistemas en producción son escasos y su disponibilidad se limita a grupos con mayores recursos económicos. Por ejemplo, el sistema agéntico Claude Code está disponible únicamente mediante una suscripción mensual superior a 100 euros en España, mientras que exploraciones que requieren múltiples instancias simultáneas pueden superar los 1000 euros mensuales.

Paradójicamente, si estas tecnologías resultan tan eficaces como prometen, las organizaciones con mayores recursos serán las más beneficiadas, disponiendo de las versiones más avanzadas y obteniendo como resultado ventajas competitivas. En un mercado donde todas las organizaciones rivalizan, esto podría incrementar las desigualdades.

La solución más directa residiría en regular estos sistemas para garantizar una competencia equitativa. Sin embargo, tales regulaciones limitarían a organizaciones privilegiadas como OpenAI, cuya labor consiste precisamente en desarrollar estas tecnologías. Cabe preguntarse si habrían sido desarrolladas inicialmente bajo un marco regulatorio más restrictivo. Esto plantea la cuestión: ¿se debe limitar el avance tecnológico en favor de la igualdad? Podría argumentarse que la calidad de vida del más próspero en la época medieval era inferior a la del más desfavorecido en la actualidad.

Por tanto, ¿fomentan estos sistemas la educación de calidad y la reducción de las desigualdades? En caso de ser regulados, probablemente sí. ¿Deberían serlo? Posicionarse ante esta cuestión política no sería más que una especulación sesgada por consideraciones personales. Afortunadamente, es responsabilidad de aquellos cuya labor investigadora se centra en este tipo de dilemas, y cuyos esfuerzos han resultado en los ODS.

Índice de contenidos

Índice de contenidos	v
Índice de figuras	x
Índice de tablas	xii
Índice de algoritmos	1
1 Introducción	1
2 Antecedentes	3
2.1. Onboarding en proyectos software	3
2.1.1. Trabajo previo	4
2.2. Agentes de Grandes Modelos de Lenguaje (LLM)	4
2.2.1. Modelos LLM	4
2.2.2. Interacción con herramientas externas	5
2.2.3. Abstracciones en frameworks	6
2.3. Model Context Protocol (MCP)	7
2.4. Estado del arte en arquitecturas de agentes LLM	8
2.4.1. Arquitectura RAG	8
2.4.2. Arquitecturas de interacción entre agentes	9
2.4.3. Evaluación de agentes	11
2.5. Agentes LLM en proyectos software	11
2.6. Ajuste de modelos para agentes LLM	12
3 Planificación	13
3.1. Alcance	13
3.1.1. Objetivos concretos del proyecto	14
3.1.2. Fases del proyecto	14
3.1.3. Descomposición de tareas	15
3.2. Periodos de realización de tareas e hitos	18

3.2.1. Dependencias entre tareas	18
3.2.2. Diagrama de Gantt	20
3.2.3. Hitos	20
3.3. Gestión del tiempo	22
3.3.1. Estimación de cada tarea	22
3.4. Gestión de riesgos	22
3.5. Gestión de interesados	23
3.6. Gestión de Comunicaciones e Información	24
3.6.1. Sistema de información	24
3.6.2. Sistema de comunicación	25
3.7. Herramientas disponibles	25
4 Captura de requisitos	27
4.1. Requisitos principales	27
4.1.1. Requisitos funcionales	27
4.1.2. Requisitos no funcionales	28
4.2. Dominios de conocimiento	29
4.2.1. Anotación de preguntas	29
4.2.2. Metodología empresarial	30
4.3. Recursos a utilizar	31
4.3.1. Proyecto software	31
4.3.2. Recursos generados	31
5 Diseño del sistema	33
5.1. Diseño de agentes	33
5.2. Arquitectura del sistema	34
5.2.1. Diagrama de agentes	36
5.3. Servidores MCP utilizados	38
5.3.1. Servidores SSE	38
5.3.2. Servidores STUDIO	39
5.4. Cliente MCP	41
5.5. Estructura del proyecto	43
6 Agentes especializados	45
6.1. Estructura SpecializedAgent	46
6.1.1. Gestión de herramientas	47
6.1.2. Sistema de citas	47
6.2. Agentes implementados	49
6.2.1. Agente código	49
6.2.2. Agente Google Drive	53
6.2.3. Agente Sistema de Ficheros	53

6.2.4. Agente Confluence	54
6.2.5. Agente GitLab	55
7 Orquestación de agentes	57
7.1. Agente Principal	57
7.1.1. Respuesta estructurada	58
7.1.2. Agente Planificador	59
7.2. Agente orquestador	60
7.3. Variaciones de orquestación	60
7.3.1. Caso práctico	61
8 Mejoras introducidas	65
8.1. Prompting Few-Shot	65
8.2. Memoria persistente	66
8.2.1. Agrupación de memoria	67
8.3. Diseño adaptativo	68
8.3.1. Criterios de clasificación	69
8.3.2. Aumento de datos	70
8.3.3. Clasificación de preguntas	70
9 Evaluación	73
9.1. Sistema de evaluación	73
9.1.1. Métricas de evaluación	74
9.1.2. Implementación de evaluación	75
9.1.3. Dataset ground truth	77
9.2. Resultados obtenidos	78
9.2.1. Mejora de versión inicial	78
9.2.2. Variaciones de orquestación	81
9.2.3. Integración de memoria	82
10 Seguimiento y control	83
10.1. Gestión del alcance	83
10.2. Gestión de riesgos	84
10.2.1. R1-Concurrencia exploratoria	84
10.2.2. R2-Variabilidad del alcance	84
10.2.3. R4-Filtrado de credenciales	84
10.2.4. R5-Pérdida de recursos	85
10.3. Gestión del tiempo	85
10.3.1. Gestión de fechas	85
10.3.2. Dedicaciones	86
11 Retos	89

11.1. Análisis y diseño	89
11.2. Comportamiento de agentes	90
11.3. Desafíos técnicos	90
12 Conclusiones y líneas futuras	91
12.1. Conclusiones	91
12.1.1. Líneas generales	91
12.1.2. Protocolo MCP	92
12.1.3. Orquestación de agentes	92
12.2. Objetivos del proyecto	92
12.3. Lecciones aprendidas	93
12.4. Líneas futuras	93
A Definición de términos técnicos	95
A.1. Conjunto de datos etiquetados	95
A.2. Entrenamiento de redes neuronales	95
A.3. Distancia coseno	96
A.4. Tokenizador	96
A.5. Pool de conexiones asíncronas	96
A.6. Tabla de cierre (<i>Closure Table</i>)	96
B Elicitación de preguntas	99
C Actas de reuniones	109
A.1. Acta de reunión TFG (25/02/2025)	109
A.1.1. Orden del día	109
A.1.2. Resumen de la reunión	110
A.1.3. Estado del proyecto	110
A.1.4. Decisiones	110
A.2. Acta de reunión TFG (12/03/2025)	111
A.2.1. Orden del día	111
A.2.2. Resumen de la reunión	111
A.2.3. Estado del proyecto	112
A.2.4. Decisiones	112
A.3. Acta de reunión TFG (20/03/2025)	113
A.3.1. Orden del día	113
A.3.2. Resumen de la reunión	113
A.3.3. Estado del proyecto	113
A.3.4. Decisiones	114
A.4. Acta de reunión TFG (02/04/2025)	115
A.4.1. Orden del día	115

A.4.2.	Resumen de la reunión	115
A.4.3.	Estado del proyecto	115
A.4.4.	Decisiones	116
A.5.	Acta de reunión TFG (04/04/2025)	117
A.5.1.	Orden del día	117
A.5.2.	Resumen de la reunión	117
A.5.3.	Decisiones	117
A.6.	Acta de reunión TFG (15/04/2025)	119
A.6.1.	Orden del día	119
A.6.2.	Resumen de la reunión	119
A.6.3.	Estado del proyecto	119
A.6.4.	Decisiones	120
A.7.	Acta de reunión TFG (29/04/2025)	121
A.7.1.	Orden del día	121
A.7.2.	Resumen de la reunión	121
A.7.3.	Estado del proyecto	122
A.7.4.	Decisiones	122
D	Detalles de entrenamiento	123
A.1.	Configuración del Dataset y Arquitectura del Modelo	123
A.2.	Estrategias de Entrenamiento	124
A.3.	Metodología Experimental	124
A.4.	Resultados	125
Bibliografía		127

Índice de figuras

2.1.	Ejemplo de interacción de un modelo LLM con una herramienta externa	6
2.2.	Esquema de funcionamiento del Model Context Protocol	8
2.3.	Esquema de funcionamiento de la arquitectura RAG en un LLM Fuente	9
2.4.	Técnicas avanzadas de RAG	10
3.1.	Estructura de Descomposición de Trabajo (EDT) del proyecto	15
3.2.	Dependencias entre tareas del proyecto	19
3.3.	Diagrama de Gantt del proyecto	21
3.4.	Estimación horaria de cada tarea	26
5.1.	Esquema de implementación de patrón ReAct en Langgraph	34
5.2.	Diagrama de clases UML de del sistema de agentes.	35
5.3.	Esquema general de agentes del sistema	36
5.4.	Estructura de directorios principales del proyecto	44
6.1.	Agentes especialistas junto a sus respectivas fuentes de datos	45
6.2.	Grafo de ejecución de agentes especializados	46
6.3.	Diagrama de funcionamiento del sistema de citas	48
6.4.	Flujo operativo del agente de código	49
6.5.	Diagrama relacional de la base de datos con el código fuente del proyecto software . .	51
6.6.	Algoritmo de chunking considerando definiciones de funciones y clases en el código fuente	52
6.7.	Comparación de agente Confluence estándar y enfocado en el cacheo del prompt . .	55
7.1.	Flujo operativo del agente planificador	60
7.2.	Variaciones de orquestación principales experimentadas	62
8.1.	Flujo operativo del sistema de memoria de los agentes especializados	67
8.2.	Agrupación de memoria por clústeres	68
8.3.	Agrupación de 3 clústeres	69
9.1.	Mecanismo de evaluación de agentes	75
9.2.	Comparación de métricas entre versión inicial y mejorada de agentes especialistas .	79

9.3. Evaluación de agentes orquestador y planificador antes y después de prompting few-shot	80
9.4. Comparación de rendimiento y costo entre diferentes variaciones de orquestación . .	81
9.5. Resultados de evaluación para el sistema con y sin memoria	82
10.1. Comparación de dedicación horaria estimada y real	87

Índice de tablas

3.1. Cronograma de Hitos del Proyecto	21
10.1. Cronograma de Hitos del Proyecto	85
C.1. Tareas asignadas (25/02/2025)	110
C.2. Tareas asignadas (12/03/2025)	112
C.3. Tareas asignadas (20/03/2025)	114
C.4. Tareas asignadas (02/04/2025)	116
C.5. Tareas asignadas (04/04/2025)	118
C.6. Tareas asignadas (15/04/2025)	120
C.7. Tareas asignadas (29/04/2025)	122
D.1. Proceso de búsqueda de hiperparámetros y resultados óptimos	125

Introducción

En la última década, el campo de la generación de contenido mediante inteligencia artificial ha experimentado avances significativos [1], culminando en el desarrollo de los grandes modelos de lenguaje (LLM). Estos modelos, capaces de comprender y generar texto en lenguaje natural, han abierto numerosas posibilidades para la automatización de tareas cada vez más complejas. Su uso programático, representado mediante agentes LLM, se ha convertido en objeto de estudio en múltiples áreas de aplicación [2, 3, 4].

Dentro de estos casos de uso, la incorporación de nuevos desarrolladores a proyectos software (*onboarding*) resulta especialmente relevante. Este proceso implica la búsqueda y síntesis de información dispersa en código fuente, documentación y aplicaciones de gestión, tareas que los agentes LLM abordan con especial eficacia. Aunque muchos proyectos buscan automatizar el proceso de desarrollo completo [5, 6, 7], pocos trabajos han explorado específicamente el potencial formativo en este contexto.

Paralelamente, para optimizar la efectividad de dichos agentes, han surgido diversas arquitecturas como *Retrieval Augmented Generation* (RAG) [8] y *Reasoning and Acting* (ReAct) [9]. Asimismo, tecnologías como LangChain¹, LangGraph² y LangSmith³ constituyen marcos de desarrollo para facilitar su aplicación. En este contexto, el *Model Context Protocol* (MCP)⁴ emerge con el objetivo de estandarizar el funcionamiento de los agentes independientemente de su implementación específica.

Aprovechando este auge tecnológico, LKS Next⁵ desarrolla productos que automatizan tareas

¹LangChain: <https://www.langchain.com/>

²LangGraph: <https://www.langchain.com/langgraph>

³LangSmith: <https://www.langchain.com/langsmith>

⁴Model Context Protocol: <https://docs.anthropic.com/en/docs/agents-and-tools/mcp>

⁵LKS Next <https://www.lksnext.com/es/>

en producción mediante estos sistemas. Sin embargo, el carácter innovador requiere un estudio constante de los últimos avances, objetivo que persigue el presente Trabajo de Fin de Grado. Mediante la exploración de estas tecnologías en el ámbito del onboarding, se busca analizar arquitecturas de aplicación efectivas y evaluar el uso del protocolo MCP.

El sistema desarrollado, junto con su código fuente completo, se encuentra disponible públicamente en el repositorio del proyecto en GitHub: https://github.com/MartinLopezDeIpina/tfg_agentes_software

El resto del presente documento se organiza como sigue: el Capítulo 2 introduce tanto el dominio de aplicación como el técnico. Los Capítulos 3 y 4 presentan la planificación del proyecto y los requisitos correspondientes. Posteriormente, los Capítulos 5, 6, 7, 8 y 9 detallan el desarrollo del proyecto y el sistema implementado, comenzando por el diseño y arquitectura general, para continuar con los agentes concretos y su evaluación. Finalmente, el Capítulo 10 presenta el seguimiento y control del proyecto, mientras que los Capítulos 11 y 12 exponen los retos y conclusiones finales.

CAPÍTULO

2

Antecedentes

Una vez establecido el objetivo de este proyecto en la introducción, en este capítulo se describirán los conceptos generales necesarios para la comprensión de este documento. Para ello, en primer lugar se detalla el proceso de onboarding en proyectos software y las dificultades que presenta, junto con el trabajo previo realizado en este ámbito.

Por otro lado, se explicará a grandes rasgos qué son los agentes basados en grandes modelos de lenguaje (*Large Language Models*), su funcionamiento e interacción con herramientas externas. Se introduce además el Model Context Protocol como estándar de comunicación entre estos componentes.

Finalmente, se aborda el estado del arte en arquitecturas de agentes, sus aplicaciones en proyectos software, técnicas de evaluación y varias consideraciones sobre el ajuste de estos.

2.1. Onboarding en proyectos software

El proceso de incorporación de nuevos desarrolladores de software constituye un desafío persistente para las organizaciones tecnológicas, donde los recién incorporados enfrentan una sobrecarga informativa mientras los desarrolladores senior ven afectada su productividad al destinar tiempo considerable a actividades de formación y mentoría [10].

Si bien prácticas como la designación de mentores han demostrado ser efectivas para facilitar la integración de nuevos miembros, estas incrementan significativamente la carga de trabajo sobre los profesionales experimentados, generando potenciales retrasos en los proyectos [11].

En este contexto, los modelos de lenguaje de gran escala emergen como una alternativa prometedora para transformar el proceso de onboarding. Su capacidad para sintetizar y razonar sobre una cantidad de información cada vez mayor ofrece la promesa de una orientación personaliza-

da e instantánea que podría reducir la dependencia de los desarrolladores senior, preservar la productividad global de los equipos, y facilitar una incorporación más eficiente [12].

2.1.1. Trabajo previo

La Universidad del País Vasco, en colaboración con LKS Next, desarrolló el prototipo denominado I Need a Hero (INAH), diseñado para aprovechar el potencial de los LLM en la localización de expertos dentro de la organización [13]. INAH opera en dos fases: primero crea perfiles de “héroes” extrayendo información de currículos de empleados dispuestos a asistir; luego, ante una consulta, utiliza GPT-3.5 para identificar las competencias requeridas y localizar a los profesionales que las poseen.

En una línea de estudio complementaria, un trabajo reciente ha presentado el sistema “Onboarding Buddy”, el cual implementa una arquitectura multiagente que organiza diversos componentes especializados para proporcionar asistencia contextualizada durante la incorporación de nuevos desarrolladores [14]. El sistema fundamenta su funcionamiento en la generación dinámica de planes mediante cadena de pensamiento 2.4.2, evaluados posteriormente para determinar su posible descomposición en sub-tareas procesadas en paralelo por otros agentes.

Aunque las investigaciones anteriores han demostrado el potencial de los agentes en este ámbito, dichas propuestas se limitan a obtener información desde una única fuente de datos. Por tanto, el presente trabajo pretende explorar el potencial de dichos sistemas en escenarios similares a entornos productivos, donde la información se encuentra distribuida entre múltiples fuentes.

2.2. Agentes de Grandes Modelos de Lenguaje (LLM)

Los agentes de Inteligencia Artificial son programas informáticos que implementan modelos computacionales para ejecutar diversas funciones específicas del contexto en el que se aplican. Tras siete décadas y media de investigación, los esfuerzos en el campo se han focalizado en agentes basados en grandes modelos de lenguaje.

2.2.1. Modelos LLM

Los Grandes Modelos de Lenguaje son redes neuronales especializadas en el procesamiento del lenguaje natural que funcionan mediante un mecanismo de entrada-salida de tokens 2.2.1. Estos modelos reciben secuencias de tokens como entrada, denominada comúnmente "*prompt*", y generan secuencias de tokens como salida, aplicando durante este proceso las representaciones y relaciones semánticas aprendidas durante su fase de entrenamiento con extensos corpus textuales [1].

Para comprender el funcionamiento de estos agentes, resulta imprescindible asimilar previamente conceptos como la tokenización, las representaciones vectoriales del lenguaje y el ajuste de dichos modelos.

Tokens Los tokens constituyen la unidad mínima de texto que el modelo puede procesar. Dado que dichos modelos operan sobre estructuras matemáticas, requieren transformar el lenguaje natural en representaciones matriciales. Para lograr esta conversión, el texto se segmenta en dichas unidades mínimas, que pueden corresponder a caracteres individuales, fragmentos de texto o palabras completas. El conjunto íntegro de estas unidades reconocibles por el modelo configura su vocabulario.

Representaciones vectoriales Constituyen vectores numéricos de dimensionalidad fija que codifican la semántica inherente a cada token. Por ejemplo, una dimensión específica podría especializarse en representar conceptos abstractos. En este contexto, la representación vectorial del token “animal” contendría un valor más elevado en dicha dimensión que la correspondiente al término “gato”, reflejando su mayor grado de abstracción conceptual.

Ajuste de modelos instruct El entrenamiento de los LLM (véase Anexo A.2) se estructura en dos fases diferenciadas. La primera corresponde al preentrenamiento, donde el modelo procesa extensos conjunto de datos textuales con operaciones como intentar predecir el siguiente token en la secuencia. Esta fase permite al modelo captar las complejas estructuras sintácticas y relaciones semánticas inherentes al lenguaje natural. La segunda fase consiste en el ajuste fino, donde el modelo previamente entrenado se especializa mediante conjuntos de datos específicos y etiquetados (consulte Anexo A.1), optimizando su capacidad para ejecutar tareas concretas de clasificación o generación de texto.

Los agentes basados en LLM implementan en su mayoría modelos *instruct*, variantes especialmente ajustadas para responder a consultas e instrucciones de usuarios. Dentro de esta categoría se encuentran GPT (base de ChatGPT¹) de OpenAI², Claude Sonnet de Anthropic³, y LLama-Instruct de Meta⁴.

2.2.2. Interacción con herramientas externas

Los agentes LLM poseen la capacidad de interactuar con diversas herramientas como búsquedas web, bases de datos o interfaces de usuario. Fundamentalmente, este tipo de modelos solo genera tokens de texto, por lo que la integración de herramientas se implementa mediante palabras clave o tokens especiales que este puede incluir en su salida. Para ello, en el texto de entrada se especifica el esquema de la función a utilizar y, si decide emplearla, el modelo generará el texto correspondiente. Posteriormente, se procesa la respuesta para extraer llamadas a funciones si las hubiese.

La interacción con herramientas sigue un patrón secuencial. Tras realizar la llamada a la herramienta, la salida de esta se utilizará como entrada para el siguiente mensaje del modelo. La

¹ChatGPT:<https://chatgpt.com>

²OpenAI:<https://openai.com/>

³Anthropic:<https://www.anthropic.com/>

⁴Meta: <https://about.meta.com/es/>

Figura 2.1 ilustra el esquema de un agente con acceso a una API del clima que consulta información meteorológica. Como el modelo carece de dicha información en tiempo real, se le indica en el prompt la posibilidad de invocar esta función. Al incluir la llamada en su texto de salida, se ejecuta la función y su respuesta se transmite al modelo para generar el resultado final.

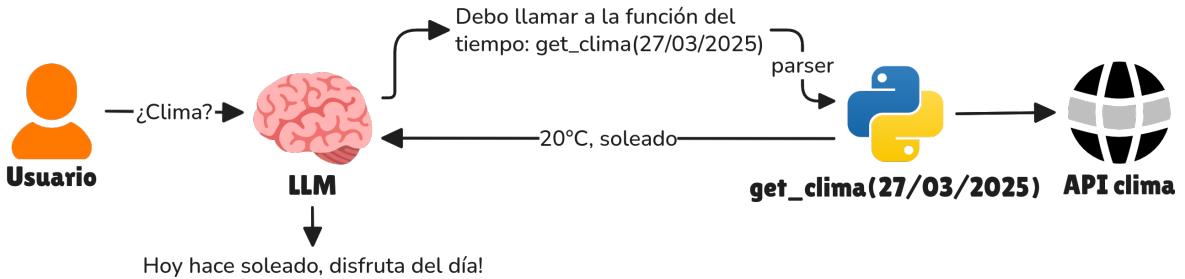


Figura 2.1: Ejemplo de interacción de un modelo LLM con una herramienta externa.

2.2.2.1. Patrón ReAct

El agente Reasoning and Act (ReAct) constituye uno de los patrones más utilizados [9]. Se basa en un ciclo de tres pasos fundamentales: el razonamiento como generación de pensamiento sobre posibles acciones, la acción como ejecución de la herramienta seleccionada y la observación como procesamiento del resultado obtenido. Este ciclo se repite iterativamente hasta que durante la fase de razonamiento se determina que la tarea ha sido completada.

2.2.3. Abstracciones en frameworks

En pleno auge de los agentes LLM, surgen cada vez más bibliotecas y frameworks que estandarizan su implementación. Estos marcos de trabajo ofrecen abstracciones de alto nivel para reutilizar funcionalidades comunes presentes en la mayoría de sistemas de agentes. Las principales funcionalidades proporcionadas son las siguientes:

- **Gestión de modelos:** la ejecución de modelos de lenguaje requiere del dominio de estos, ya que cada uno posee tokenizadores (véase Anexo A.4) específicos y esquemas propios de entrada y salida. Los frameworks ofrecen interfaces unificadas, facilitando el uso de diversos modelos sin necesidad de conocimientos técnicos excesivamente detallados.
- **Interacción conversacional:** la comunicación con los agentes se efectúa mediante un esquema conversacional, donde el modelo recibe un texto de entrada y genera una respuesta correspondiente. Las respuestas y entradas se concatenan secuencialmente para preservar el contexto de la conversación, cada consulta subsiguiente incorpora todos los intercambios precedentes.

- **Uso de herramientas externas:** toda la complejidad de la interacción se abstrae en el framework, por lo que el desarrollador únicamente debe especificar la función que desea incorporar.
- **Interacción entre agentes:** los agentes pueden establecer comunicación entre sí, permitiendo la construcción de sistemas con mayor complejidad. Algunos frameworks establecen protocolos que definen las modalidades de comunicación entre los distintos agentes.

Para este trabajo utilizaremos LangChain para la gestión de llamadas a APIs de modelos y prompting, LangGraph para la orquestación de flujos agénticos, y LangSmith para el seguimiento de trazas de llamadas a modelos y herramientas, así como para la ejecución de las evaluaciones del sistema. Para la gestión de los conjuntos de datos y entrenamiento de modelos utilizaremos HuggingFace.⁵

2.3. Model Context Protocol (MCP)

El protocolo MCP, desarrollado por Anthropic, estandariza la comunicación entre agentes LLM y herramientas. Permite que aplicaciones diversas ofrezcan herramientas a agentes externos sin exponer detalles de implementación [15].

La Figura 2.2 ilustra el esquema operativo del protocolo. En este ejemplo, los desarrolladores de Jira⁶ y GitHub⁷ han implementado un servidor MCP que traduce las interacciones con sus APIs y proporciona herramientas al cliente MCP. Esto permite que el desarrollador del agente se enfoque exclusivamente en conectar las herramientas del cliente con el agente, sin necesidad de interactuar con APIs externas. Asimismo, el cliente MCP gestiona la comunicación entre servidores, facilitando al agente el acceso directo a múltiples herramientas.

El protocolo ofrece dos modos de operación para establecer la comunicación entre cliente y servidor:

- **Comunicación SSE:** el protocolo *Server-Sent Events* (SSE) establece un canal de comunicación unidireccional sobre HTTP desde el servidor hacia el cliente, proporciona actualizaciones en tiempo real con capacidad de streaming. En el protocolo MCP, el cliente efectúa solicitudes para la ejecución de herramientas en el servidor mediante HTTP, a lo que el servidor puede responder mediante eventos SSE.
- **Comunicación STDIO:** el protocolo de entrada y salida estándar (STDIO) facilita la comunicación bidireccional entre cliente y servidor a nivel de proceso en el sistema operativo. Este mecanismo permite el intercambio de información en formato JSON a través de los canales

⁵HuggingFace: <https://huggingface.co/>

⁶Jira: <https://www.atlassian.com/es/software/jira>

⁷GitHub: <https://github.com/>

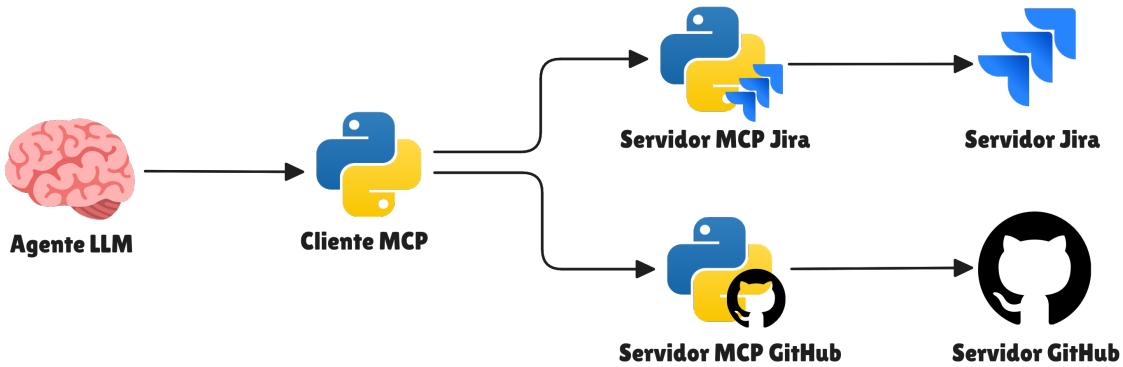


Figura 2.2: Esquema de funcionamiento del Model Context Protocol.

estándar del sistema. Su diseño, orientado principalmente a entornos locales, restringe la conexión a un único cliente por servidor al limitarse a la comunicación entre dos procesos.

La aplicación Claude Desktop⁸ de Anthropic demuestra el potencial del protocolo. Esta plataforma permite integrar servidores MCP de terceros con configuración mínima mediante gestores de paquetes estándar, facilitando el acceso a múltiples herramientas en la interfaz de chat.

2.4. Estado del arte en arquitecturas de agentes LLM

La comunidad científica ha diseñado diversas arquitecturas de agentes para optimizar el rendimiento de los modelos disponibles. La arquitectura RAG se distingue por complementar la entrada del modelo con información recuperada de documentos relevantes. Otras propuestas se centran en mejorar la comunicación y coordinación entre agentes.

2.4.1. Arquitectura RAG

Los modelos LLM poseen un conocimiento restringido a los datos con los que fueron entrenados. Para superar esta limitación, el enfoque RAG (Retrieval-Augmented Generation) complementa la generación del LLM mediante la recuperación de información relevante desde repositorios de conocimiento externos. La Figura 2.3 ilustra un ejemplo de su funcionamiento.

La recuperación de documentos relevantes se puede implementar mediante recuperadores dispersos: expresiones regulares, búsqueda de n-gramas, palabras clave, entre otras. No obstante, el enfoque predominante consiste en el uso de recuperadores densos [8], conocidos como indexación vectorial. En este método, los documentos se transforman en vectores, generalmente mediante LLMs especializados en codificación, denominados *embedders*. Al representar los documentos en

⁸Claude Desktop: <https://claude.ai/download>

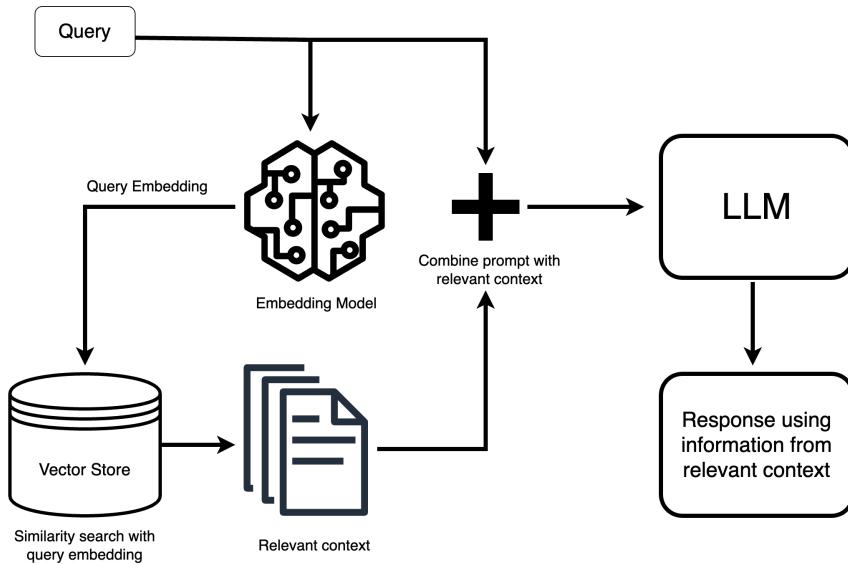


Figura 2.3: Esquema de funcionamiento de la arquitectura RAG en un LLM [Fuente](#).

un espacio vectorial, es posible recuperar aquellos semánticamente más pertinentes mediante la comparación del vector de consulta con los vectores de los documentos indexados, utilizando métricas como la distancia coseno (definida en Anexo A.3).

2.4.1.1. Estrategias RAG avanzadas

La optimización del rendimiento en arquitecturas RAG ha sido ampliamente estudiada [16, 8], enfocándose en tres áreas principales: el procesado de documentos, los sistemas de recuperación y la mejora del flujo de generación.

Dentro del procesado de documentos, la ventana solapada (Figura 2.4a) superpone fragmentos de texto al dividir documentos largos en segmentos indexables para evitar pérdida de información en los límites entre fragmentos. En cuanto a la mejora del flujo de generación, los sistemas de salto múltiple (Figura 2.4b) alternan entre recuperación y generación para abordar tareas complejas [17, 18, 19, 20, 21].

2.4.2. Arquitecturas de interacción entre agentes

La interacción entre agentes LLM constituye un campo de investigación activo, distinguiéndose diversos avances en módulos de memoria, planificación e interacción multiagente [22].

- **Módulos de memoria:** en una interacción conversacional, el modelo procesa todos los mensajes previos, pudiendo generar un contexto excesivamente amplio. Para mitigar este problema, se han desarrollados módulos de memoria que almacenan información relevante

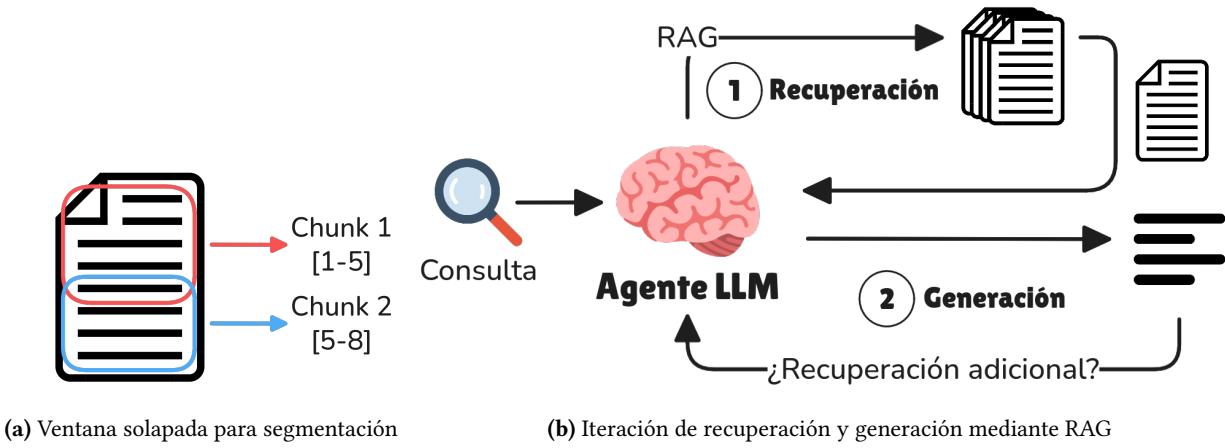


Figura 2.4: Técnicas avanzadas de RAG

de interacciones pasadas de forma resumida [23, 24, 25]. Esta memoria puede consultarse posteriormente mediante RAG, permitiendo recuperar los elementos más relevantes según el contexto [26].

Algunos módulos se inspiran en la estructura de la memoria humana [27], incorporando mecanismos que almacenan información con distintas temporalidades y niveles de relevancia [22, 28].

- **Planificación:** los mecanismos de planificación potencian el razonamiento de los agentes sobre sus acciones futuras.

Entre estos mecanismos destaca el prompting de cadena de pensamiento (Chain of Thought) [29], que instruye al modelo para elaborar un razonamiento secuencial previo a su decisión final, permitiendo así descomponer problemas complejos paso a paso. Partiendo de este enfoque, estrategias avanzadas como la autoconsciencia [25] lo amplían mediante la generación de múltiples cadenas de razonamiento independientes y la posterior selección de la respuesta óptima entre ellas [30, 31]. De manera complementaria, las técnicas de reflexión [32, 33, 34] implementan un proceso iterativo donde el propio modelo evalúa y refina sus respuestas.

Por otro lado, la estructuración de acciones constituye una metodología ampliamente adoptada en la planificación [35, 36, 37]. Esta técnica permite definir planes de alto nivel que posteriormente se desglosan en acciones específicas ejecutables por los agentes [38, 39, 40, 41]. Adicionalmente, la definición de interdependencias entre estas acciones permite verificar la validez de los planes generados [42, 43, 44].

Los modelos razonadores como o1 de OpenAI o DeepSeek-R1 incorporan estas técnicas de

forma nativa [45]. Estos han sido entrenados con datos que incluyen ejemplos de razonamiento y planificación, lo que les permite generar respuestas que siguen dichas estructuras.

- **Interacción entre agentes:** los sistemas multiagente implementan una arquitectura donde diversos agentes especializados son coordinados por un componente central denominado orquestador [46, 47]. En este paradigma, cada agente se especializa en una función particular, como la búsqueda de información, la ejecución de herramientas o la generación de texto. El orquestador evalúa las consultas entrantes y las dirige hacia el agente más competente para resolverlas.

Enfoques complementarios proponen la interacción directa entre agentes especializados como mecanismo de retroalimentación [48, 49]. Por ejemplo, ChatDev [5] establece un sistema de colaboración entre agentes programadores, testers y gestores para abordar problemas de ingeniería de software. MetaGPT [50] refina esta propuesta al implementar un protocolo de comunicación basado en el patrón publicador/suscriptor entre los agentes, permitiéndoles difundir información de forma selectiva.

2.4.3. Evaluación de agentes

La evaluación del rendimiento de agentes permite comparar su eficacia en diferentes dominios y configuraciones. Se han desarrollado diversas estrategias que abordan desde la calidad de las respuestas hasta comportamientos específicos como la selección de herramientas.

Las métricas de dominio constituyen el enfoque más directo, utilizando bancos de pruebas con preguntas y respuestas anotadas para comparar programáticamente los resultados obtenidos [51, 52, 53]. Complementariamente, es posible evaluar la correcta utilización de herramientas mediante conjuntos de datos específicos como ToolLLM [54], que anota qué herramientas son necesarias en cada caso para determinar si el agente evaluado realiza las llamadas apropiadas [55, 56].

LangSmith proporciona un marco de evaluación personalizable que permite definir métricas específicas y conjuntos de datos orientados al uso de agentes. Esta plataforma ofrece una interfaz web que facilita la ejecución y visualización de evaluaciones, habiendo sido objeto de estudio anteriormente en el TFG de Mikel Lonbide en LKS Next.

2.5. Agentes LLM en proyectos software

La integración de agentes en proyectos software ha sido objeto de estudio enfocándose principalmente en el ámbito de la generación automática de código. Se pueden destacar tres niveles de autonomía en los productos desarrollados: sugerencias de autocompletado, asistentes de programación y agentes autónomos.

Sugerencias de autocompletado: GitHub Copilot⁹ o Cursor¹⁰ ofrecen sugerencias de código en tiempo real integradas directamente en el entorno de desarrollo. Estas herramientas han demostrado eficacia para aumentar la productividad en tareas repetitivas [57], aunque presentan limitaciones en razonamiento profundo debido a su enfoque en rapidez de respuesta.

Asistentes de programación: herramientas como GitHub Copilot Chat, Cursor o Cody¹¹ de SourceGraph¹² proporcionan interfaces de chat interactivas para consultas específicas sobre código [58]. SourceGraph implementa un sistema que combina análisis estructural del código mediante grafos de dependencias con mecanismos RAG para generar respuestas contextualizadas [59, 60].

Agentes autónomos: soluciones como Aider¹³ o DevinAI¹⁴ buscan automatizar el ciclo completo de desarrollo, aunque permanecen en fase experimental debido a su complejidad [6]. Aider utiliza Tree-sitter¹⁵ para analizar código fuente y generar grafos de dependencias, implementando algoritmos de clasificación para determinar la relevancia contextual [61].

2.6. Ajuste de modelos para agentes LLM

Los modelos instruct del estado del arte, aunque capaces de resolver tareas agénticas, presentan limitaciones significativas para aplicaciones especializadas. Su gran tamaño (cientos de miles de millones de parámetros) los hace costosos computacionalmente [62], mientras que su acceso únicamente vía API plantea preocupaciones de privacidad al requerir envío de datos a servidores externos, especialmente relevante en desarrollo de software con información sensible.

Para abordar estas limitaciones, diversos enfoques como FireAct [63], AgentTuning [64] y Agent-FLAN [65] proponen el ajuste fino de modelos menores mediante destilación. Esta técnica utiliza un modelo grande para generar datos sintéticos de entrenamiento, que posteriormente se emplean para especializar modelos más pequeños en tareas específicas, obteniendo precisión similar con una fracción del costo computacional.

⁹GitHub Copilot: <https://github.com/features/copilot>

¹⁰Cursor: <https://www.cursor.com/>

¹¹Cody: <https://sourcegraph.com/cody>

¹²SourceGraph: <https://sourcegraph.com/>

¹³Aider: <https://aider.chat/>

¹⁴DevinAI: <https://devin.ai/>

¹⁵Tree-sitter: <https://tree-sitter.github.io/tree-sitter/>

3

CAPÍTULO

Planificación

En este capítulo se presenta la planificación del proyecto, abordando el alcance definido, los períodos de realización de tareas y los diversos ámbitos de gestión: temporal, de riesgos, de comunicaciones e información, y de partes interesadas. El objetivo de esta planificación es establecer una hoja de ruta estructurada que permita el cumplimiento de todos los objetivos del proyecto dentro de los plazos establecidos.

3.1. Alcance

Este proyecto aborda el desarrollo de un sistema basado en agentes LLM implementado sobre una solución software propia de LKS Next, el cual deberá responder a consultas formuladas en lenguaje natural que un desarrollador recién incorporado podría plantear durante su proceso de integración al proyecto. Con ello se pretende investigar el comportamiento de diversas arquitecturas de agentes definidas en el Capítulo 2 y evaluar su viabilidad para su futura incorporación en los procesos productivos de la empresa.

En dicho sistema, se implementarán diversas modalidades de interacción entre agentes especializados en fuentes de datos específicas, evaluando la eficacia de los distintos patrones de comunicación entre dichos componentes.

Cabe destacar que el alcance del proyecto no está definido en su totalidad, dada la complejidad de estimación inherente a su naturaleza exploratoria y al uso de tecnologías emergentes. Para mitigar riesgos en el desarrollo, se ha establecido un protocolo preventivo que contempla reuniones quincenales de seguimiento y control.

3.1.1. Objetivos concretos del proyecto

Para facilitar el logro del objetivo principal, se han identificado y definido los siguientes objetivos específicos que estructuran el avance progresivo del proyecto:

- **Estudio de arquitecturas agénticas:** realizar un análisis de las diversas arquitecturas de agentes, considerando distintas estrategias de interacción y mecanismos de acceso a fuentes de información.
- **Desarrollo de sistema de onboarding:** implementar un prototipo que simule una aplicación integrada en la empresa como parte de una iniciativa organizacional más amplia en este ámbito, aplicando las arquitecturas propuestas sobre un proyecto software corporativo. El objetivo consiste en analizar su eficacia en la asistencia a nuevos integrantes durante su proceso de incorporación a la empresa.
- **Integración del Model Context Protocol:** exploración de las características y beneficios que aporta la implementación del protocolo MCP, con el objetivo de realizar una valoración objetiva para su posible integración en el entorno profesional de la empresa.
- **Evaluación de agentes:** desarrollar un sistema de evaluación para proporcionar métricas comparativas cuantificables sobre el rendimiento de los diferentes enfoques de agentes.
- **Valoración de ajuste de agentes:** analizar la relación coste-beneficio asociada al proceso de ajuste fino de modelos LLM para su aplicación en agentes concretos.

Adicionalmente, el proyecto contempla desarrollar el sistema de onboarding incorporando en la medida de lo posible en su base de conocimiento la metodología de trabajo implementada en la empresa. Mediante la adhesión a los estándares definidos en un entorno profesional real, se pretende garantizar que los resultados obtenidos constituyan un reflejo de la viabilidad de implementación y eficacia de proyectos similares en un sistema de explotación.

3.1.2. Fases del proyecto

Tal y como se ha mencionado anteriormente, el alcance del proyecto no está completamente definido debido a su naturaleza exploratoria. Consecuentemente, se propone un ciclo de vida iterativo-incremental con iteraciones de aproximadamente dos semanas de duración, permitiendo una adaptación progresiva a los requisitos emergentes.

La primera iteración se centra en la captura de requisitos del proyecto, donde se explorará y definirá el alcance del sistema de agentes a desarrollar. Complementariamente, se realizará un estudio de las arquitecturas de agentes más relevantes para determinar los enfoques técnicos óptimos para la implementación. Tras establecer estas bases, la segunda iteración abordará la implementación de un sistema de agentes mínimo que contenga la estructura general del sistema, proporcionando un marco operativo inicial.

Con este sistema mínimo implementado, la tercera iteración corresponderá al desarrollo de un mecanismo de evaluación, con el objetivo de establecer métricas que permitan mejorar el sistema en la cuarta iteración, donde se aplicarán las optimizaciones identificadas.

Finalmente, la quinta iteración se dedicará a la implementación de mejoras sobre el sistema base. Se contemplan tanto arquitecturas de agentes adicionales como el ajuste fino de un modelo LLM para su integración en el flujo agéntico.

Gracias a la implementación del ciclo de vida iterativo, se logra la consecución de los objetivos del proyecto de forma progresiva y estructurada. Este enfoque permite obtener retroalimentación constante por parte de los directores del proyecto durante cada iteración, facilitando así la posibilidad de reorientar la dirección del trabajo ante la aparición de posibles contratiempos.

3.1.3. Descomposición de tareas

La Estructura de Descomposición de Trabajo (EDT) se ha diseñado considerando el ciclo de vida iterativo-incremental adoptado. La Figura 3.1 ilustra un diagrama de los paquetes de trabajo, seguido de la explicación detallada de las tareas individuales de cada paquete.

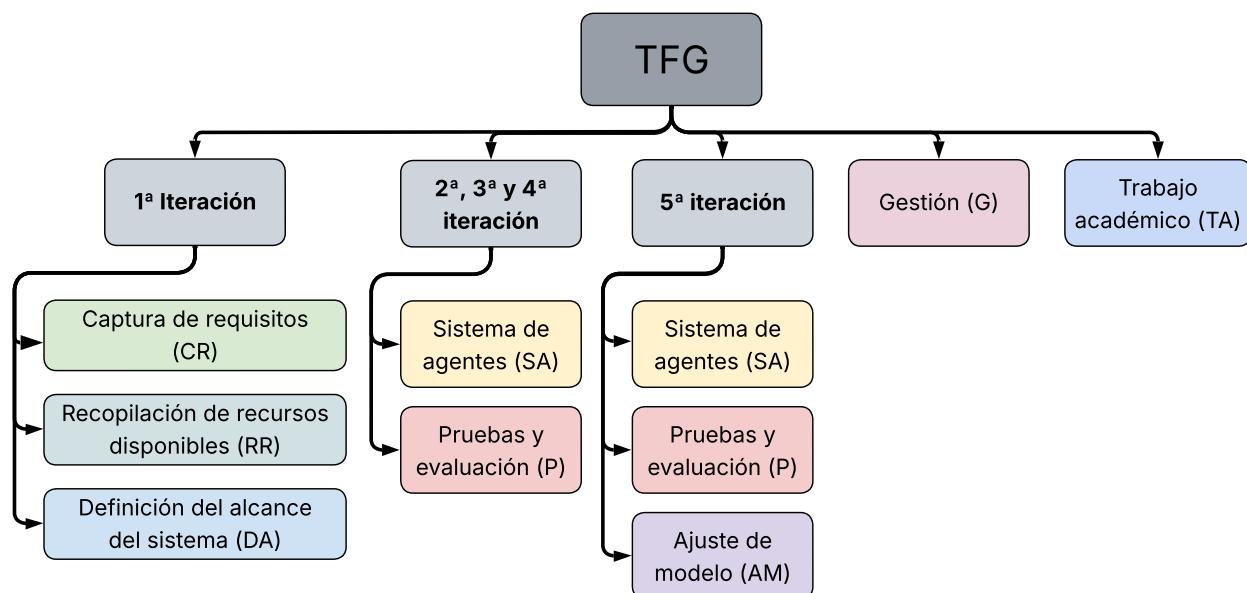


Figura 3.1: Estructura de Descomposición de Trabajo (EDT) del proyecto

- **1ª iteración - Definición de alcance, requisitos y bases del proyecto:** establecimiento de los fundamentos del proyecto.
 - **Captura de requisitos (CR):** actividades orientadas a la definición de los requisitos necesarios para el desarrollo del proyecto.

- **Definición de requisitos del sistema:** identificación y documentación de los requisitos funcionales y no funcionales que debe cumplir el proyecto, validados con los directores al inicio del mismo.
 - **Elicitación de requisitos del dominio:** recopilación de las tipologías de consultas que el sistema debe resolver mediante un cuestionario electrónico dirigido al personal técnico para definir el alcance funcional del sistema.
 - **Recopilación de recursos disponibles (RR):** actividades centradas en la selección y generación de recursos esenciales para el desarrollo.
 - **Selección de proyecto software:** identificación y documentación del proyecto software sobre el que se desarrollará el sistema.
 - **Generación de recursos:** elaboración de documentación complementaria para la implementación del sistema de agentes.
 - **Definición del alcance del sistema de agentes (DA):** delimitación del alcance considerando implementaciones previas y trabajo académico existente.
 - **Investigación de arquitecturas del estado del arte:** análisis de las arquitecturas relevantes documentadas en la literatura académica.
 - **Exploración de proyectos similares:** estudio de implementaciones similares en proyectos software y procesos de onboarding.
- **2^a iteración - Implementación del sistema base:** desarrollo del núcleo funcional del sistema con las capacidades mínimas necesarias.
- **Sistema de agentes (SA):** actividades centradas en el diseño e implementación de un sistema de agentes LLM para asistencia en proyectos software.
 - **Diseño del sistema:** conceptualización e implementación básica de los módulos fundamentales del sistema.
 - **Implementación de agentes especializados:** desarrollo de agentes adaptados a las diversas fuentes de información disponibles.
 - **Sistema de comunicación mínima:** creación de un mecanismo básico de orquestación para los agentes implementados.
 - **Pruebas y evaluación (P):** actividades destinadas a verificar y analizar el rendimiento de los módulos implementados.
 - **Pruebas automatizadas:** desarrollo de pruebas unitarias para algoritmos críticos de los agentes especializados.
 - **Integración continua:** implementación de un flujo de trabajo automatizado para la ejecución de pruebas unitarias.
- **3^a iteración - Desarrollo de capacidades de evaluación:** establecimiento de mecanismos de medición y evaluación para cuantificar el rendimiento y guiar las mejoras posteriores del sistema.

- **Sistema de agentes (SA):** actividades centradas en el diseño e implementación de un sistema de agentes LLM para asistencia en proyectos software.
 - **Sistema de evaluación:** implementación de mecanismos de evaluación automática sobre el sistema mínimo.
 - **Captura de datos de evaluación:** anotación manual de ejemplos representativos para evaluar el rendimiento del sistema.
- **Pruebas y evaluación (P):** actividades destinadas a verificar y analizar el rendimiento de los módulos implementados.
 - **Evaluación del sistema mínimo:** ejecución de evaluaciones automatizadas e identificación de elementos clave para mejoras posteriores.
- **4^a iteración - Optimización y refinamiento:** mejora del sistema existente mediante la aplicación de optimizaciones identificadas a través de la evaluación previa.
 - **Sistema de agentes (SA):** actividades centradas en el diseño, implementación y evaluación de un sistema de agentes LLM para asistencia en proyectos software.
 - **Mejora de agentes:** refinamiento de los agentes implementados para optimizar su rendimiento según las métricas establecidas.
 - **Variaciones de orquestación:** análisis e implementación de estrategias alternativas de orquestación.
 - **Pruebas y evaluación (P):** actividades destinadas a verificar y analizar el rendimiento de los módulos implementados.
 - **Evaluación comparativa:** ejecución de evaluaciones automatizadas y análisis comparativo respecto a la iteración anterior.
- **5^a iteración - Exploración de mejoras:** implementación de técnicas especializadas y arquitecturas alternativas para maximizar las capacidades del sistema.
 - **Sistema de agentes (SA):** actividades centradas en el diseño e implementación de un sistema de agentes LLM para asistencia en proyectos software.
 - **Arquitecturas de interacción alternativas:** implementación y evaluación de mecanismos adicionales de interacción entre agentes.
 - **Módulos de memoria:** integración de sistemas de memoria y evaluación de su impacto en el rendimiento global.
 - **Agentes avanzados:** desarrollo de un agente con un proceso de ejecución extenso para el análisis del coste-beneficio.
 - **Incorporación del modelo ajustado:** desarrollo de adaptadores e integración del modelo en el sistema existente.

- **Ajuste de modelo (AM):** actividades orientadas al entrenamiento especializado de un modelo para un agente específico.
 - **Selección del agente:** identificación justificada del agente candidato para el ajuste del modelo.
 - **Extracción de datos:** recopilación automatizada de datos de entrenamiento.
 - **Entrenamiento del modelo:** diseño y ejecución del ciclo de entrenamiento del modelo LLM seleccionado.
 - **Pruebas y evaluación (P):** actividades destinadas a verificar y analizar el rendimiento de los módulos implementados.
 - **Evaluación comparativa:** análisis del rendimiento de las nuevas características respecto al sistema precedente.
- **Gestión (G):**
- **Planificación (PL):** establecimiento de directrices, objetivos y actividades para el desarrollo óptimo del proyecto.
 - **Seguimiento y control (SC):** monitorización periódica mediante reuniones bisemanales con los directores del proyecto.
- **Trabajo académico (TA):**
- **Memoria (M):** elaboración del documento académico que recoge todo el trabajo realizado.
 - **Defensa (D):** preparación de la presentación y defensa del proyecto ante el tribunal evaluador.

3.2. Periodos de realización de tareas e hitos

En este apartado se detallan las dependencias entre las diferentes tareas, así como la estimación de duración y fechas de cada una de ellas.

3.2.1. Dependencias entre tareas

Las dependencias de los paquetes de trabajo del proyecto requieren una ejecución secuencial planificada. La Figura 3.2 ilustra dichas dependencias.

El proyecto se inicia con la elaboración de la planificación general, estableciendo el alcance del proyecto, fechas y recursos horarios disponibles. La primera iteración aborda la captura de requisitos del proyecto (CR), que define los requisitos del sistema (DRP) y procede posteriormente con los paquetes DA, ER y RR.

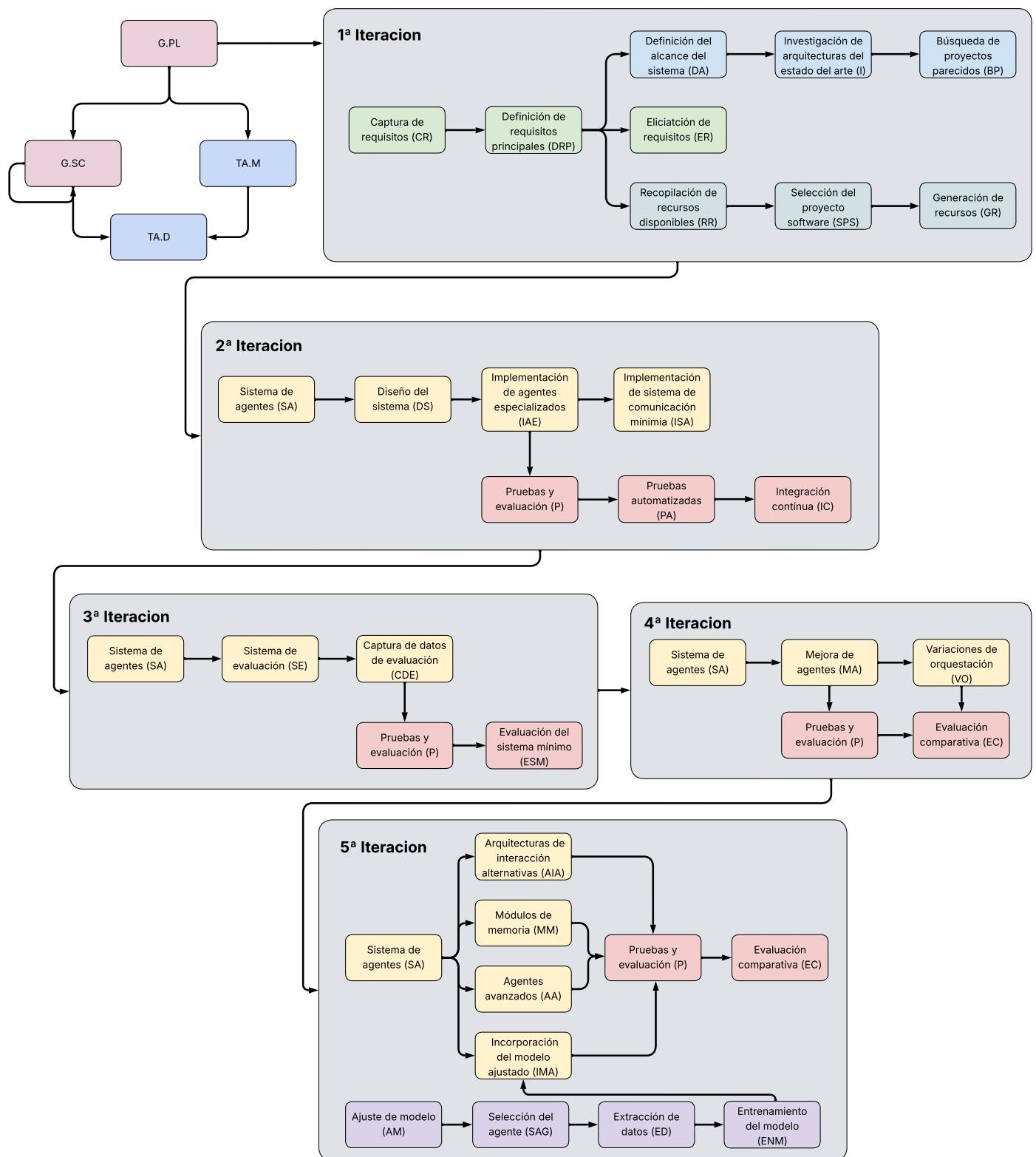


Figura 3.2: Dependencias entre tareas del proyecto

La segunda iteración desarrolla el sistema base de agentes (SA), comenzando por el diseño (DS) para continuar con la implementación (IAE, ISA). Una vez implementado, se ejecutan las pruebas automatizadas (PA) y su integración continua (IC).

Las iteraciones tercera y cuarta mantienen una estructura similar, comenzando con la implementación (SA) seguida de pruebas (P). La tercera iteración se centra en el sistema de evaluación, desarrollando primero el mecanismo (SE) para posteriormente anotar los datos de evaluación correspondientes (CDE). La cuarta iteración realiza inicialmente la mejora de agentes (MA) y posteriormente las variaciones de orquestación (VO), manteniendo este orden secuencial para evitar que las mejoras de agentes individuales introduzcan sesgos en los resultados de las variaciones de orquestación. Es importante completar la tercera iteración antes de iniciar la cuarta, ya que modificar los datos de evaluación comprometería la objetividad del análisis.

La quinta iteración comprende cuatro tareas independientes (AIA, MM, AA, AM) que se evalúan individualmente (P) antes de realizar la comparación con las demás versiones del sistema (EC).

Paralelamente, tras finalizar la planificación inicial se inician el seguimiento y control (SC) y la redacción de la memoria (M). Una vez completada la memoria y validado el proyecto durante el proceso de seguimiento y control, se procede con la preparación de la defensa (D).

3.2.2. Diagrama de Gantt

La Figura 3.3 muestra el diagrama de Gantt, donde se visualiza de manera aproximada la distribución temporal a cada paquete de trabajo durante el transcurso del proyecto. Los rombos negros señalan los hitos clave que se detallan en la Sección 3.2.3.

El proyecto se inicia con el paquete de planificación, seguido de la activación del seguimiento y control del proyecto junto con la primera iteración.

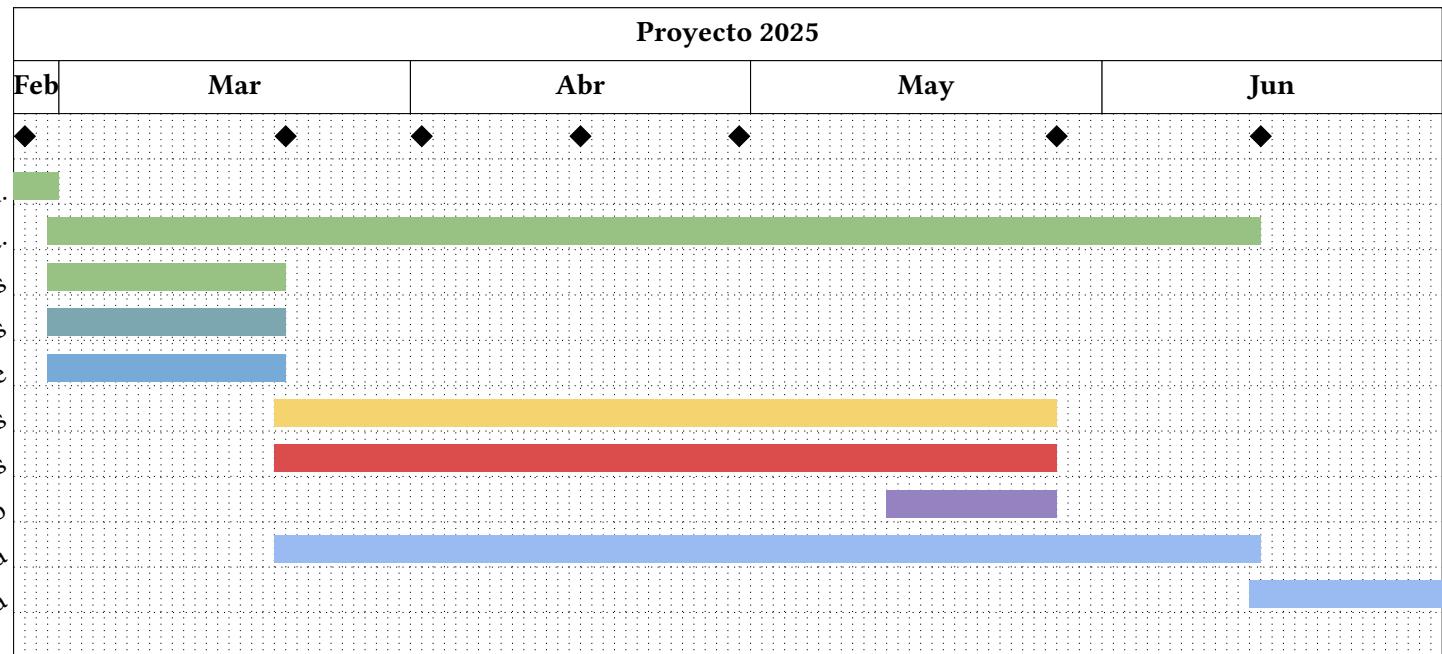
A partir de la segunda iteración se desarrollan los paquetes del sistema de agentes y evaluación, que se extienden hasta la quinta iteración. Paralelamente se inicia la redacción de la memoria, al disponerse de material suficiente para los primeros capítulos.

El paquete de ajuste de modelo está planificado para la segunda mitad de la quinta iteración, correspondiente a las últimas dos semanas de implementación. Finalmente, se dedican dos semanas exclusivamente a la memoria, seguidas de la preparación de la defensa.

Esta distribución temporal permite enfocar los esfuerzos según las prioridades de cada fase: planificación y definición del alcance inicialmente, implementación del sistema en la fase intermedia, y trabajo académico al final, manteniendo una redacción continua de la memoria a lo largo del desarrollo.

3.2.3. Hitos

En la Tabla 3.1 se detallan los hitos establecidos para el desarrollo del proyecto. Estos hitos intermedios permiten evaluar el progreso alcanzado y ajustar el alcance en caso necesario, aspecto

**Figura 3.3:** Diagrama de Gantt del proyecto

fundamental dada la incertidumbre inherente al carácter exploratorio del proyecto. La finalización de la fase de implementación ha sido programada para el 31 de mayo, tras lo cual se destinarán dos semanas íntegramente a la elaboración de la memoria hasta el 14 de junio, proporcionando así un margen de 8 días previos a la fecha límite de entrega. Este período de contingencia está planificado para abordar posibles contratiempos que pudieran surgir durante el transcurso del proyecto.

Hito	Fecha
Inicio del proyecto	25/02/2025
Fin Iteración 1	20/03/2025
Fin Iteración 2	01/04/2025
Fin Iteración 3	15/04/2025
Fin Iteración 4	29/04/2025
Fin Iteración 5	27/05/2025
Fin memoria	14/06/2025
Defensa del proyecto	Por determinar

Tabla 3.1: Cronograma de Hitos del Proyecto

3.3. Gestión del tiempo

Se ha gestionado el tiempo disponible para el proyecto considerando el alcance definido para cumplir todos los objetivos del proyecto.

3.3.1. Estimación de cada tarea

La estimación específica de cada tarea se puede ver en la Tabla 3.4

Se ha asignado un tiempo considerable a la primera iteración para definir el alcance y requisitos del sistema (35 horas). Dado el carácter exploratorio del proyecto y su alcance inicialmente ambiguo, resulta indispensable establecer un enfoque adecuadamente fundamentado para evitar rectificaciones en fases posteriores.

La segunda iteración requiere una estimación de 63 horas al comprender el desarrollo de un sistema base completo. El sistema de evaluación se ha estimado en 30 horas, mientras que la mejora de agentes individuales y las variaciones de orquestación se han cuantificado en 15 horas cada una, dado que únicamente requieren modificaciones sobre los agentes existentes.

Se ha destinado el tiempo restante (95 horas) a las mejoras del sistema, con estimaciones preliminares para cada componente. La gestión del proyecto se ha cuantificado en 30 horas totales, suficientes para las reuniones bisemanales y el desarrollo de la planificación. La redacción de la memoria se ha estimado en 60 horas conforme a las recomendaciones institucionales, y la preparación de la defensa en 10 horas.

3.4. Gestión de riesgos

- **R1- Concurrencia exploratoria:** dado que el proyecto está enfocado en tecnologías emergentes, existe la posibilidad de que durante el período de desarrollo emergan iniciativas paralelas que aborden la misma problemática.
 - **Prevención:** investigación de soluciones existentes antes de implementar cada módulo del sistema.
 - **Contingencia:** convocatoria de reunión de seguimiento y control con los directores del proyecto para evaluar el estado del desarrollo y decidir si mantener la solución actual o reorientar el enfoque.
- **R2- Variabilidad del alcance:** al ser un proyecto exploratorio con un alcance inicialmente ambiguo, podría sufrir alteraciones no planificadas que ocasionen un sobrecoste horario y obliguen a replanificar el alcance para no sobrepasar los recursos disponibles.
 - **Prevención:** implementación de seguimiento y control mediante reuniones quincenales para evaluar la correcta evolución de las actividades.

- **Contingencia:** reajuste del alcance y replanificación temporal en caso necesario para garantizar la viabilidad del trabajo dentro de los plazos establecidos.
- **R3- Dependencia de sistemas externos:** la implementación del proyecto depende significativamente de sistemas externos (modelos LLM via APIs y servidores MCP). Cualquier alteración o interrupción en estos servicios podría comprometer la funcionalidad del sistema implementado.
 - **Prevención:** diseño de arquitectura modular y sistema con manejo robusto de excepciones que garantice la continuidad operativa.
 - **Contingencia:** integración de servicios alternativos o desacoplamiento temporal de componentes afectados según la necesidad.
- **R4- Filtrado de credenciales:** el acceso a recursos externos requiere claves secretas susceptibles de ser detectadas por algoritmos de rastreo automatizados en plataformas públicas. La exposición inadvertida podría ocasionar pérdidas económicas o comprometer la seguridad corporativa.
 - **Prevención:** gestión de credenciales mediante variables de entorno, evitando su inclusión en código fuente, y mantenimiento de repositorios en modo privado como medida adicional de protección.
 - **Contingencia:** revocación de credenciales comprometidas y regeneración de claves de acceso.
- **R5- Pérdida de recursos:** el desarrollo se fundamenta en múltiples recursos esenciales (código fuente, documentación, artefactos y memoria académica). La pérdida de cualquiera debido a fallos técnicos podría ocasionar retrasos significativos.
 - **Prevención:** implementación de sistemas de respaldo automático con copias de seguridad actualizadas diariamente en la nube según se describe en la Sección 3.6.1.
 - **Contingencia:** restauración de la información desde las copias de seguridad en la nube, limitando la pérdida máxima al trabajo de una jornada.

3.5. Gestión de interesados

El desarrollo del presente trabajo comprende los siguientes interesados:

- **Autor:** cuya graduación universitaria depende de la consecución exitosa del proyecto.
- **Directores:** se consideran los dos directores académicos y el director en la empresa, para quienes la supervisión de esta iniciativa constituye parte de sus responsabilidades profesionales.

- **LKS Next:** organización que financia parte del desarrollo y se beneficiará mediante la propiedad intelectual generada, así como por la contribución a sus objetivos de innovación en soluciones de IA aplicada.
- **Universidad UPV/EHU:** institución académica responsable del desarrollo. Se beneficia como parte del conjunto de trabajos de investigación aplicada que contribuyen a su misión formativa e investigadora.
- **Tribunal evaluador:** conjunto de profesores responsables de la evaluación académica del trabajo, cuyo interés se centra en la comprensibilidad y organización del trabajo para optimizar el proceso evaluativo.

3.6. Gestión de Comunicaciones e Información

La gestión eficaz de la información y comunicación constituye un elemento fundamental para garantizar el desarrollo exitoso del proyecto.

3.6.1. Sistema de información

La gestión de la información del proyecto se ha estructurado mediante los siguientes sistemas tecnológicos:

- **Repositorio GitHub para código fuente:** el código desarrollado será alojado en un repositorio privado de GitHub.
- **Repositorio GitHub para documentación:** la memoria del proyecto, elaborada utilizando LaTeX en el entorno local del alumno, será sincronizada con un repositorio dedicado en GitHub.
- **Almacenamiento en Google Drive¹:** los diversos recursos y materiales auxiliares recopilados durante las fases de desarrollo serán almacenados en esta plataforma.

Esta estructura de gestión de la información aporta diversos beneficios al proyecto. Por un lado, facilita la supervisión continua por parte de los directores e implementa un control de versiones organizado que documenta la evolución del trabajo. Por otro lado, garantiza copias de seguridad actualizadas que protegen la integridad de los datos. Adicionalmente, asegura la accesibilidad para todos los interesados.

¹Google Drive: <https://workspace.google.com/intl/es/products/drive/>

3.6.2. Sistema de comunicación

La comunicación eficaz entre alumno, director en la empresa y directores académicos resulta imprescindible para el correcto seguimiento y control del proyecto. Las herramientas a utilizar son:

- **Correo electrónico:** canal principal para consultas con los directores del proyecto y comunicación formal con otros miembros de la empresa.
- **Google Meet²:** plataforma destinada a la realización de reuniones telemáticas entre los participantes.
- **Google Chat³:** herramienta complementaria para resolución de consultas rápidas con el director en la empresa.

3.7. Herramientas disponibles

Para el desarrollo del proyecto se dispone de varias herramientas que facilitan su implementación y gestión eficiente:

- **IDE PyCharm⁴:** se usará la versión Professional del entorno de desarrollo PyCharm, obtenida a través del paquete educativo de GitHub, que proporciona herramientas especializadas para el desarrollo en Python.
- **Asistencia de herramientas de inteligencia artificial:** se dispone de herramientas de asistencia basadas en inteligencia artificial como GitHub Copilot y una suscripción a la aplicación ClaudeAI⁵, que optimizan las tareas de programación, el procesamiento documental y la redacción de la presente memoria.
- **Claves de acceso a modelos:** la empresa LKS Next ha facilitado las credenciales de acceso necesarias para la integración y ejecución de los diferentes modelos LLM utilizados en el proyecto.
- **Plataformas de computación en la nube:** se dispone de acceso a infraestructuras de computación en la nube con unidades de procesamiento gráfico (GPU) dedicadas para el entrenamiento e inferencia del modelo ajustado.

²Google Meet: <https://meet.google.com>

³Google Chat: <https://mail.google.com/chat>

⁴PyCharm: <https://www.jetbrains.com/es-es/pycharm/>

⁵ClaudeAI: <https://claude.ai>

Iteración	Paquete de trabajo	Tarea	Dedicación estimada (h)
1 ^a Iteración	Captura de requisitos(CR)	Definición de requisitos principales	7
	Recopilación de recursos disponibles (RR)	Elicitación de requisitos	3
	Definición del alcance del sistema de agentes (DA)	Selección del proyecto software	5
		Generación de recursos	5
		Investigación de arquitecturas del estado del arte	10
		Búsqueda de proyectos parecidos	5
SUBTOTAL			35
2 ^a Iteración	Sistema de agentes (SA)	Diseño del sistema	10
	Pruebas y evaluación (P)	Implementación de agentes especializados	35
		Implementación de sistema de comunicación mínima	10
		Pruebas automatizadas	5
		Integración continua	3
		SUBTOTAL	63
3 ^a Iteración	Sistema de agentes (SA)	Sistema de evaluación	20
	Pruebas y evaluación (P)	Captura de datos de evaluación	5
		Evaluación del sistema mínimo	5
		SUBTOTAL	30
4 ^a Iteración	Sistema de agentes (SA)	Mejora de agentes	15
	Pruebas y evaluación (P)	Variaciones de orquestación	15
		Evaluación comparativa	5
		SUBTOTAL	35
5 ^a Iteración	Sistema de agentes (SA)	Arquitecturas de interacción alternativas	15
	Pruebas y evaluación (P)	Módulos de memoria	15
		Agentes RAG avanzados	15
		Ajuste de modelo	40
		Evaluación comparativa	10
		SUBTOTAL	95
	Gestión (G)	Planificación (PL)	15
		Seguimiento y control (SC)	15
		SUBTOTAL	30
	Trabajo académico (T)	Memoria (M)	60
		Defensa (D)	10
		SUBTOTAL	70
Horas totales:			358

Figura 3.4: Estimación horaria de cada tarea

CAPÍTULO 4

Captura de requisitos

Una vez asentados los conceptos con los que se trabajará en el Capítulo 2 y definidos los objetivos del proyecto en el Capítulo 3, en el presente capítulo se detallarán los requisitos establecidos para el desarrollo del sistema.

La captura de requisitos se ha estructurado en tres fases: en primer lugar, la definición de los requisitos principales y el alcance general con los directores del proyecto; en segundo lugar, la identificación de los dominios de conocimiento en los que el sistema debe especializarse; y finalmente, la determinación de los recursos específicos a utilizar para la implementación de la solución.

4.1. Requisitos principales

El sistema agéntico implementado debe tener las siguientes características para cumplir con las necesidades del director en la empresa:

4.1.1. Requisitos funcionales

- **Agentes especializados:** el sistema contemplará un mínimo de 4 agentes cuya labor esté especializada en responder preguntas sobre fuentes de datos específicas.
- **Fuentes de información:** los agentes del sistema dispondrán de acceso a un conjunto de recursos que incluirá el repositorio de código del proyecto software, una documentación externa y un sistema de gestión de tareas. Al menos una de estas fuentes deberá ser accedida mediante una implementación RAG.

- **Orquestación:** se requiere un agente coordinador cuya función sea decidir a qué agente especializado delegar una acción a realizar.
- **Protocolo MCP:** la implementación priorizará el uso del protocolo MCP: al menos uno de los agentes especializados empleará dicho protocolo para obtener las herramientas a utilizar.
- **Evaluación de agentes:** se implementará un sistema de evaluación objetivo para determinar si la mejora de agentes incrementa la precisión del sistema. Dado un agente y su mejora, será posible determinar cuál de los dos responde mejor a las preguntas realizadas, utilizando para ello una métrica cuantificable.
- **Enfoque del sistema:** la finalidad principal del sistema consistirá en responder preguntas sobre un proyecto software, logrando una tasa de precisión mínima del 75 % sobre un conjunto predefinido de no menos de 25 preguntas, aprobado por los directores del proyecto. Esta precisión se medirá utilizando una métrica cuantificable previamente establecida.
- **Librerías utilizadas:** el desarrollo se sustentará en tres librerías: LangChain para el control de modelos y gestión de prompts (empleada en al menos el 75 % de las ejecuciones), LangGraph para estructurar el flujo lógico (requiriendo que mínimo tres cuartas partes de los agentes operen sobre un grafo compilado de dicha librería), y LangSmith como herramienta de monitorización (garantizando la visualización de todas las llamadas a modelos en la plataforma).

Cabe destacar que los requisitos iniciales del director en la empresa contemplaban únicamente una evaluación manual del sistema. La implementación de un sistema de evaluación automatizada, junto con el establecimiento de una precisión mínima cuantificable, se incorporaron posteriormente en la planificación al identificarse como mejoras significativas para el proyecto.

4.1.2. Requisitos no funcionales

- **Control de excepciones:** el sistema incorporará un control de excepciones que garantice la ejecución aún si un agente o una herramienta genera una excepción. Se contempla la captura de excepciones de todos los agentes y todas las herramientas utilizadas.
- **Control del tiempo:** la ejecución completa del sistema para un caso de uso no excederá los 5 minutos.
- **Control del presupuesto:** se establece un límite máximo de 25 céntimos por ejecución de un caso de uso para el coste de API de modelos. El director en la empresa no especificó una cantidad concreta, sino que instó al uso razonable de los recursos; por ello, se ha elegido esta cifra como límite que no debe sobrepasarse.

- **Independencia de modelos:** la arquitectura garantizará independencia respecto a un único proveedor de modelos LLM. La sustitución de los modelos utilizados será posible en menos de 5 minutos.
- **Modularidad del sistema:** el orquestador funcionará de manera desacoplada respecto a los agentes especializados. La adición o eliminación de un agente especializado requerirá menos de 5 minutos de desarrollo.

4.2. Dominios de conocimiento

De acuerdo con los objetivos previamente establecidos en la Sección 3.1.1, es necesario alinear el sistema con la metodología de desarrollo implementada en la organización. Con este propósito, se ha efectuado una recopilación de las posibles interrogantes que una nueva incorporación plantearía al sistema durante su integración a un proyecto software, procediendo posteriormente a realizar un análisis de la metodología actualmente vigente en el entorno empresarial.

4.2.1. Anotación de preguntas

Se ha llevado a cabo una elicitation de requisitos mediante la implementación de un cuestionario electrónico dirigido al personal técnico de la sede de Zuatzu en LKS Next, incluyendo coordinadores de equipo con experiencia en la incorporación de desarrolladores junior. En dicho cuestionario, se solicitó a los profesionales que, acorde a su ámbito de especialización, anotasen las interrogantes que formularían a un sistema de onboarding hipotético.

Se recabaron un total de 63 interrogantes aportadas por 8 profesionales del ámbito del desarrollo software. Posteriormente, dichas cuestiones fueron ampliadas y categorizadas mediante la utilización del modelo Claude 3.7 Sonnet¹ en su versión razonadora, tras lo cual se procedió a una anotación manual para eliminar elementos redundantes. La compilación clasificada final puede consultarse en el Anexo B.1, donde se han identificado los siguientes dominios de conocimiento:

- **Información general:** objetivo, finalidad y contexto del proyecto.
- **Entorno y despliegue:** entornos de desarrollo y sistemas de despliegue disponibles.
- **Gestión del proyecto:** comunicación y coordinación del equipo, metodología de contribución y gestión de tareas y requisitos.
- **Estándares y prácticas:** estándares de codificación, estándares visuales, estándares de integración y aspectos legales.
- **Documentación:** información sobre las fuentes de documentación disponibles para el proyecto, incluyendo su ubicación.

¹Claude 3.7 Sonnet: <https://www.anthropic.com/clause/sonnet>

- **Recursos adicionales:** recursos externos al proyecto como formaciones y documentación de librerías.
- **Arquitectura del sistema:** cuestiones de diseño del código fuente sobre todos los niveles de abstracción del modelo C4[66]. Este modelo define los niveles de abstracción de un proyecto software en 4 niveles:
 - Actores y sistemas externos con los que interactúa el sistema
 - Contenedores y aplicaciones que componen el sistema
 - Componentes que definen cada contenedor
 - Diagrama de clases e interfaces de cada componente

A modo de ejemplo, la siguiente pregunta se ha clasificado como parte de la gestión del proyecto: ¿Dónde puedo encontrar la descripción detallada de las tareas asignadas o disponibles?

4.2.2. Metodología empresarial

La metodología implementada ha sido consultada directamente con los desarrolladores de LKS Next. Al tratarse de una empresa consultora, las prácticas aplicadas presentan variaciones en función del cliente específico al que se presta servicio. No obstante, se ha identificado que dicha metodología se fundamenta en las siguientes directrices comunes:

- **Gestión del tiempo:** los integrantes del equipo utilizan una aplicación propietaria para el registro y contabilización de las horas invertidas en cada proyecto.
- **Gestión de tareas:** la asignación, seguimiento y monitorización de tareas se efectúa a través de herramientas especializadas como Jira².
- **Documentación:** la documentación técnica y funcional de los proyectos se distribuye en diversas plataformas, incluyendo páginas web específicas o directorios designados dentro del propio repositorio del proyecto.
- **Repositorio del proyecto:** la gestión del código fuente se realiza de manera centralizada en un repositorio común, alojado en plataformas como GitHub o el GitLab³ propio de LKS Next.

Adicionalmente, el flujo de trabajo establecido entre los equipos de diseño y desarrollo sigue una metodología específica. En una entrevista con un miembro del departamento de diseño (acta disponible en Anexo A.5), se ha documentado el siguiente proceso:

²Jira: <https://www.atlassian.com/es/software/jira>

³GitLab: <https://about.gitlab.com/>

En primera instancia, los requisitos visuales del proyecto se documentan en un repositorio de Confluence⁴ sincronizado con el cliente. Posteriormente, el equipo de diseño elabora un prototipo visual empleando herramientas especializadas como Figma⁵. Una vez aprobados los diseños, estos se transforman en maquetas HTML funcionales, las cuales son compartidas con el equipo de desarrollo a través del sistema de almacenamiento en la nube de Google Drive.

4.3. Recursos a utilizar

Tras definir los requisitos fundamentales y los dominios de conocimiento necesarios, se requiere establecer el contexto específico donde integrar el sistema a desarrollar.

4.3.1. Proyecto software

Debido a las restricciones legales existentes, no es viable la utilización de proyectos comerciales en entornos de producción. Bajo estas restricciones, para el desarrollo del presente trabajo se ha seleccionado el proyecto propietario IA-core-tools. Este constituye un conjunto de herramientas para el desarrollo de agentes basados en LLM con capacidad de acceso a repositorios de información mediante RAG. La aplicación está implementada en Flask⁶ y proporciona una API que facilita el desarrollo de agentes de manera interactiva a través de una interfaz visual, permitiendo a los desarrolladores de LKS Next crear soluciones para proyectos en producción.

Se dispone de acceso completo al repositorio de GitLab del proyecto, lo que proporciona información detallada sobre todos los usuarios, contribuciones realizadas, gestión de incidencias y administración de tareas. Asimismo, se dispone de las maquetas HTML utilizadas para crear la interfaz de usuario.

Cabe destacar que la metodología de gestión implementada difiere ligeramente de la empleada en proyectos comerciales. En este caso, el desarrollador principal es Aritz Galdos (supervisor en la empresa del presente proyecto), quien coordina reuniones tanto presenciales como virtuales con los contribuyentes del proyecto. Debido a las restricciones legales mencionadas anteriormente, no se dispone de acceso a la aplicación de gestión horaria del proyecto.

4.3.2. Recursos generados

Con el objetivo de una evaluación detallada del sistema, se ha procedido a la generación de documentación que replica con la mayor fidelidad posible las características del proyecto original. Para este propósito, se ha empleado nuevamente el modelo Claude 3.7 Sonnet, utilizando como datos de entrada toda la información de gestión previamente mencionada, así como ficheros clave extraídos del repositorio del código fuente. Posteriormente, se ha realizado un proceso de

⁴Confluence: <https://www.atlassian.com/es/software/confluence>

⁵Figma: <https://www.figma.com/>

⁶Flask: <https://flask.palletsprojects.com/en/stable/>

refinamiento manual de la documentación generada con el objetivo de eliminar imprecisiones e información errónea.

En una primera fase, se han elaborado tres documentos específicos relacionados con la sección visual del proyecto: `funcionamiento_y_diseño_interfaz`, `guia_de_estilos_visual` y `limitaciones_y_mejoras_pendientes`. Estos documentos han sido incorporados a un repositorio de Confluence para simular la documentación habitualmente proporcionada por los diseñadores del equipo.

Seguidamente, se han desarrollado trece documentos que constituyen una simulación de la “documentación oficial” del proyecto. Estos documentos recogen los elementos necesarios para la comprensión de la gestión, diseño y metodología de contribución del proyecto: `arquitectura-software.md`, `despliegue.md`, `equipo-y-comunicacion.md`, `estandares-codigo.md`, `flujos-trabajo.md`, `guia-contribucion.md`, `informacion-cliente.md`, `metodologia.md`, `modelo-negocio.md`, `onboarding.md`, `README.md`, `referencias-tecnicas.md` y `sistema-gestion-tareas.md`.

Por último, se ha generado la “documentación API” del proyecto mediante la utilización del agente RepoAgent^[67]. Esta documentación contiene explicaciones detalladas de todas las clases y métodos implementados en Python, que conforman la estructura del dominio y del modelo de negocio. El uso de este agente externo se explica detalladamente en la sección 6.2.1.1.

Esta infraestructura documental permite simular el proceso de onboarding de un nuevo desarrollador en el proyecto IA-core-tools, evaluando las respuestas del sistema agéntico ante las interrogantes típicas identificadas en los dominios de conocimiento.

Una vez definidos los requisitos y recursos del sistema, el siguiente capítulo comienza con la descripción de la implementación.

5

CAPÍTULO

Diseño del sistema

Tras establecer los parámetros correspondientes a la gestión del proyecto, este capítulo presenta el diseño del sistema desarrollado. Para ello se aborda: el diseño de agentes basado en LangGraph, la arquitectura distribuida del sistema multiagente, la implementación de servidores y cliente MCP, y la estructura de directorios del proyecto.

5.1. Diseño de agentes

LangGraph proporciona un marco de trabajo estructurado para la creación de flujos de ejecución en forma de grafo. Estos grafos se construyen mediante el objeto StateGraph para establecer la lógica de enrutamiento entre nodos, lo que requiere un estado compartido que coordine la ejecución. Los nodos se definen como funciones de Python que operan sobre el estado del grafo, representado por un diccionario tipado. El sistema permite dos tipos de nodos: los de ejecución realizan operaciones modificando el estado, mientras que los condicionales determinan qué nodo ejecutar según el estado actual.

La Figura 5.1 ejemplifica este paradigma mediante una implementación del patrón ReAct (véase Sección 2.2.2.1). La ejecución se inicia en el nodo de razonamiento, donde una función invoca al LLM y almacena la respuesta como mensaje en el estado. Posteriormente, el nodo condicional `should_continue()` evalúa si la respuesta del modelo requiere iteraciones adicionales: cuando contiene llamadas a herramientas se procede a ejecutarlas, mientras que su ausencia indica que la consulta ha sido resuelta. Tras la ejecución de las herramientas, se incorpora su resultado a los mensajes del estado, reiniciando el paso de razonamiento con la nueva información.

Este enfoque proporciona un marco de trabajo orientado a la composición, donde un nodo puede constituir a su vez otro grafo compilado que represente a otro agente. No obstante, surge una problemática de redundancia cuando dos agentes comparten gran parte del grafo que los

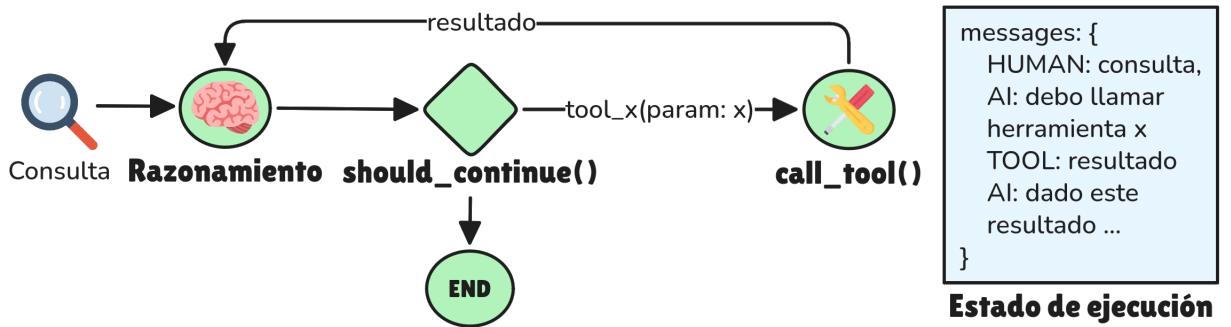


Figura 5.1: Esquema de implementación de patrón ReAct en Langgraph

compone. Para evitar la duplicación de código, podría desarrollarse un grafo que contemple la ejecución de ambos tipos de agentes, determinando cuál ejecutar según un parámetro en el estado. Si bien esta solución es viable, la programación orientada a objetos ofrece una alternativa más elegante: la herencia.

Es por este motivo que se ha optado por un enfoque híbrido que combina composición y herencia. Cada agente está implementado mediante una clase de Python, la cual contiene la función `create_graph()`, encargada de componer el grafo representativo de la ejecución del agente. De este modo, la clase del agente define las funciones que representan cada nodo del grafo, pudiendo heredar aquellas definidas en clases más abstractas.

La Figura 5.2 ilustra esta arquitectura de clases, donde se observa la jerarquía de herencia desde la clase base `BaseAgent` hasta los agentes más especializados. Adicionalmente, esta metodología facilita el almacenamiento de datos no vinculados a la ejecución individual. Los datos encapsulados en los atributos de las clases representan información invariable en cada ejemplo de ejecución, como el nombre del agente o las herramientas disponibles; mientras que los atributos almacenados en el estado del grafo son específicos de la ejecución, como los mensajes generados durante la instancia de ejecución concreta.

El funcionamiento específico de cada tipo de agente se detalla en la Sección 5.2.1.

5.2. Arquitectura del sistema

El sistema implementado explora arquitecturas aplicables a entornos con restricciones de procesamiento, orientándose a responder consultas mediante el acceso eficiente a múltiples fuentes de datos. El diseño contempla las siguientes restricciones:

- **Ventana de contexto limitada:** los proyectos en producción pueden contener millones de líneas de código y documentación de comparable magnitud. Los modelos del estado del arte no poseen todavía la capacidad para procesar dicho volumen textual.

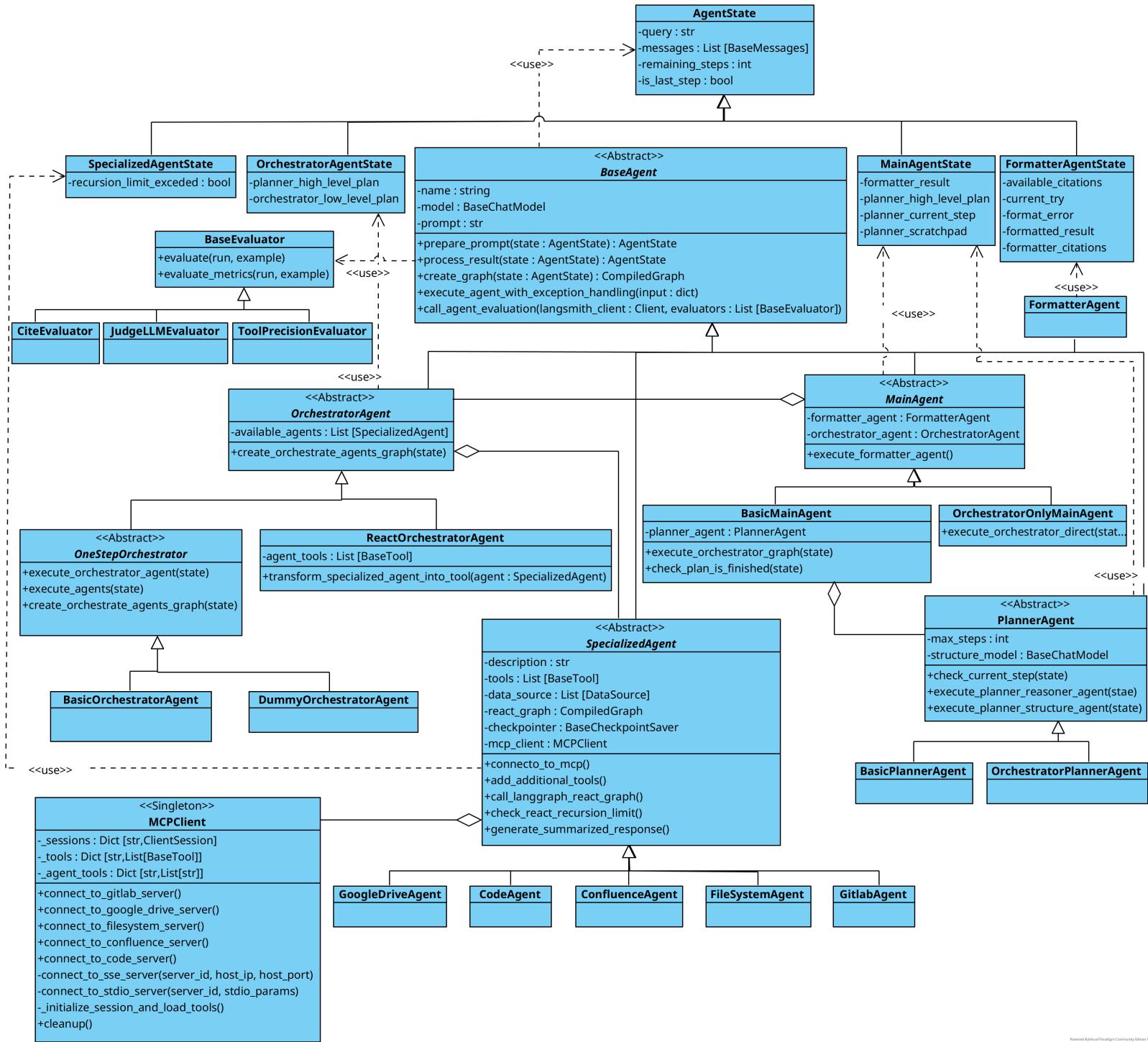


Figura 5.2: Diagrama de clases UML de del sistema de agentes.

- **Costo asociado:** incluso cuando el texto cabe en la ventana de contexto, las iteraciones con grandes volúmenes textuales generan un gasto computacional prohibitivo en entornos productivos. Esto crea la necesidad de optimizar selectivamente el acceso a la información.

Para abordar estas limitaciones, el sistema adopta un enfoque distribuido ilustrado en la Figura 5.3. La arquitectura reparte el procesamiento entre agentes especializados, cada uno interactuando exclusivamente con su fuente asignada. Estos agentes procesan los documentos originales y sintetizan únicamente la información pertinente, evitando así que niveles superiores deban procesar detalles irrelevantes. El agente orquestador analiza una operación y determina qué especialistas deben intervenir según las fuentes más apropiadas.

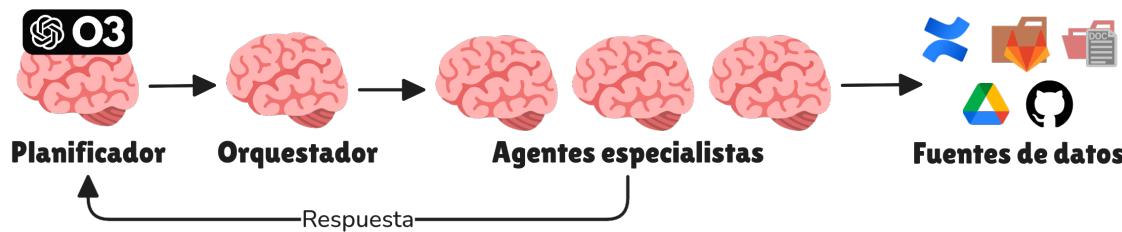


Figura 5.3: Esquema general de agentes del sistema

Adicionalmente, se implementa una coordinación de alto nivel mediante el agente planificador, que establece la secuencia lógica de operaciones. Por ejemplo, ante una consulta sobre ejemplos de aplicación de la guía de estilos, se priorizaría localizar dicha guía para posteriormente examinar implementaciones específicas en el código fuente.

Una vez explicado el funcionamiento general del sistema; el Capítulo 7 explica en mayor profundidad cada variación implementada.

5.2.1. Diagrama de agentes

Como se puede observar en el diagrama de clases (Figura 5.2), la clase abstracta BaseAgent constituye la base de la que heredan todos los agentes. Dicha clase establece las funcionalidades comunes que implementan todos los agentes del sistema:

- **Lógica de ejecución:** la función `execute_agent_with_exception_handling()` se encarga de invocar el grafo definido en la función `create_graph()`, gestionando las posibles excepciones e inicializando los parámetros de entrada requeridos. Todos los agentes derivados deben implementar la función `prepare_prompt()` e integrarla en su grafo de ejecución correspondiente, para definir el mensaje inicial del sistema que instruye al agente.
- **Gestión del resultado:** la función `process_result()` se encarga de procesar y devolver el resultado de ejecución del agente para un estado específico. Este resultado varía según el

agente; en los agentes especialistas corresponde a la respuesta que sintetiza su ejecución completa.

- **Evaluación del agente:** la función `call_agent_evaluation()` establece la lógica de evaluación específica para cada agente, utilizando las métricas definidas y representadas por la clase `BaseEvaluator`. El Capítulo 9 describe este proceso en detalle.

Los agentes que heredan del agente base son los siguientes:

- **SpecializedAgent:** representa de forma abstracta los agentes que realizan búsquedas en fuentes de información especializadas, aspecto detallado en el Capítulo 6. Integra la lógica de gestión de las herramientas utilizadas mediante la clase `MCPClient`, explicada en la Sección 5.4. Asimismo, contiene una secuencia de instancias de `DataSource`, las cuales definen todos los documentos citables en las fuentes de datos disponibles, descrita en la Sección 6.1.2.
- **OrchestratorAgent:** implementa la lógica de enrutamiento de los agentes especialistas, determinando en cada caso qué agentes ejecutar para resolver una cuestión específica. Véase la Sección 7.2.
- **PlannerAgent:** define la estrategia de planificación a seguir para una consulta determinada, elaborando planes secuenciales dinámicamente adaptables. Este proceso se detalla en la Sección 7.1.2.
- **FormatterAgent:** transforma una secuencia de ejecución de agentes en un resultado estructurado y comprensible, compuesto por una respuesta en formato textual y un conjunto de citas que referencian a documentos del proyecto software.
- **MainAgent:** establece el flujo general de la ejecución, especificando si se implementarán agentes planificadores, orquestadores y enlazando su respuesta al agente formateador. Este proceso se describe en la Sección 7.1.

Cada agente implementa su propia clase de estado para gestionar los atributos vinculados a una ejecución específica. Estas clases extienden el estado común `AgentState`, que incorpora atributos como los mensajes a incluir en el contexto de entrada. Como se ha descrito en la Sección 2.2.3, la interacción con los agentes sigue una estructura conversacional. Para materializar este enfoque, se utiliza la implementación de mensajes de LangChain: `SystemMessage` para las directrices que instruyen al agente, `HumanMessage` para las consultas o mensajes del usuario, y `AIMessage` para incorporar las respuestas de otros agentes.

Durante la fase de inicialización, se procede a la creación de los grafos correspondientes a todos los agentes mediante la instanciación de sus respectivas clases, estableciendo simultáneamente las conexiones necesarias con los servidores MCP. Una vez completado este proceso de inicialización, los agentes adquieren la capacidad de responder a todas las consultas requeridas sin necesidad de reconexión.

5.3. Servidores MCP utilizados

En esta sección se describirán los cinco servidores MCP utilizados, los cuales exponen sus herramientas mediante los dos protocolos de comunicación introducidos en la Sección 2.3: el protocolo SSE (Server-Sent Events) y el protocolo STDIO (Standard Input/Output).

5.3.1. Servidores SSE

Estos se alojan en componentes separados, ya que el protocolo SSE permite desacoplar servidores y cliente MCP.

- **Servidor Confluence:** proporciona acceso de lectura al repositorio Confluence, utilizando el servidor MCP oficial de Atlassian¹. El componente `servidor_mcp_confluence` incorpora un script de lanzamiento `launch_mcp_server_confluence.py` (detallado en el Listado 5.1).

El script inicializa el servidor mediante el gestor de paquetes uv, implementando el paquete `mcp-atlassian`². Este paquete integra el SDK de MCP para crear un servidor ASGI que maneja herramientas conectadas a la API de Atlassian. Tras la extracción de variables de entorno desde `.env`, se ejecuta el comando `uvx` en el sistema.

Listing 5.1: `launch_mcp_server_confluence.py`: ejecución de lanzamiento del servidor MCP Confluence

```

1      load_dotenv()
2      confluence_url = os.getenv('CONFLUENCE_URL')
3      ...
4
5      # Construir el comando para uvx
6      command = ["uvx", "mcp-atlassian"]
7
8      command.extend(["--transport", mcp_transport])
9      command.extend(["--port", mcp_port])
10     command.extend(["--confluence-url", confluence_url])
11     command.extend(["--confluence-username", confluence_username])
12     command.extend(["--confluence-token", confluence_token])
13
14     # Ejecutar el comando
15     try:
16         subprocess.run(command)
17         ...

```

- **Servidor Código:** proporciona acceso de lectura al repositorio software, implementando herramientas para la consulta de su código fuente. Su desarrollo se ha realizado directamente

¹Repositorio de servidor MCP Atlassian: <https://github.com/sooperset/mcp-atlassian>

²Paquete `mcp-atlassian`: <https://pypi.org/project/mcp-atlassian/>

mediante el SDK de MCP para Python³, localizado en el fichero `src/mcp_code_server.py` del componente `servidor_mcp_bd_codigo`. La Sección 6.2.1.1 detalla la implementación de dichas herramientas.

El sistema emplea la clase `FastMCP`, que integra internamente un servidor ASGI para gestionar el enrutamiento de las herramientas implementadas. Como se especifica en el Listado 5.2, el procedimiento consta de la instancia de la clase, la asociación de diversas herramientas mediante el decorador `mcp.tool()`, y la posterior ejecución del servidor. Una vez vinculadas las herramientas, se generan automáticamente las rutas `call_tool` y `get_tools`, facilitando su acceso automatizado desde el cliente MCP.

Listing 5.2: `mcp_code_server.py`: ejecución del servidor MCP con acceso al código fuente

```

1 mcp = FastMCP("code_server_id")
2
3 # Vincular herramientas al servidor
4 @mcp.tool()
5 async def get_all_repository_files_list() -> TextContent:
6     """
7         Devuelve una lista en formato string serializable a JSON de todos los
8         ficheros en el repositorio respecto a su ruta relativa
9     """
10    files_list = get_all_files_list(db_session=db_session)
11    files_list_str=str(files_list)
12    return TextContent(
13        text=files_list_str,
14        type='text'
15    )
16
17 # Ejecutar servidor
18 if __name__ == "__main__":
19     try:
20         mcp.run(transport='sse')
21         ...

```

5.3.2. Servidores STDIO

Estos servidores se ejecutan estableciendo una conexión en el cliente MCP mediante una instancia de `StdioServerParameters`, que especifica los parámetros de configuración necesarios. De esta manera, cuando el cliente inicia la conexión, el servidor se ejecuta en un subproceso del mismo.

³SDK de MCP para Python: <https://github.com/modelcontextprotocol/python-sdk>

La configuración de dicha instancia sigue en los tres servidores el mismo procedimiento ilustrado en el Listado 5.3. Se construye el comando con los argumentos correspondientes y se incorporan las credenciales como variables de entorno.

Listing 5.3: mcp_multi_client.py: instanciado de StdioServerParameters para el servidor MCP de GitLab

```

1 server_command = "npx"
2 server_args = ["-y", "@modelcontextprotocol/server-gitlab"]
3
4 # Obtener las credenciales desde las variables de entorno
5 server_env = {
6     "GITLAB_PERSONAL_ACCESS_TOKEN": os.getenv('GITLAB_PERSONAL_ACCESS_TOKEN'),
7     "GITLAB_API_URL": GITLAB_API_URL
8 }
9
10 # Crear instancia StdioServerParameters indicando las credenciales de GitLab como
11 # variables de entorno para el servidor
12 server_params = StdioServerParameters(
13     command=server_command,
14     args=server_args,
15     env=server_env
)
```

Mediante este procedimiento, se han utilizado los siguientes servidores:

- **Servidor Sistema de ficheros local:** proporciona acceso a un directorio específico, que simula el repositorio de la “documentación oficial” del proyecto. Se ha utilizado el servidor de Anthropic FileSystem⁴, ejecutándolo mediante el comando npx con el paquete del registro npm⁵ @modelcontextprotocol/server-filesystem⁶.
- **Servidor GitLab:** facilita el acceso al repositorio de GitLab con funcionalidades para leer o modificar ficheros, incidencias, usuarios y contribuciones. Se ejecuta utilizando el paquete npm @modelcontextprotocol/server-gitlab⁷. Debido a que no ofrece capacidades específicas para la lectura de ciertos recursos, se han desarrollado herramientas complementarias accediendo manualmente a la API de GitLab, detalladas en la Sección 6.2.5.
- **Servidor Google Drive:** proporciona acceso a un directorio de Google Drive, donde se alojan las maquetas HTML del proyecto software. A pesar de existir un repositorio oficial de MCP para la conexión con Google Drive, este no ofrece todas las funcionalidades requeridas,

⁴FileSystem: <https://github.com/modelcontextprotocol/servers/tree/main/src/filesystem>

⁵Npm: <https://www.npmjs.com/>

⁶Server-filesystem: <https://www.npmjs.com/package/@modelcontextprotocol/server-filesystem>

⁷Server-gitlab: <https://www.npmjs.com/package/@modelcontextprotocol/server-gitlab>

por lo que se ha empleado una modificación desarrollada en JavaScript⁸. En consecuencia, en lugar de especificar el paquete a utilizar en la instancia StdioServerParameters, se indica la ruta del script a ejecutar. También se ha modificado la herramienta de listar ficheros para mostrar recursivamente todos los ficheros dentro de los directorios listados.

Para obtener acceso al directorio de Google Drive mediante la API, se ha configurado una aplicación en Google Cloud⁹ siguiendo las directrices establecidas en el servidor MCP de Google Drive oficial¹⁰. En dicha plataforma, se han obtenido las credenciales de acceso, las cuales se encuentran almacenadas en `mcp_google_drive/credentials/gcp-oauth.keys.json`.

La API requiere de autenticación mediante el protocolo OAuth2.0¹¹ con la cuenta de Google. Para la generación del token temporal `.gdrive-server-credentials.json`, es necesario ejecutar el script del servidor especificando el parámetro auth, el cual iniciará el proceso de autenticación en el navegador.

5.4. Cliente MCP

Una vez explicada la estructura general de agentes y servidores MCP, en esta sección se detallará el mecanismo de comunicación entre los agentes y sus respectivos servidores para acceder a las herramientas disponibles.

Se ha implementado un cliente MCP mediante el patrón Singleton que permite gestionar todas las conexiones activas con los servidores MCP. Esta clase mantiene un diccionario para todas las herramientas y sesiones de conexiones MCP, utilizando el identificador del servidor como clave, tal como se ilustra en el Listado 5.4.

Listing 5.4: `mcp_multi_client.py`: clase Singleton MCPClient

```

1 class MCPClient:
2     _instance = None
3
4     # Sesiones con servidores: id de servidor -> objeto de session
5     _sessions: Dict[str, ClientSession] = {}
6     # Herramientas disponibles por cada servidor: id servidor -> lista herramientas
7     _tools: Dict[str, List[BaseTool]] = {}
8     # Herramientas requeridas por cada agente: id agente -> nombre herramienta
9     _agent_tools: Dict[str, List[str]] = {}

```

⁸Servidor MCP Google Drive: <https://github.com/felores/gdrive-mcp-server/blob/main/index.ts>

⁹Google Cloud: <https://cloud.google.com/?hl=es>

¹⁰Directrices de autenticación: <https://github.com/modelcontextprotocol/servers/tree/main/src/gdrive>

¹¹OAuth2.0: <https://oauth.net/2/>

Para acceder a las herramientas, cada agente ejecuta los tres pasos secuenciales ilustrados en el Listado 5.5:

Listing 5.5: google_drive_agent_graph.py: función connect_to_mcp en agente Google Drive

```

1  async def connect_to_mcp(self):
2      # 1. Obtener instancia del cliente
3      self.mcp_client = MCPClient.get_instance()
4      # 2. Conectar al servidor correspondiente al agente
5      await self.mcp_client.connect_to_google_drive_server()
6      # 3. Registrar agente y obtener herramientas
7      self.mcp_client.register_agent(self.name, self.tools_str)
8      self.tools = self.mcp_client.get_agent_tools(self.name)

```

La función de conexión al servidor correspondiente ejecutará la función del protocolo utilizado: connect_to_stdio_server() (Listado 5.6) para STDIO y connect_to_sse_server() (Listado 5.7) para SSE.

Listing 5.6: mcp_multi_client.py: función connect_to_stdio_server en el cliente MCP

```

1  async def connect_to_stdio_server(self, server_id: str, stdio_params:
2      StdioServerParameters):
3      # Establecer la conexión usando el exit_stack global
4      stdio_transport = await global_exit_stack.enter_async_context(stdio_client(
5          stdio_params))
6      stdio, write = stdio_transport
7
8      # Crear la sesión
9      session = await global_exit_stack.enter_async_context(ClientSession(stdio,
10         write))
11      self._sessions[server_id] = session
12
13      await self._initialize_session_and_load_tools(server_id)

```

Listing 5.7: mcp_multi_client.py: función connect_to_sse_server en el cliente MCP

```

1  async def connect_to_sse_server(self, server_id: str, host_ip: str, host_port: int
2      ):
3      # Usar el exit_stack global
4      streams = await global_exit_stack.enter_async_context(
5          sse_client(f"http://[{host_ip}]:{host_port}/sse"))
6
7      session = await global_exit_stack.enter_async_context(
8          ClientSession(streams[0], streams[1]))
9

```

```

10     self._sessions[server_id] = session
11
12     await self._initialize_session_and_load_tools(server_id)

```

Las conexiones implementan el SDK de MCP, generando objetos ClientSession en un AsyncExitStack global. Esta pila gestiona la liberación automática de recursos y permite cerrar todas las conexiones de forma controlada mediante cleanup(), como se detalla en el Listado 5.8.

Listing 5.8: mcp_multi_client.py: función cleanup() en el cliente MCP

```

1 @staticmethod
2 async def cleanup():
3     try:
4         if global_exit_stack:
5             await global_exit_stack.aclose()
6     ...

```

Finalmente, ambos protocolos ejecutan _initialize_session_and_load_tools() (Listado 5.9), que realiza tres tareas: inicializa la sesión de conexión, adapta las herramientas MCP a instancias BaseTool de LangChain, y añade manejo de excepciones para evitar que los errores se propaguen al agente.

Listing 5.9: mcp_multi_client.py: función _initialize_session_and_load_tools en el cliente MCP

```

1 async def _initialize_session_and_load_tools(self, server_id: str):
2     await self._sessions[server_id].initialize()
3     # Adaptar herramientas a BaseTool de LangChain
4     tools = await load_mcp_tools(self._sessions[server_id])
5     # Añadir control de excepciones a las herramientas
6     wrapped_tools = [patch_tool_with_exception_handling(tool) for tool in tools]
7     self._tools[server_id] = wrapped_tools

```

5.5. Estructura del proyecto

El repositorio se estructura en cuatro componentes independientes, cada uno con su propio entorno virtual de Python, como ilustra la Figura 5.4. El componente principal sistema_agentes contiene todos los agentes desarrollados y el cliente MCP, organizando el código fuente en src, los prompts y documentación en static, las variables de configuración en config.py y .env, y la lógica principal en main.py.

Los tres componentes restantes (servidor_mcp_bd_codigo, servidor_mcp_confluence y servidor_mcp_google_drive) contienen cada uno un servidor MCP independiente.

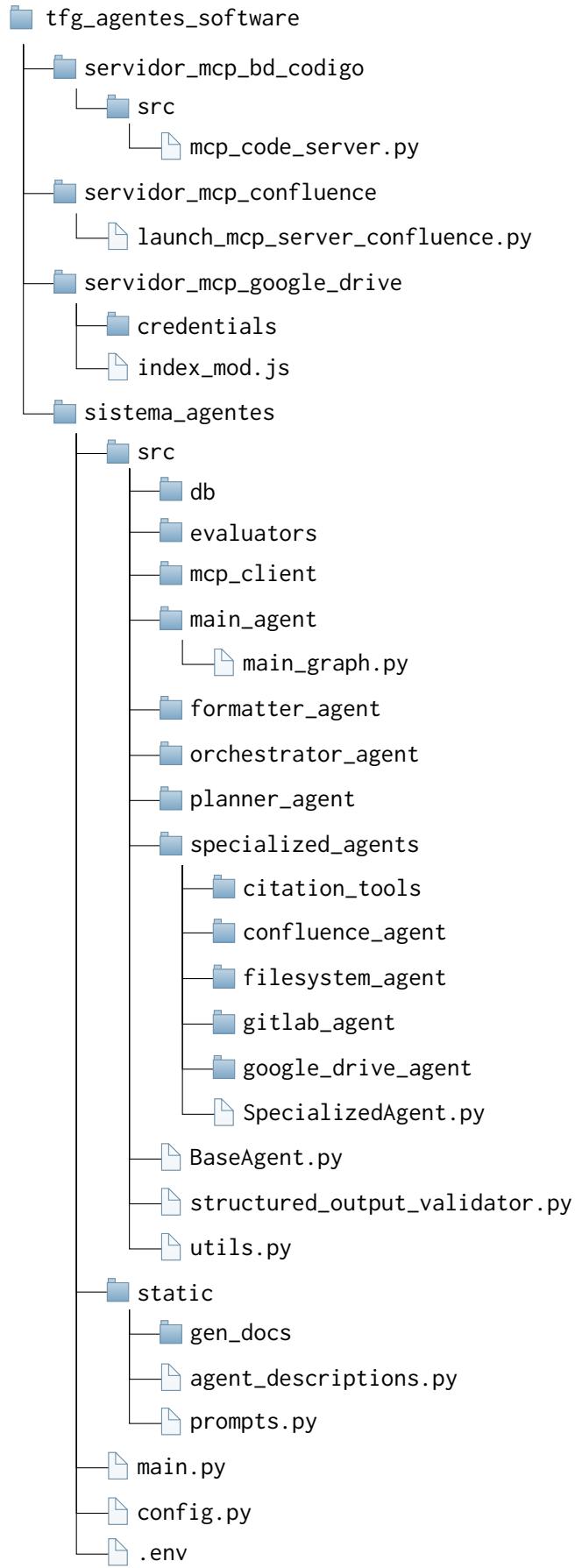


Figura 5.4: Estructura de directorios principales del proyecto

Agentes especializados

Una vez definido el diseño general del sistema, en este capítulo se detallan los agentes especializados que acceden a las fuentes de datos del proyecto software.

Para ello, se introduce primero la estructura de la clase base `SpecializedAgent`, junto al sistema de citas integrado. Posteriormente, se detallan las herramientas y la lógica de ejecución de los cinco agentes que extienden esta base. La Figura 6.1 muestra un esquema general de dichos agentes junto a sus respectivas fuentes de datos.

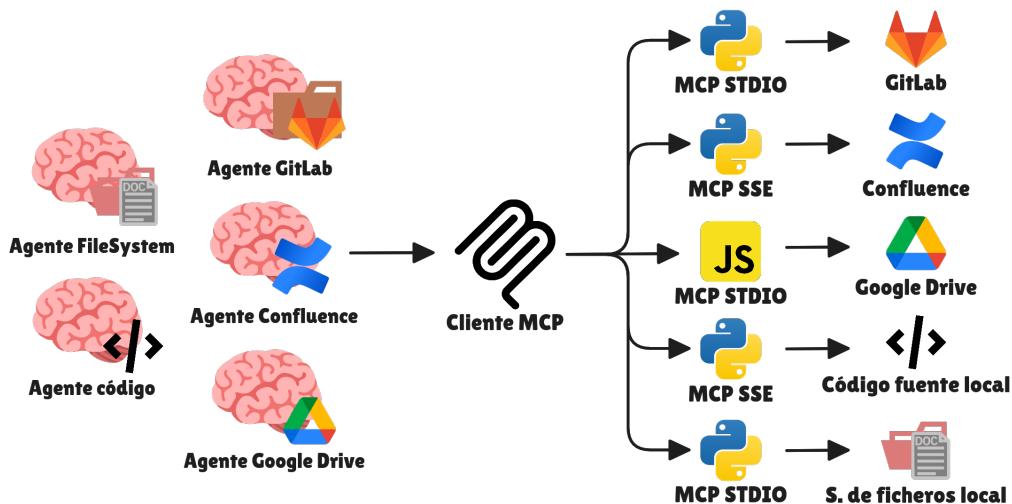


Figura 6.1: Agentes especialistas junto a sus respectivas fuentes de datos

6.1. Estructura SpecializedAgent

El grafo común de este agente comprende tres pasos principales: la configuración del prompt inicial mediante la función `prepare_prompt()`, heredada de `BaseAgent` y posteriormente extendida por cada agente especialista, la ejecución de un subgrafo que implementa el patrón ReAct (véase la Sección 2.2.2.1) y la ejecución de un agente resumidor cuando sea necesario. La Figura 6.2 ilustra dicho flujo de ejecución.

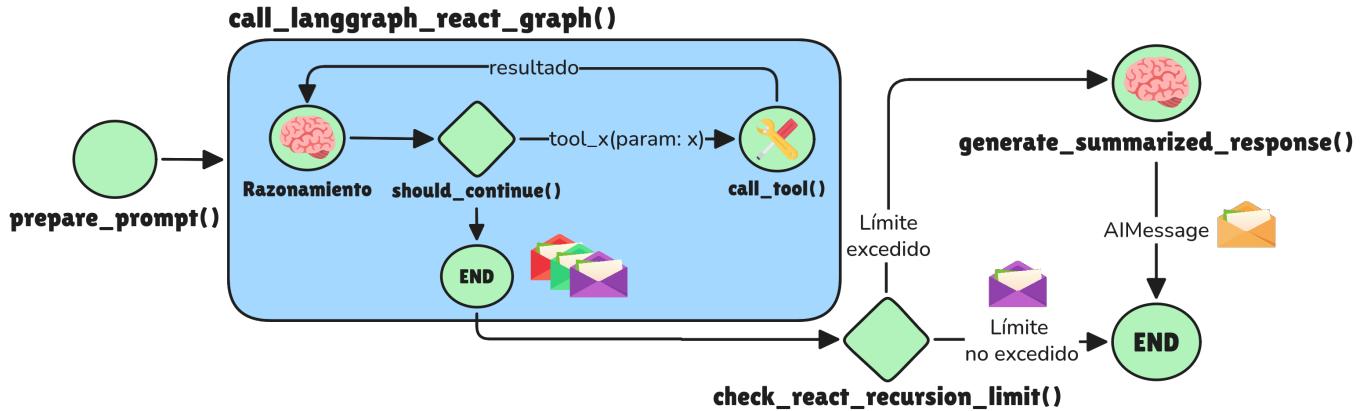


Figura 6.2: Grafo de ejecución de agentes especializados

El grafo ReAct se ha implementado utilizando el agente prefabricado `create_react_agent()` de LangGraph. Este agente acepta una serie de herramientas y un prompt inicial, entrando en un bucle de ejecución donde el agente invoca herramientas y observa sus resultados. El grafo finaliza cuando el mensaje del agente no contiene llamadas a herramientas, es decir, cuando incluye la respuesta final.

Se ha establecido un límite de iteraciones para el grafo ReAct, ya que se ha observado que ocasionalmente entra en un bucle excesivamente extenso al no encontrar la información requerida. Cuando se alcanza dicho límite, un agente resumidor genera una respuesta con la información disponible, observando todas las ejecuciones de las herramientas. En caso de no llegar a dicho límite, la respuesta del agente especialista será el último mensaje del grafo ReAct.

El Listado 6.1 ilustra la creación del grafo para los agentes especialistas, añadiendo un nodo para cada etapa mencionada. El condicional `check_react_recursion_limit` dirige la ejecución al nodo resumidor en caso de haber sobrepasado el límite de pasos en el agente ReAct.

Listing 6.1: `create_graph`: grafo de agentes especializados

```

1 def create_graph(self) -> CompiledGraph:
2     # Crear grafo ReAct
3     self.react_graph = create_react_agent()
  
```

```

4     model=self.model,
5     tools=self.get_agent_tools(),
6     checkpointer=self.checkpointer
7 )
8
9 # Crear grafo del SpecializedAgent
10 graph_builder = StateGraph(SpecializedAgentState)
11
12 # Añadir nodos
13 graph_builder.add_node("prepare", self.prepare_prompt)
14 graph_builder.add_node("react", self.call_langgraph_react_graph)
15 graph_builder.add_node("response_summarizer", self.generate_summarized_response)
16
17 # Establecer flujo entre nodos: prepare -> react -> condicional -> summarizer
18 graph_builder.set_entry_point("prepare")
19 graph_builder.add_edge("prepare", "react")
20 graph_builder.add_conditional_edges("react", self.check.react_recursion_limit)
21
22 return graph_builder.compile()

```

Para acceder al estado de ejecución del agente ReAct tras finalizar abruptamente por el límite de mensajes, se utiliza el sistema de autoguardado de LangGraph. Mediante la clase AsyncPostgresSaver, todos los estados de ejecución se almacenan en una colección de PostgreSQL y son accesibles mediante su identificador.

6.1.1. Gestión de herramientas

Algunas herramientas proporcionadas por los servidores MCP resultan innecesarias o contraproducentes para determinados agentes. Para evitar incluir contenido innecesario en los prompts, se indica al instanciar un agente el nombre de las herramientas que se utilizarán, filtrando posteriormente las herramientas en el cliente.

Por otra parte, algunos servidores MCP no proporcionan todas las funcionalidades requeridas. Para obtener estas herramientas adicionales, la función `add_additional_tools()` ejecutada durante la inicialización del agente incorpora, cuando es necesario, herramientas complementarias específicas para cada agente.

6.1.2. Sistema de citas

Para que el usuario final obtenga las fuentes de información utilizadas en la respuesta a su consulta, los agentes especializados deben referenciar los documentos más relevantes consultados que fundamentan su respuesta. Un enfoque directo consistiría en solicitar al agente que indique la ruta o nombre del archivo en el prompt, pero sería propenso a errores debido a que las direcciones URL de cada fuente de datos siguen patrones diferenciados. Para garantizar que los documentos

citados siempre existan y contengan una dirección válida, se ha implementado el sistema ilustrado en la Figura 6.3.

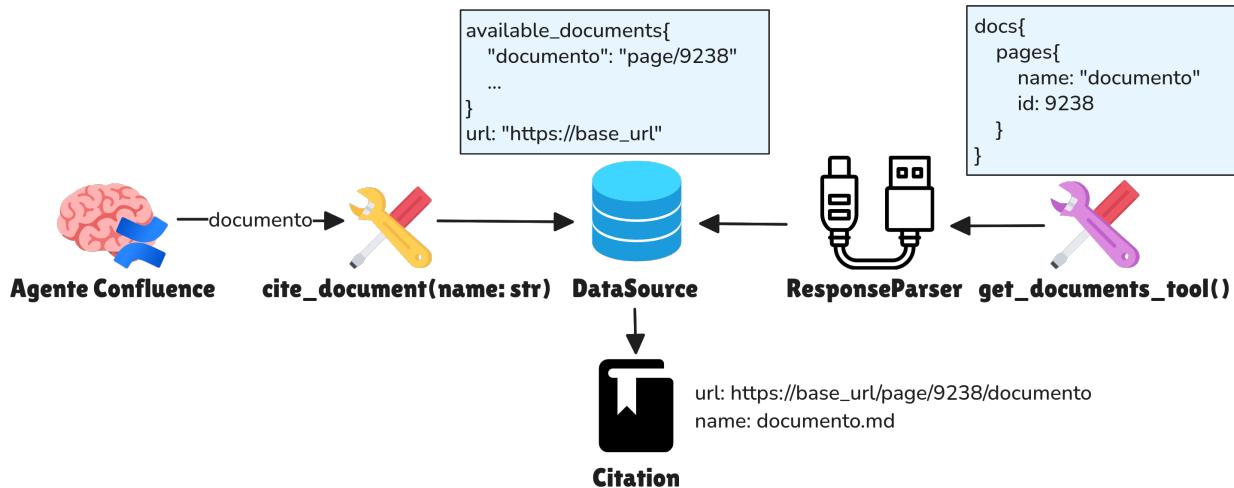


Figura 6.3: Diagrama de funcionamiento del sistema de citas

El sistema proporciona a cada agente una herramienta para citar documentos que se construye dinámicamente según las fuentes a las que tiene acceso. Cada fuente de datos se representa mediante una instancia **DataSource** (Listado 6.2), asignada a los agentes durante su inicialización. Al inicializar el agente, cada **DataSource** utiliza herramientas específicas del agente para catalogar todos los documentos citables y construye una estructura indexada. Debido a que las herramientas tienen una salida no estructurada, cada **DataSource** incorpora un adaptador (**ResponseParser**) para convertir la salida de la herramienta en la lista de documentos esperados.

Listing 6.2: **DataSource**: clase destinada a almacenar los documentos citables para una fuente de datos

```

1 class DataSource(ABC):
2     url: str
3     # Nombre de herramienta para obtener la lista de documentos disponibles
4     get_documents_tool_name: str | List[str]
5     # Nombre documento -> prefijo url: {"doc_confluence.md": "page/id_9238"}
6     available_documents: dict[str, str]
7     # Identificador de la fuente de datos
8     docs_id: str
9     parser: ResponseParser

```

Durante este proceso de indexación, cada **DataSource** almacena la información necesaria para construir automáticamente las URL completas cuando un agente necesita citar documentos. Esto se implementa mediante la combinación de una URL base compartida y un diccionario que mapea

cada documento con su prefijo específico en la URL. La Figura 6.3 muestra el ejemplo concreto del agente Confluence. En este caso, cada documento tiene asociado un prefijo constituido por el tipo de documento y un ID numérico en su estructura URL (url_base/tipo/id/nombre). El DataSource combina estos elementos para construir el enlace completo.

De esta manera, el agente dispondrá de una herramienta donde bastará con indicar el nombre del documento a citar, pudiendo referenciar también la propia fuente de datos mediante el nombre especificado en la descripción de la herramienta. Las citas se propagarán mediante objetos estructurados por toda la arquitectura de agentes para imprimir finalmente las citas empleadas.

6.2. Agentes implementados

En esta sección se detallan los cinco agentes desarrollados que extienden las funcionalidades descritas en la sección anterior.

6.2.1. Agente código

El flujo de este agente comprende el proceso ilustrado en la Figura 6.4. Mediante `prepare_prompt()`, el sistema configura el prompt inicial incluyendo las instrucciones del agente, el árbol de directorios obtenido con la herramienta `get_repository_tree_tool()` y fragmentos de código, denominados *chunks*, relevantes para la consulta actual mediante la herramienta `get_code_repository_rag_docs_from_query_tool()`. Una vez que recibe la consulta del usuario a través de un `HumanMessage`, el agente evalúa tres opciones: buscar chunks adicionales sobre un subdirectorio específico, acceder a archivos concretos con la herramienta `get_file_from_repository_tool()` especificando su ruta relativa, o responder directamente a la consulta.

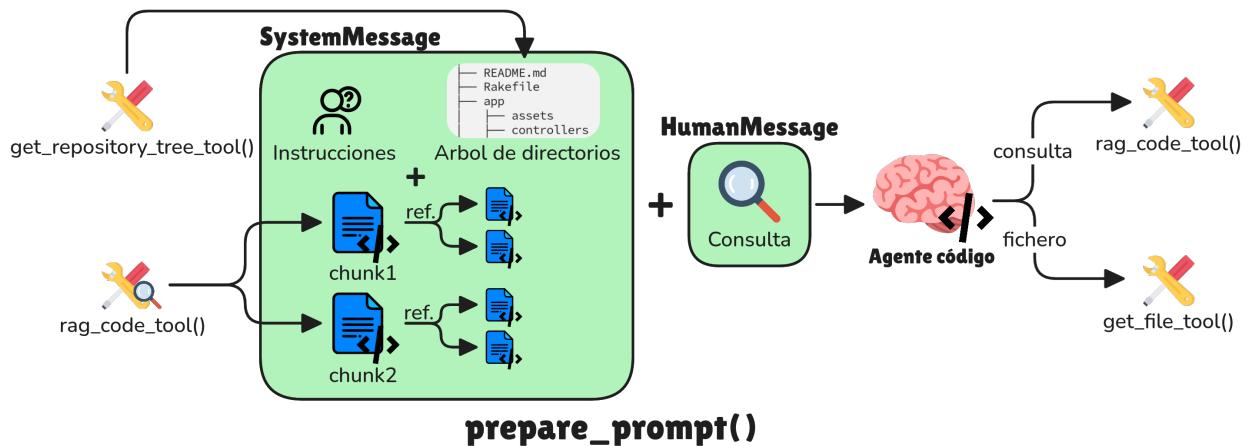


Figura 6.4: Flujo operativo del agente de código

Los chunks devueltos por dicha herramienta contienen tanto el código como la ruta relativa del archivo donde se declaran, permitiendo buscar en dicho archivo si fuese necesario. Adicionalmente, al extraer un chunk, se incluyen chunks referenciados o que referencian ese fragmento. Se define que un chunk referencia a otro cuando llama a una función o instancia una clase definida en el otro chunk.

6.2.1.1. Herramientas de acceso a código

Para implementar las herramientas del agente, en el componente `servidor_mc_bd_codigo` se han dividido los ficheros del proyecto GitLab en chunks y se han indexado en la base de datos PostgreSQL. El sistema RAG implementado utiliza la extensión PGVector¹ sobre PostgreSQL, que permite integrar vectores embeddings en tablas SQL convencionales, facilitando operaciones de búsqueda semántica combinadas con consultas SQL.

Se ha definido el modelo relacional mostrado en la Figura 6.5, utilizando el ORM SQLAlchemy² sobre Python. La estructura se basa en una tabla `FileSystem` para cada fichero o directorio, que puede contener múltiples `FileChunk` indexados en la columna `embedding` de tipo `Vector`. Cada `FileChunk` puede referenciar o ser referenciado por otros `FileChunk` mediante la relación muchos-a-muchos `ChunkReference`. La tabla `Ancestor` implementa un patrón de tabla de cierre (*closure table*, consulte Anexo A.6) para acceder eficientemente a los ficheros dentro de cualquier subdirectorio.

El Listado 6.3 muestra cómo se obtienen los fragmentos más relevantes para una consulta específica. El proceso comienza desde una entrada del sistema de ficheros (`FSEntry`), utiliza la tabla `Ancestor` para filtrar todos los ficheros descendientes, realiza una operación de unión con la tabla `FileChunk` para obtener todos los fragmentos contenidos en dichos ficheros, y finalmente los ordena según su relevancia semántica respecto a la consulta del usuario.

Listing 6.3: Obtener chunks relevantes para una consulta dentro de un subdirectorío específico

```

1 # Filtrar primero los descendientes con la closure table
2 descendant_ids = select(Ancestor.descendant_id).where(
3     Ancestor.ancestor_id == fs_entry.id
4 ).scalar_subquery()
5
6 # Buscar los chunks por orden de similitud haciendo un join con la tabla de fsentry
7 consulta = select(FileChunk, FileChunk.embedding.cosine_distance(query_embedding))\
8     .label('distance'))\
9     .join(FSEntry, FSEntry.id == FileChunk.file_id)\ \
10    .where(FileChunk.file_id.in_(descendant_ids))\ \
11    .order_by('distance')\ \
12    .limit(max_results)

```

¹PGVector: <https://github.com/pgvector/pgvector>

²SQLAlchemy: <https://www.sqlalchemy.org/>

```
12     results = self.db_session.execute(consulta).all()
```

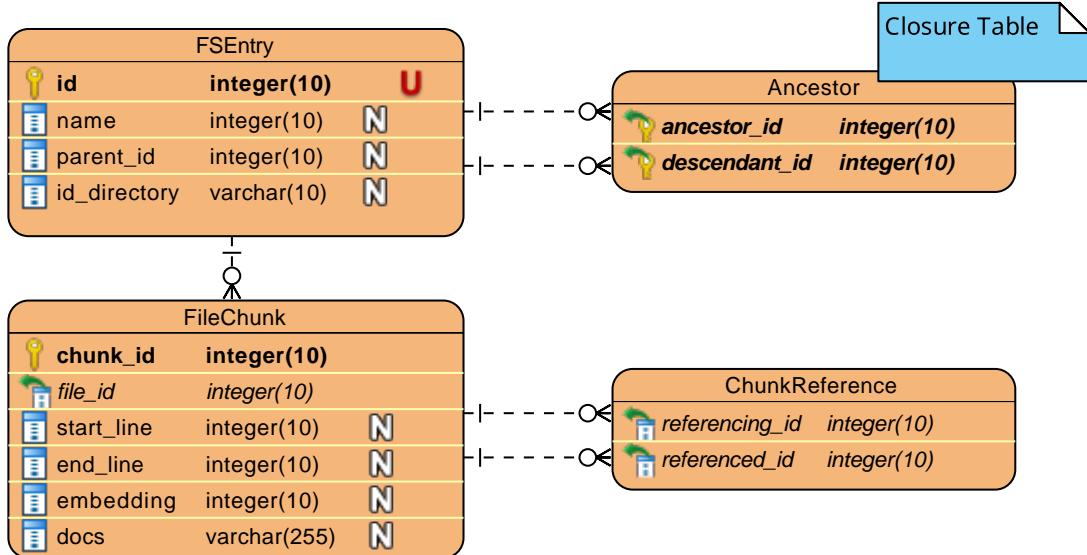


Figura 6.5: Diagrama relacional de la base de datos con el código fuente del proyecto software

Procedimiento de chunking Para dividir los ficheros en chunks, se han considerado las definiciones del código fuente (clases y funciones), utilizando la librería grep-ast³ para identificar definiciones y referencias en cada archivo.

La segmentación emplea un patrón State (Figura 6.6) que recorre secuencialmente el fichero añadiendo definiciones al chunk actual hasta alcanzar su capacidad máxima. Si la siguiente definición no cabe y el chunk contiene alguna definición, se guarda el chunk actual y se inicia uno nuevo. Si el chunk está vacío pero la siguiente definición es demasiado grande, se divide: para funciones mediante segmentación equitativa, y para clases mediante subdivisión por funciones internas.

El proceso de chunking se implementa mediante una función recursiva que, partiendo del directorio raíz, procesa cada elemento ignorando directorios y archivos específicos. Para cada fichero, obtiene definiciones y referencias mediante grep-ast, aplica el patrón State para la división, y registra en un diccionario las definiciones declaradas con su correspondiente identificador de chunk. Finalmente, un algoritmo relaciona las referencias con sus definiciones correspondientes en el modelo relacional.

³grep-ast: <https://github.com/Aider-AI/grep-ast>

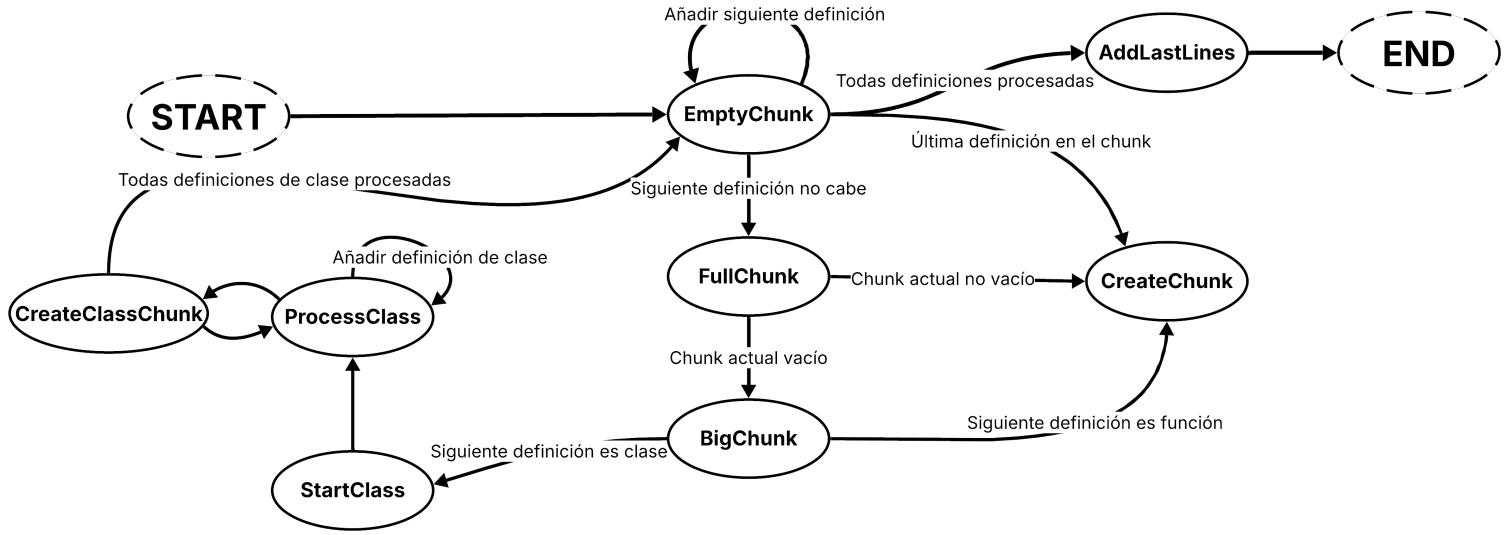


Figura 6.6: Algoritmo de chunking considerando definiciones de funciones y clases en el código fuente

La fiabilidad del procedimiento se ha verificado mediante nueve tests unitarios de caja negra con pytest⁴, ejecutados automáticamente en cada actualización de las ramas develop o master del repositorio mediante un flujo de integración continua configurado en el archivo .github/workflows/build.yml e integrado con SonarCloud⁵.

Procedimiento de indexación Para indexar cada chunk, se ha utilizado una plantilla que considera: el código del chunk, el fichero completo, el árbol del repositorio y la documentación API generada por RepoAgent (cuya ejecución se ilustra en el Listado 6.4). Este enfoque, en lugar de indexar únicamente el código fuente, permite que las consultas posteriores encuentren fragmentos de código semánticamente relevantes al compararse con estos embeddings contextualizados.

Para ejecutar el proceso de indexación de forma estructurada y asíncrona, se ha implementado un algoritmo basado en el patrón Pipeline⁶ que desacopla la iteración de las operaciones mediante tres niveles de ejecución:

- **PipelinePipelineStage:** Procesos iniciales a nivel de repositorio, incluyendo la generación del árbol de directorios mediante treelib⁷ y la creación de documentación API con RepoAgent.

⁴pytest: <https://docs.pytest.org/en/stable/>

⁵SonarCloud: <https://www.sonarsource.com/products/sonarcloud/>

⁶Patrón Pipeline: <https://medium.com/@bonnotguillaume/software-architecture-the-pipeline-design-pattern-from-zero-to-hero-b5c43d8a4e60>

⁷treelib: <https://pypi.org/project/treelib/>

- **FilePipelineStage:** Estructuración del procesamiento a nivel de fichero, permitiendo un seguimiento organizado.
- **ChunkPipelineStage:** Operaciones para cada chunk, abarcando la creación de documentación y su posterior indexación vectorial.

La ejecución sigue un flujo jerárquico donde primero se completan las operaciones a nivel de repositorio, seguidas por el procesamiento asíncrono de ficheros y finalmente la ejecución asíncrona de operaciones a nivel de chunk, manteniendo un orden secuencial dentro de cada nivel.

Listing 6.4: generate_extra_docs: Ejecución de agente RepoAgent para crear la documentación API

```

1 def generate_extra_docs(files_to_ignore: List[str], repo_path: str, extra_docs_path: str):
2     try:
3         files_to_ignore_str = ",".join(files_to_ignore)
4         command = f"repoagent run --model gpt-4o-mini --target-repo-path {repo_path}"
5         command += f"--markdown-docs-path {extra_docs_path} --ignore-list {files_to_ignore_str}"
6         exit_code = execute_and_stream_command(command)
    ...

```

6.2.2. Agente Google Drive

La función de este agente consiste en buscar y analizar las maquetas HTML almacenadas en Google Drive. Mediante la herramienta `gdrive_list_files()`, la función `prepare_prompt()` proporciona al agente información sobre todos los documentos disponibles. Posteriormente, el agente determina si debe buscar fragmentos relevantes en los documentos utilizando la herramienta `gdrive_search()`, o acceder a documentos completos mediante la herramienta `gdrive_read_file()`.

6.2.3. Agente Sistema de Ficheros

Este agente responde a consultas utilizando la documentación oficial como fuente de datos. La función `prepare_prompt()` proporciona la lista de documentos disponibles a través de la herramienta `directory_tree()` en el prompt inicial, pudiendo acceder a archivos específicos con `read_file()` o `read_multiple_files()`, o realizar búsquedas de pasajes relevantes a través de `rag_search_documentation()`.

Esta última herramienta se ha implementado utilizando la clase `PGVector` de LangChain. A diferencia de la implementación nativa de la librería `PGVector` utilizada en el agente de código, esta versión ofrece una abstracción más sencilla para búsquedas RAG, pero sacrifica la capacidad de combinar operaciones de búsqueda semántica con consultas SQL avanzadas.

Como muestra el Listado 6.5, se crean colecciones mediante instancias de PGVector donde una columna se vectoriza para búsqueda semántica y las demás actúan como metadatos filtrables. Entre estos metadatos se incluye la ruta del archivo indexado, lo que permite al agente identificar el origen de cada fragmento y decidir si necesita extraer el documento completo.

Listing 6.5: PGVector: uso de clase para indexar o buscar documentos

```

1 # Instanciar la clase en su versión asíncrona
2 vector_store = PGVector(
3     embeddings=self.embeddings,
4     collection_name=prefix_name,
5     connection=self.engine,
6     use_jsonb=True,
7     async_mode=True
8 )
9
10 # Añadir documentos (clase LangChain con texto y metadatos) al store
11 await self.vector_store.add_documents(docs)
12
13 # Buscar documentos similares mediante una operación vectorial
14 results = await self.vector_store.asimilarity_search(query, k=top_k)

```

6.2.4. Agente Confluence

Para este agente se han implementado dos versiones alternativas: una estándar y otra con un enfoque en el cacheo del prompt.

El *prompt caching* almacena la representación interna del LLM para secuencias de texto iniciales repetidas, evitando cálculos redundantes en consultas posteriores. APIs como OpenAI ofrecen descuentos significativos cuando se cachea gran parte de la entrada. La Figura 6.7 compara ambos enfoques, donde las secciones enmascaradas representan fragmentos del prompt inicial que se repiten entre diferentes llamadas y son cacheados. En ambos casos el cacheo finaliza al incorporar la consulta específica del usuario, dado que esta varía en cada interacción. Sin embargo, la segunda implementación logra cachear un volumen de texto considerablemente mayor.

Implementación estándar En esta implementación estándar, `prepare_prompt()` utiliza `confluence_search()` para identificar los documentos disponibles en la documentación del estilo visual, permitiendo al agente acceder posteriormente a documentos específicos mediante `confluence_get_page()`.

Enfocado en el cacheo del prompt Esta versión incorpora toda la documentación directamente en el contexto de entrada, permitiendo al agente responder mediante referencias a los documentos

pertinentes. Esta estrategia resulta viable únicamente cuando los documentos completos pueden incluirse dentro de la ventana de contexto del modelo.

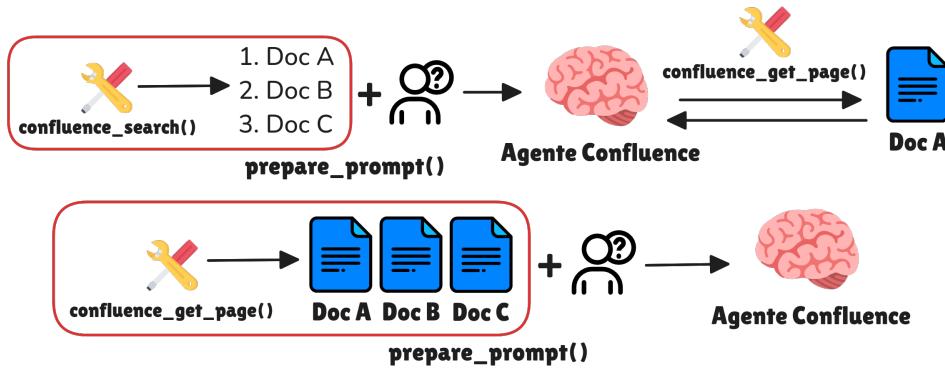


Figura 6.7: Comparación de agente Confluence estándar y enfocado en el cacheo del prompt

6.2.5. Agente GitLab

El agente recibe inicialmente mediante `prepare_prompt()` estadísticas del repositorio obtenidas a través de la herramienta `get_gitlab_project_statistics()`, incluyendo datos sobre contribuciones, usuarios e incidencias. Posteriormente, puede emplear las siguientes herramientas:

- `get_gitlab_project_commits(user_name, since, until, result_limit)`: recupera contribuciones (*commits*) con opciones de filtrado por usuario, fechas o cantidad.
- `get_gitlab_project_members()`: obtiene información sobre usuarios contribuyentes. Es de destacar que para filtrar contribuciones por usuario, el agente debe invocar primero esta herramienta para obtener los nombres de usuario en GitLab.
- `get_gitlab_branches()`: proporciona información sobre las ramas git del proyecto.
- `get_gitlab_issues()`: recupera las incidencias (*issues*) registradas en el proyecto.

Cabe destacar que, en lugar de citar documentos, este agente tiene la capacidad de citar contribuciones o incidencias del repositorio.

La implementación ha requerido el desarrollo de herramientas específicas mediante la API de GitLab utilizando la librería `requests`⁸. Como se detalla en el Listado 6.6, las herramientas se crean llamando a la API con las credenciales y la URL correspondiente.

⁸requests: <https://pypi.org/project/requests/>

Listing 6.6: Ejemplo de herramienta para agente GitLab directamente desde la API

```
1 # La herramienta llama a una función que gestiona la paginación en varias requests
2 @tool
3 def get_gitlab_project_members():
4     return execute_gitlab_api_request_with_pagination("members")
5 ...
6
7 # Ejecuta peticiones individuales para implementar la paginación
8 def execute_gitlab_api_request(url: str, params: Dict[str, Any] = None) -> Response:
9     gitlab_token = os.getenv('GITLAB_PERSONAL_ACCESS_TOKEN')
10    request_url = f"{GITLAB_API_URL}/projects/{GITLAB_PROJECT_URL}/{url}"
11    headers = { "PRIVATE-TOKEN": gitlab_token }
12
13    return requests.get(request_url, headers=headers, params=params)
```

Orquestación de agentes

Habiendo abordado tanto el diseño del sistema como el funcionamiento de los agentes especialistas, este capítulo se centra en los mecanismos de coordinación que permiten la operación del sistema completo.

Para ello se introducirá la lógica del agente principal (MainAgent), el agente planificador y el agente orquestador. Posteriormente se detallarán las variaciones exploradas en dicho marco de coordinación, concluyendo con el análisis de un caso práctico de ejecución.

7.1. Agente Principal

Este agente establece la lógica de enrutamiento entre el planificador y el orquestador, derivando finalmente el resultado al agente formateador. Todas las versiones principales de este agente pueden visualizarse en la Figura 7.2.

El proceso se inicia con la ejecución del agente planificador, que genera un plan de alto nivel compuesto por múltiples pasos secuenciales, donde cada paso constituye una cadena de texto que define específicamente qué información buscar. Posteriormente, se activa el agente orquestador con el primer paso del plan generado, cuya función consiste en determinar qué agentes especializados deben ejecutarse para dicha etapa. Los agentes seleccionados son ejecutados de forma asíncrona, generando cada uno un `CitedAIMessage`, estructura que integra tanto el contenido de la respuesta como una lista de citas referentes a los documentos consultados.

Este ciclo se itera hasta que el planificador establece la finalización del plan. A continuación, el agente formateador estructura el contenido en un formato estandarizado, consistente en un mensaje con formato de marcado estructurado (*Markdown*) y un índice de identificadores que referencia solo las citas empleadas de entre todas las generadas en las respuestas de los agentes.

Considerando el desarrollo de múltiples versiones de planificadores y orquestadores, se ha implementado un patrón Builder para facilitar la construcción del sistema a partir de los agentes especificados, permitiendo así las variaciones detalladas en la Sección 7.3. El Listado 7.1 muestra la creación, inicialización y ejecución del sistema mínimo con los cinco agentes especialistas.

Listing 7.1: Instanciación y ejecución del sistema mínimo con el patrón Builder

```

1 builder = FlexibleAgentBuilder()
2 agent = await (await (builder
3         .reset()
4         .with_main_agent_type("basic")
5         .with_planner_type("basic")
6         .with_orchestrator_type("basic")
7         .with_specialized_agents([
8             CodeAgent(),
9             CachedConfluenceAgent(),
10            GitlabAgent(),
11            FileSystemAgent(),
12            GoogleDriveAgent(),
13        ])
14        .initialize_agents())).build()
15
16 result = await agent.execute_agent_graph_with_exception_handling({
17     "query": "Qué entornos de despliegue existen en el proyecto?",
18 })

```

7.1.1. Respuesta estructurada

El sistema necesita interpretar las respuestas de los agentes de forma programática en varios escenarios. Para conseguir este objetivo, se implementan respuestas estructuradas que permiten procesar información de manera automatizada dentro de la arquitectura del sistema. Por ejemplo, el agente formateador puede separar claramente el texto de la respuesta de las citas utilizadas, mientras que el orquestador puede generar listas de agentes especializados en un formato directamente utilizable por el código.

El Listado 7.2 muestra la implementación mediante el adaptador `with_structured_output`, que incorpora en el prompt un esquema JSON específico que guía al modelo a generar una salida estructurada. Esta salida, gracias a los modelos Pydantic¹, puede ser automáticamente deserializada a objetos Python para su procesamiento. En los casos donde el modelo no logra producir la estructura correcta, se implementa un mecanismo de recuperación utilizando un `RetryOutputParser` de LangChain, que emplea un segundo modelo para reformular la respuesta

¹Pydantic: <https://docs.pydantic.dev/latest/>

según el formato requerido. Si este segundo intento también falla, el proceso se repite hasta obtener una salida válida o llegar al máximo de repeticiones.

Listing 7.2: Validación de la salida estructurada de un LLM

```

1  async def execute_structured_llm_with_validator_handling(prompt: str | Sequence[
2      BaseMessage], output_schema: Type[BaseModel], max_retries: int, llm: BaseChatModel
3      ) -> BaseModel:
4
5      structured_model = llm.with_structured_output(output_schema)
6      parser = PydanticOutputParser(pydantic_object=output_schema)
7      retry_parser = RetryOutputParser.from_llm(parser=parser, llm=default_llm)
8
9      last_exception: Optional[Exception] = None
10     raw_response: Optional[str] = None
11
12     for _ in range(max_retries):
13         try:
14             response = await structured_model.invoke(prompt)
15             # Validación programática de la salida del modelo
16             raw_response = response if isinstance(response, str) else None
17             if not isinstance(response, output_schema):
18                 response = output_schema.model_validate(response)
19             return response
20
21         except Exception as e:
22             last_exception = e
23             if raw_response is not None:
24                 try:
25                     # Reintento mediante un modelo parseador
26                     response = retry_parser.parse(raw_response)
27                     if not isinstance(response, output_schema):
28                         response = output_schema.model_validate(response)
29                     return response
30                 except Exception as retry_e:
31                     last_exception = retry_e
32             raise last_exception

```

7.1.2. Agente Planificador

Este agente establece planes de alto nivel para estructurar el proceso de respuesta mediante un procedimiento lógico secuencial. Adicionalmente, incorpora la capacidad de modificar dinámicamente dichos planes en función del estado de ejecución actual.

El flujo operativo del agente se ilustra en la Figura 7.1. En primera instancia, se verifica que el plan no esté finalizado mediante el condicional `check_current_plan()`. Se considera que un

plan ha concluido cuando la ejecución del planificador previo así lo ha determinado o cuando se ha excedido el límite de iteraciones establecido. En caso de detectarse la finalización del plan, se termina inmediatamente la ejecución del planificador.

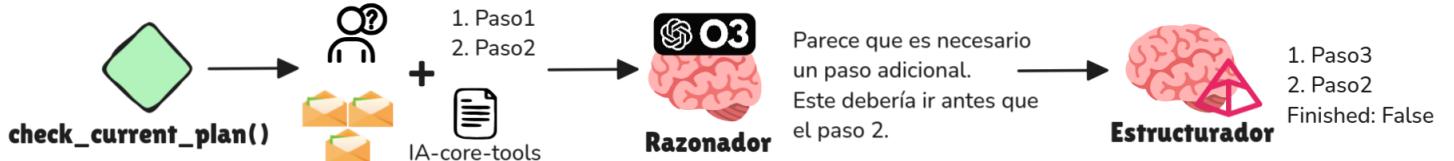


Figura 7.1: Flujo operativo del agente planificador

Posteriormente, un agente razonador procesa la consulta del usuario junto con una breve descripción del proyecto. En caso de no tratarse de la primera iteración de ejecución, este recibe también los planes anteriores y las respuestas de los agentes especializados, que constituyen el resultado de la ejecución del primer paso de todos los planes precedentes. La función del agente razonador consiste en analizar los siguientes pasos a ejecutar, generando un nuevo plan o modificando el existente.

Cabe destacar que el razonador no estructura directamente su plan, puesto que los agentes razonadores han sido optimizados para generar respuestas sin un formato predeterminado, habiéndose demostrado que imponer restricciones de formato reduce la precisión del razonamiento [68]. Por consiguiente, otro agente, denominado estructurador, transforma el plan textual elaborado por el razonador a un modelo de Pydantic. Este plan consiste en un objeto Python representado como una lista de pasos textuales con un argumento booleano que indica si el plan ha finalizado.

7.2. Agente orquestador

Este agente recibe una tarea específica, ya sea derivada de un paso del plan establecido o formulada como consulta directa por el usuario, y determina qué agentes especializados deben ejecutarse de forma asíncrona.

Para fundamentar su toma de decisiones, su prompt incorpora una descripción detallada de cada agente disponible, incluyendo una síntesis del tipo de información que gestiona cada uno. Este contexto se construye dinámicamente en función de los agentes integrados en el sistema, almacenando dicha caracterización en cada objeto `SpecializedAgent`.

7.3. Variaciones de orquestación

El sistema base sigue la estructura descrita en la sección 7.1, donde un orquestador selecciona los agentes a ejecutar para los pasos del plan generados por un planificador. Como se observa en la Figura 7.2a, este diseño establece una separación completa entre el agente planificador y el agente orquestador. Consecuentemente, el planificador no dispone de información sobre los

agentes especialistas disponibles, mientras que el orquestador carece de conocimiento sobre el plan general o la consulta original del usuario, concentrándose exclusivamente en el siguiente paso del plan actual.

En contraste, la arquitectura ilustrada en la Figura 7.2b integra el planificador y orquestador en un único agente. Esta fusión permite que disponga de la cuestión del usuario, el plan completo actual y los agentes que puede ejecutar. En esta implementación, tras la fase de razonamiento en el planificador, en lugar de extraer los pasos del plan a ejecutar, se obtienen directamente los agentes a ejecutar con sus respectivas consultas.

Estos dos primeros enfoques presentan ventajas e inconvenientes diferenciados. Mientras que el segundo enfoque resulta más simple y podría presumirse una precisión superior al integrar toda la información contextualizada en un único agente, el primero establece una separación más clara. Esta separación entre planificación y ejecución podría traducirse en decisiones más meditadas, potencialmente mejorando la precisión.

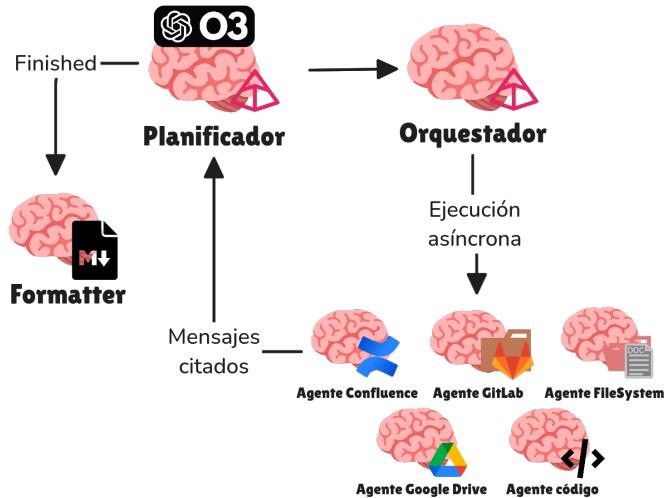
La tercera aproximación, ilustrada en la Figura 7.2c, explora un diseño de máxima simplicidad. Eliminando completamente la fase de planificación, implementa un orquestador que invoca a los agentes especializados mediante el patrón ReAct, interactuando con ellos encapsulándolos en herramientas. El propio orquestador determina la necesidad de ejecutar agentes adicionales, proporcionando una respuesta directa cuando considera que dispone de información suficiente.

7.3.1. Caso práctico

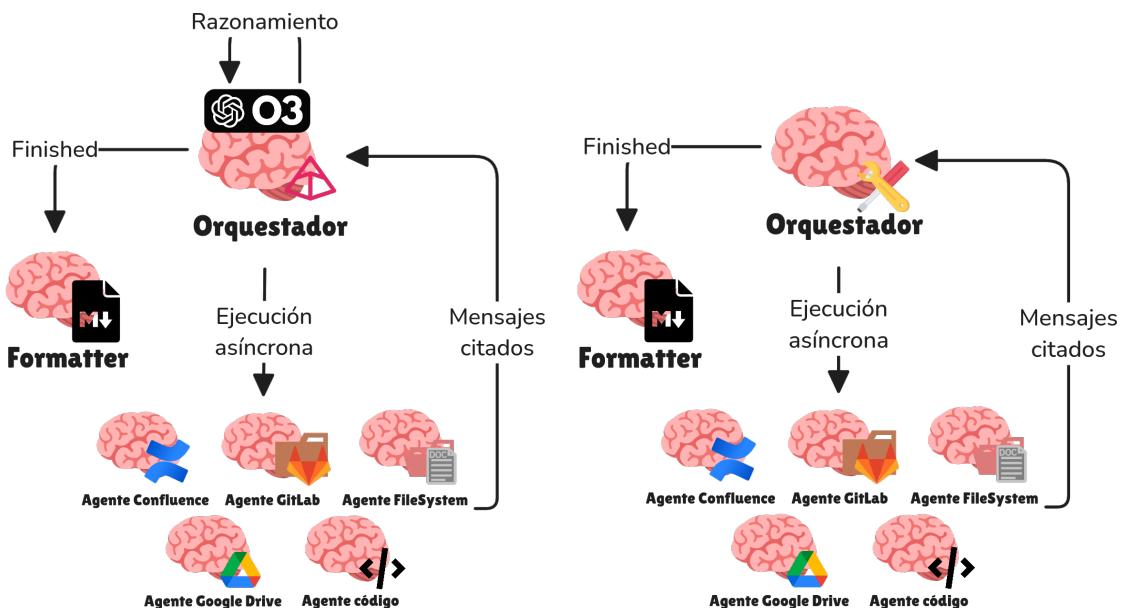
Para ilustrar el funcionamiento de cada enfoque, esta sección analiza el comportamiento de las diferentes estructuras ante la siguiente consulta del usuario: *¿Podrías proporcionar ejemplos de código donde se apliquen los principios de la guía de estilos?*

Este ejemplo requiere dos pasos lógicos secuenciales: primero, identificar los principios establecidos en la guía de estilos, lo que implica localizar y extraer el documento completo; posteriormente, analizar qué principios específicos deben buscarse y realizar una exploración en el repositorio para localizar su implementación. La respuesta deberá incluir citas tanto de la guía de estilos como de los archivos que contienen los fragmentos de código incluidos en la respuesta.

- **Sistema base:** El planificador genera un plan secuencial de dos pasos: primero localizar la guía de estilos y después identificar su aplicación en el código. Para el primer paso, el orquestador invoca al agente Confluence. El planificador evalúa el resultado obtenido y, si encuentra la guía, autoriza proceder al segundo paso; en caso contrario, modifica el plan para ampliar la búsqueda. Una vez identificada la guía, en el siguiente paso el orquestador consulta al agente de código para localizar la implementación de los principios específicos.
- **Sistema unificado:** Este implementa un procedimiento similar al sistema básico, pero integrando ambas funciones en cada iteración del orquestador. El agente orquestador establece los dos pasos a ejecutar y, sin delegar en un agente externo, indica directamente la ejecución



(a) Sistema con planificador y orquestador independientes



(b) Sistema con planificador y orquestador fusionados

(c) Sistema con orquestador como agente ReAct

Figura 7.2: Variaciones de orquestación principales experimentadas

del agente Confluence para el primer paso. Posteriormente, analiza los resultados obtenidos, modifica el plan de ser necesario y ejecuta el siguiente paso.

- **Sistema con orquestador autónomo:** El orquestador recibe la consulta y, sin planificación previa, debe determinar qué agente debe invocarse para obtener la guía de estilos. Tras procesar el resultado, invocará al agente de código para buscar la implementación de los principios identificados.

Es relevante destacar las limitaciones del sistema con orquestador autónomo. Aunque presenta mayor simplicidad y rapidez de ejecución, previsiblemente mostrará un rendimiento inferior en escenarios complejos. Al carecer de una fase explícita de reflexión, podría, por ejemplo, invocar al agente de código sin haber localizado correctamente la guía de estilos.

8

CAPÍTULO

Mejoras introducidas

Partiendo de la arquitectura base descrita en los capítulos anteriores, en esta sección se analizan las mejoras exploradas para incrementar la eficacia del sistema multiagente.

Para ello, se han integrado tres mecanismos complementarios: un prompting con ejemplos ilustrativos (*few-shot*), un sistema de memoria persistente y un diseño adaptativo que modifica el comportamiento del sistema según la complejidad de la consulta formulada.

8.1. Prompting Few-Shot

El aprendizaje mediante ejemplos de entrada consiste en proporcionar al LLM ejemplos del comportamiento esperado para ajustar la salida del modelo. Este enfoque permite obtener mejoras de rendimiento para tareas específicas de forma flexible, modificando únicamente el prompt de entrada [69].

El comportamiento del orquestador y planificador resulta idóneo para esta estrategia, ya que existen varios escenarios donde se requiere que actúen siguiendo unos criterios específicos. Por ejemplo, se necesita que el agente planificador ajuste su plan dinámicamente en función de la información recabada. El Listado 8.1 ilustra un ejemplo donde se le instruye al agente que tiene la capacidad de razonar sobre si finalizar el plan, aún en contra de lo planeado inicialmente, omitiendo el paso de obtención de información adicional previsto. Con este fin, se le indica el contexto en el que se encuentra, la respuesta esperada y una explicación del comportamiento esperado.

Listing 8.1: Integración de ejemplos few-shot al agente planificador

```
1 examples = [  
2     {
```

```

3     "current_info": "The previous plan was to find information about X and then
4     about Y. Information about X was gathered",
5     "question": "Provide information about X and Y",
6     "plan": "Enough information for X and Y was gathered. Finished",
7     "explanation": "Dynamically adjust your plan as you go, some steps might be
8     unnecessary"
9
10    },
11    ...
12 ]
13
14 # Indicar la plantilla con la que se convertirá cada ejemplo en el prompt
15 example_prompt = PromptTemplate.from_template("\t{explanation}:\n\t\tCurrent
16 information:{current_info}\n\t\tQuestion:{question}\n\t\tPlan:{plan}")
17
18 # Aplicar la plantilla a todos los ejemplos
19 def get_planner_few_shots(examples_list: List[dict]):
20     few_shots_template = FewShotPromptTemplate(
21         examples=examples_list,
22         example_prompt=example_prompt,
23         input_variables=[],
24         suffix="",
25         prefix="Here are some abstract examples:"
26     )
27     return few_shots_template.format()
28 planner_few_shots = get_planner_few_shots(examples)

```

Los ejemplos se han redactado de forma abstracta respecto al proyecto software utilizado, evitando incluir información específica sobre fuentes de datos o agentes particulares. Esta abstracción previene el sobreajuste, que se manifestaría como una dependencia excesiva hacia patrones específicos de los ejemplos few-shot y limitaría la capacidad del agente para generalizar ante consultas que no sigan exactamente dichos patrones.

8.2. Memoria persistente

La memoria persistente consiste en proporcionar al agente información de ejecuciones anteriores para ampliar su conocimiento, simulando funciones cognitivas humanas (véase Sección 2.4.2).

Este enfoque se ha implementado en los agentes especializados con el objetivo de mejorar la búsqueda de información. De esta forma, el agente obtendrá resúmenes de respuestas anteriores que sean relevantes para la consulta actual, proporcionando información complementaria que el agente podría no haber extraído correctamente.

La Figura 8.1 ilustra el funcionamiento de dicho mecanismo. En primer lugar, se añaden al prompt del agente memorias relevantes realizando una búsqueda RAG sobre un objeto `AsyncPostgresSaver`. Este objeto representa una abstracción de LangChain para guardar elementos en PostgreSQL, similar a `PGVector` utilizado en la Sección 6.2.3. Una vez el agente ha generado su respuesta, un agente resumidor comprime el resultado en aproximadamente 75 caracteres. Este resumen se indexa en la base de datos utilizando dicha abstracción, guardando adicionalmente las citas referenciadas.

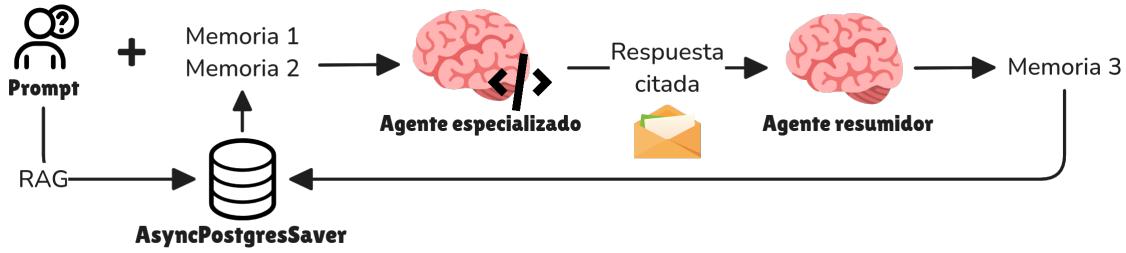


Figura 8.1: Flujo operativo del sistema de memoria de los agentes especializados

Es de destacar que las memorias recuperadas se insertan como mensajes `AIMessage` entre el mensaje del sistema y la consulta del usuario, ya que al incluirlas en el `SystemMessage` se observó que el agente tenía a ignorarlas.

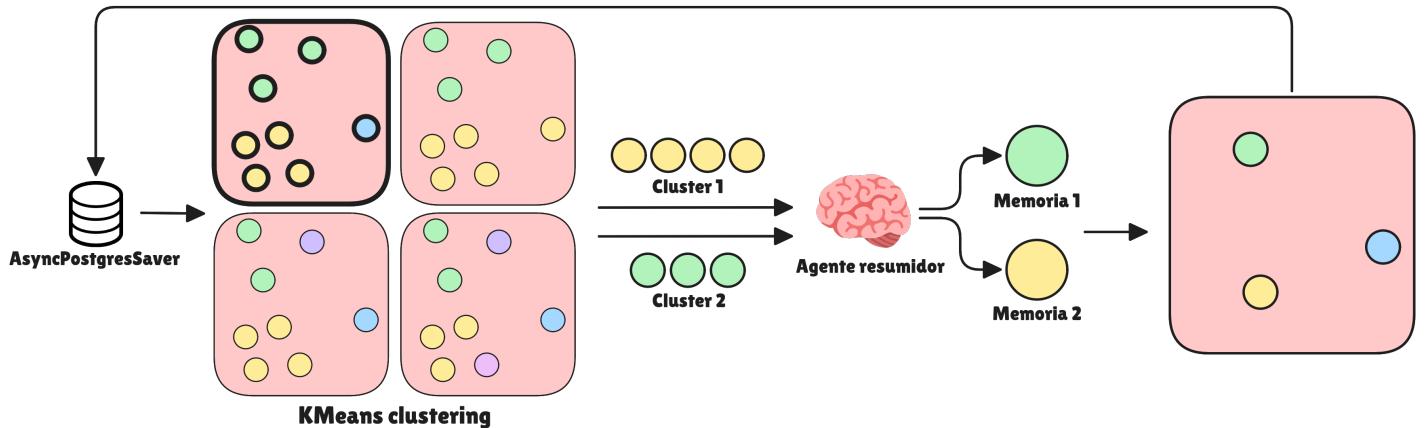
La extracción de las memorias se realiza mediante un RAG híbrido. En lugar de obtener únicamente las memorias más similares al prompt actual, se considera también las veces que estas han sido accedidas, favoreciendo las memorias más frecuentemente utilizadas. Este mecanismo actúa como un sistema de olvido, donde las memorias que no se acceden frecuentemente tienden a desvanecerse en favor de aquellas más relevantes y utilizadas. De este modo, se calcula la relevancia final considerando en un 75 % la puntuación de relevancia y en un 25 % la puntuación de acceso. Esta última se obtiene en relación a la memoria más accedida. Por ejemplo, si la memoria más accedida se ha extraído 10 veces, y la memoria actual ha sido accedida 5 veces, la puntuación para ella será de 0.5. Si la puntuación de relevancia semántica es a su vez 0.8, la media ponderada será de 0.725.

8.2.1. Agrupación de memoria

Para evitar que conceptos relevantes se pierdan ante un conjunto de memorias excesivamente grande, se ha implementado un sistema de agrupación que resume pasajes similares en memorias unificadas, replicando nuevamente el olvido de la memoria humana.

La Figura 8.2 ilustra dicho sistema. Cuando las memorias de un agente especializado específico alcanzan cierto número, se ejecuta el sistema de agrupación. Este realiza primero una agrupación semántica de las diferentes memorias en los denominados *clusters*. Para ello se utiliza el algoritmo

K-Means clustering de la librería scikit-learn¹. Este algoritmo agrupa las diferentes memorias basándose en la distancia entre todas las dimensiones de los embeddings, siendo dos memorias más similares semánticamente cuando poseen embeddings más cercanos.



Una vez distinguidos los diferentes clústeres, un agente resumidor agrupa todas las memorias de cada clúster en memorias individuales. Al ser estos similares semánticamente, debería ser capaz de abstraer los conceptos comunes en todas las memorias. Estas memorias agrupadas se añaden a la base de datos, eliminando las memorias originales del clúster.

Para determinar el número óptimo de clústeres, se ha aplicado el método del codo. Esta técnica calcula la suma de distancias entre cada elemento y el centro de su clúster para distintas cantidades de grupos, generando una curva que relaciona el número de clústeres con la distancia total (Figura 8.3b). A medida que aumenta el número de clústeres, la distancia total disminuye; sin embargo, el punto donde esta curva presenta mayor inclinación (calculado mediante su segunda derivada) representa el equilibrio óptimo.

La Figura 8.3 ilustra el agrupamiento de 12 memorias del agente de código en 3 clústeres, mientras que la Figura 8.3a presenta la simplificación de sus embeddings a un espacio bidimensional. En esta última visualización se observan los tres grupos identificados, evidenciándose su separación.

8.3. Diseño adaptativo

Tras evaluar las diferentes variaciones de orquestación (véase Capítulo 9), los resultados muestran que, aunque con un rendimiento superior, el paso de planificación añade una complejidad considerable frente al enfoque más simple. Mientras que el sistema simple es capaz de responder algunas preguntas con mayor rapidez, no consigue resolver las preguntas más difíciles. Es por ello

¹scikit-learn: <https://scikit-learn.org/stable/>

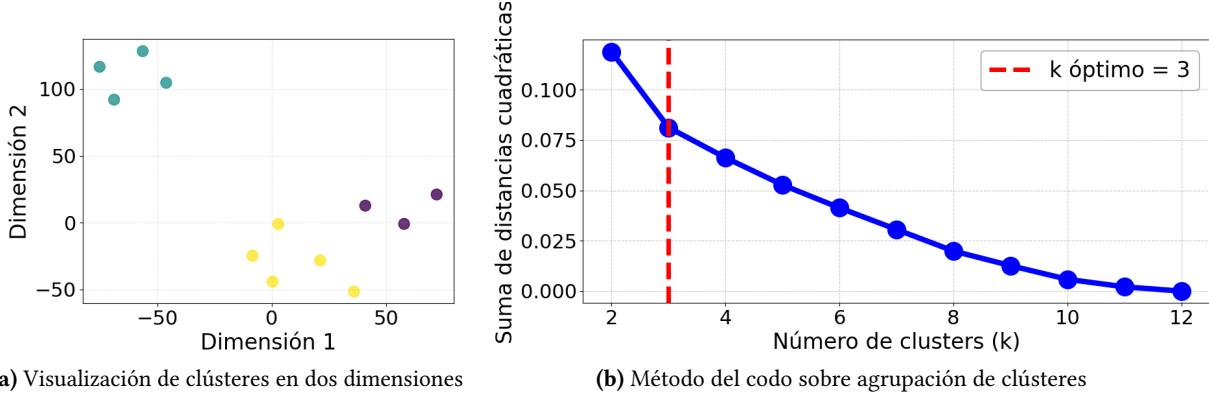


Figura 8.3: Agrupación de 3 clústeres

que un sistema híbrido surge como la solución ideal [70], utilizando la planificación para preguntas complejas y omitiéndola cuando no es necesario.

La implementación de este enfoque se ha estructurado en tres fases: la determinación de criterios para identificar preguntas complejas, la ampliación del conjunto de evaluación para mejorar la precisión de evaluación, y el análisis comparativo entre un modelo especializado y un agente clasificador. Finalmente, se procedió a la integración del clasificador seleccionado en el flujo del sistema.

8.3.1. Criterios de clasificación

Para determinar qué tipos de preguntas clasificar como difíciles, se han utilizado las evaluaciones detalladas en la Sección 9.2.2. Se consideran difíciles aquellas preguntas en las que se cumple cualquiera de estas condiciones: el rendimiento de evaluación es inferior en la orquestación simple que en la compleja, o el resultado de evaluación es menor a 0.5 en cualquiera de las dos orquestaciones. Esto resultó en un total de 19 preguntas difíciles y 27 fáciles.

Tras analizar ambos conjuntos, se evidenció que las consultas que más desafían al sistema son aquellas cuyas fuentes de información no son fácilmente accesibles. Por ejemplo, el sistema resuelve eficazmente preguntas genéricas de gestión, ya que determinar qué agente contiene dicha información es sencillo. Por el contrario, la información sobre módulos o implementaciones específicas es más difícil de ubicar: ¿Dónde puedo encontrar la documentación técnica actualizada para las tecnologías o herramientas específicas?. La ubicación de esta información no es clara, podría estar en la documentación general, en Confluence, o incluso en el repositorio de código.

8.3.2. Aumento de datos

Tras determinar los criterios de clasificación, se instruyó al modelo Claude Sonnet 3.7 con dichos criterios y el contexto del proyecto, expandiendo las preguntas originales a 200 ejemplos únicos.

Posteriormente, se utilizó la técnica de aumento sencillo de datos (EDA) [71] sobre dichos ejemplos. Esta consiste en realizar una serie de operaciones sobre los ejemplos originales para variar ligeramente su composición, reemplazando palabras por sinónimos, alterando el orden de algunas palabras, o eliminando palabras específicas. Para ello se ha utilizado el script² desarrollado por los autores de dicha técnica, obteniendo un total de 2000 preguntas tras el aumento.

Estos ejemplos se dividieron en tres conjuntos individuales: entrenamiento (75 %), evaluación (15 %) y pruebas (15 %). Todas las preguntas resultantes de la técnica EDA para una sola pregunta original se mantuvieron en el mismo conjunto, evitando así que variaciones de una misma pregunta aparezcan en diferentes particiones, lo cual sesgaría los resultados de evaluación. Finalmente se subieron a un dataset público en HuggingFace³.

8.3.3. Clasificación de preguntas

Para clasificar las consultas del usuario por dificultad, se han utilizado dos enfoques alternativos:

- **Agente clasificador:** Añadiendo en el prompt los criterios de clasificación y 5 ejemplos representativos de cada clase, el agente debe determinar mediante una salida estructurada la dificultad de la pregunta.
- **Modelo clasificador:** Se ha utilizado el modelo especializado en clasificación RoBERTa-base [72], en su versión entrenada con un corpus textual derivado de la biblioteca nacional de España [73]. Dicho modelo se ha ajustado con el dataset mencionado en la sección anterior. Los detalles del entrenamiento están disponibles en el anexo D, mientras que el modelo está públicamente disponible en HuggingFace⁴.

8.3.3.1. Evaluación de ambos enfoques

Ambos modelos se han evaluado con el conjunto de prueba de 300 preguntas, siendo el agente clasificador que utiliza el modelo GPT-4.1 mini⁵ ligeramente superior (91 % frente a 88 %).

Estos resultados demuestran el potencial de integrar modelos especializados en flujos agénticos. El LLM ajustado obtiene un rendimiento similar con aproximadamente dos órdenes de magnitud menor de parámetros, teniendo 125 millones frente a las decenas de miles de millones de parámetros

²Repositorio EDA: https://github.com/jasonwei20/eda_nlp

³Dataset de preguntas: https://huggingface.co/datasets/MartinElMolon/tfg_clasificador

⁴Modelo: https://huggingface.co/MartinElMolon/RoBERTa_question_difficulty_classifier

⁵GPT-4.1 mini: <https://platform.openai.com/docs/models/gpt-4.1-mini>

de modelos comparables al GPT propietario utilizado. Adicionalmente, este modelo se ejecutó en la CPU local, obteniendo un tiempo de respuesta inferior.

Cabe destacar que el enfoque con el modelo reducido presenta una serie de inconvenientes que lo hacen viable únicamente en escenarios concretos. Al depender de un entrenamiento previo, su modificación es menos flexible. Mientras que para modificar el comportamiento del LLM grande basta con ajustar el prompt, el LLM pequeño requeriría un reentrenamiento con un conjunto de datos modificado que represente los cambios a introducir.

Evaluación

Tras exponer la implementación de los diferentes módulos del sistema, este capítulo detalla su evaluación y la comparación entre las diferentes versiones propuestas.

Con este propósito, inicialmente se explica el sistema de evaluación implementado, tras lo que se exponen los resultados obtenidos y las conclusiones derivadas de dichos resultados.

9.1. Sistema de evaluación

La solución adoptada sigue los principios establecidos en la Sección 2.4.3: se evalúa de forma automatizada una serie de métricas cuantificables (Sección 9.1.1) mediante el SDK de LangSmith (Sección 9.1.2) sobre un dataset *ground truth* anotado manualmente (Sección 9.1.3) en el dominio del onboarding.

Se ha seleccionado esta metodología de evaluación considerando los múltiples tópicos a los que el sistema debe responder y todas las variaciones implementadas. Frente a una evaluación manual, esta metodología permite ejecutar en pocos minutos las 46 preguntas anotadas, posibilitando su repetida ejecución para todas las mejoras implementadas.

No obstante, la validación automática del sistema plantea un reto inherente al lenguaje natural: ¿cómo determinar si la respuesta a una pregunta es correcta? Para ello se ha utilizado un LLM como juez, estrategia que consiste en utilizar un agente evaluador cuya función consiste en valorar el resultado obtenido por el agente evaluado.

Para minimizar la tasa de error del juez, se han anotado las respuestas esperadas como listas textuales con los conceptos esenciales a incluir en la respuesta del agente. El agente juez genera una respuesta estructurada con un argumento booleano para cada concepto requerido, indicando si se incluye en la respuesta generada.

9.1.1. Métricas de evaluación

Se han implementado las siguientes métricas para la evaluación de agentes sobre las preguntas anotadas, aplicándose un subconjunto específico de estas según el tipo de agente evaluado:

Precisión del LLM juez Corresponde al porcentaje de conceptos totales que el LLM juez ha determinado que se incluyen en la respuesta, métrica fundamental para evaluar si la respuesta aborda adecuadamente la pregunta formulada.

Cabe destacar que el juez marcará como incorrecto todo concepto más abstracto que el anotado, aún si conceptualmente el significado es equivalente. Por ejemplo, si se anota que la función del proyecto es proporcionar herramientas de modelos de lenguaje generativos para el equipo de desarrolladores, una respuesta sobre proporcionar herramientas de inteligencia artificial para el equipo sería considerada incorrecta.

Precisión de herramientas Se comparan las herramientas utilizadas por un agente para responder una pregunta con el listado de herramientas anotado como ideal, lo cual permite identificar comportamientos erróneos cuando el agente no invoca las herramientas necesarias. Se calculan dos métricas posibles:

- Precisión de herramientas necesarias: indica la cantidad de herramientas necesarias utilizadas, favoreciendo el uso de estas.
- Precisión de herramientas innecesarias: indica la cantidad de herramientas innecesarias ignoradas, desfavoreciendo el uso de estas.

A modo de ejemplo, un agente podría tener 5 herramientas disponibles, de las cuales 2 son necesarias, 2 son innecesarias, y otra no se clasifica en ninguno de los dos grupos, pudiendo ser de utilidad pero no indispensable. Si este llama a 1 necesaria y 2 innecesarias, la precisión necesaria sería de 0.5, mientras que la innecesaria sería de 0.0. La media total sería de 0.25. El uso de la herramienta no clasificada no varía ninguna precisión.

Precisión de alucinación Los modelos tienden a responder las consultas realizadas aún si no contienen el conocimiento necesario para ello. Para evaluar esto, se han anotado algunas preguntas que son imposibles de responder con la información a la que el agente tiene acceso. Ante la consulta: *¿Qué flujo de despliegue continuo se utiliza?*, el agente no podrá responder correctamente porque no existe dicho flujo.

Esto es evaluado mediante un agente juez, el cual determina si la respuesta del agente realmente intenta responder a la pregunta o, por el contrario, reconoce que no se dispone de la información suficiente.

Precisión de citación Constituye la cantidad de documentos citados en la respuesta que estaban anotados como necesarios, métrica que permite evaluar el funcionamiento del sistema de citas.

Uso de tokens y costo económico Las evaluaciones de LangSmith permiten consultar el número de tokens empleados (tanto de entrada como de salida) en las evaluaciones, así como calcular su costo económico según los precios de la API correspondiente. Esta métrica resulta relevante ya que, entre sistemas de idéntica eficacia, el de menor costo será preferible en un entorno de producción.

9.1.2. Implementación de evaluación

El sistema de evaluación de LangSmith, ilustrado en la Figura 9.1, consiste en ejecutar cada ejemplo en el conjunto de datos de forma asíncrona con el agente evaluado. El resultado de cada ejecución se combina con el resultado anotado, y el evaluador correspondiente a cada métrica compara los resultados. Posteriormente, se calcula el promedio de cada métrica para todos los ejemplos de evaluación.

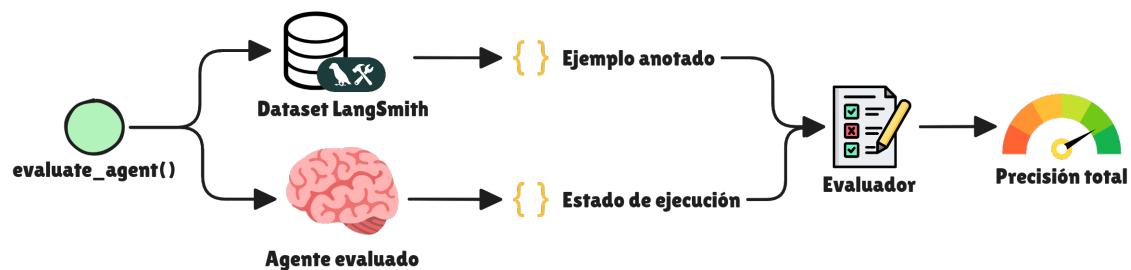


Figura 9.1: Mecanismo de evaluación de agentes

Para ello, se utiliza la función `evaluate_agent()` heredada por todos los agentes, la cual define una serie de instancias `BaseEvaluator` que representan los evaluadores a ejecutar. Este diseño permite configurar de forma modular las métricas específicas para cada tipo de agente según sus características particulares. El Listado 9.1 ilustra dicha función para los agentes especializados.

Listing 9.1: Evaluación de agentes especializados

```

1  async def evaluate_agent(self, langsmith_client: Client):
2      evaluators = [
3          ToolPrecisionEvaluator(self.get_tools_from_run_state),
4          JudgeLLEvaluator(),
5          CiteEvaluator()
6      ]
7      result = await self.call_agent_evaluation(langsmith_client, evaluators)
8      return result

```

Esta función utiliza internamente las funciones de evaluación de LangSmith mediante `call_agent_evaluation` (Listado 9.2) para ejecutar todos los ejemplos del conjunto de datos y evaluarlos con las métricas proporcionadas. El conjunto de datos de cada agente va ligado a su nombre y se obtiene mediante el cliente de LangSmith.

Listing 9.2: Llamada a evaluación de agentes

```

1  async def call_agent_evaluation(self, langsmith_client: Client,
2                                 evaluators: List[BaseEvaluator], ...):
3      ...
4      evaluator_functions = [evaluator.evaluate_metrics for evaluator in evaluators]
5      data=langsmith_client.list_examples(dataset_name=dataset_name, splits=splits)
6      run_function = self.execute_from_dataset
7
8      results = await aevaluate(
9          run_function,
10         data=data,
11         client=langsmith_client,
12         evaluators=evaluator_functions,
13         max_concurrency=max_conc,
14         experiment_prefix=evaluation_name,
15     )
16
17      return results

```

El Listado 9.3 muestra la implementación del evaluador de citas, donde se utilizan el estado de ejecución y el ejemplo anotado para calcular la precisión.

Listing 9.3: Evaluador de citas

```

1  class CiteEvaluator(BaseEvaluator):
2      async def evaluate(self, run: Run, example: Example) -> EvaluationResults:
3          run_state = run.outputs.get("run_state")
4          expected_cites = example.outputs.get("cite")
5          expected_cites_list = get_list_from_string_comma_separated_values(
6              expected_cites)
7          actual_cites = get_cites_from_state_messages(run_state)
8
8          citation_score = get_citation_score(expected_cites_list, actual_cites)
9          return EvaluationResults(
10             results=[
11                 EvaluationResult(
12                     key="cite_precision",
13                     score=StrictFloat(citation_score)
14                 )
15             ]
16         )

```

9.1.3. Dataset ground truth

La captura del conjunto de datos se ha realizado en una hoja de cálculo de Google Drive. Este documento se ha descargado posteriormente en formato de valores separados por coma (CSV) y se ha exportado como conjunto de datos a la plataforma LangSmith.

La captura comprende 46 ejemplos para el agente principal (el sistema completo) y aproximadamente 10 ejemplos para cada agente individual. Para ello se han utilizado las preguntas anotadas en la captura de requisitos, filtrando y modificando en algunos casos las preguntas para ser lo suficientemente específicas como para tener una respuesta exacta, pero a su vez con cierto grado de complejidad que requiera un razonamiento sobre la consulta. Se han utilizado las siguientes consideraciones sobre cada agente:

- **Agente principal:** comprende tres tipos de preguntas según su complejidad: consultas respondibles mediante una única fuente de datos, las que requieren múltiples fuentes, y las que necesitan múltiples fuentes en orden secuencial con dependencias entre ellas. Una pregunta del último tipo es: *¿Quién ha implementado la funcionalidad de embeddings de Mistral?*
- **Planificador:** para este agente se ha anotado la respuesta como el plan generado para un ejemplo de ejecución, incluyendo la pregunta y el estado de ejecución actual. Se han identificado tres escenarios: preguntas donde se requiere un solo paso, preguntas donde se requieren varios pasos, y preguntas a medio completar donde se requiere que el agente genere el siguiente paso o decida finalizar el plan. Por ejemplo, en el siguiente ejemplo se ha anotado también un plan a medio completar donde el agente recibe las funcionalidades marcadas por implementar, con el objetivo de que el planificador indique que se deben buscar las issues relacionadas en el siguiente paso: *¿Existe alguna issue para las funcionalidades marcadas en la documentación por implementar?*
- **Orquestador:** este agente se ha evaluado considerando los agentes seleccionados para ejecutar como herramientas. Con tal fin, se han incluido preguntas de todas las temáticas, evaluando qué agentes llamar en cada caso:
 - **Tareas de agente único:** contempla consultas diseñadas específicamente para cada agente especializado.
 - **Tareas multi-agente:** las consultas de información general y documentación requieren principalmente el `file_system_agent`, mientras que las de entorno y despliegue combinan este con el `code_agent`. Las preguntas sobre arquitectura del sistema utilizan ambos agentes, y las de gestión del proyecto emplean el `file_system_agent` junto con otros según el contexto específico. Para estándares y prácticas se añade el `confluence_agent` cuando involucran frontend, y las tareas específicas de frontend combinan los agentes `confluence_agent` y `google_drive_agent`.

- **Agentes sistema de ficheros, Confluence y Google Drive:** en los casos sencillos es necesario únicamente la herramienta de lectura de páginas. En los más desafiantes se requiere de las herramientas de búsqueda: ¿Cuáles son los colores principales y secundarios de la aplicación?
- **Agente GitLab:** incluye ejemplos donde este debe llamar primero a la herramienta de obtener información sobre los usuarios para posteriormente utilizar esos nombres de usuario como parámetros en otras herramientas. Por ejemplo, en la siguiente consulta el agente deberá buscar cuál es el nombre de usuario de “Mikel”: ¿Cuál es el id del último commit de Mikel en 2024?
- **Agente código:** en los casos sencillos los fragmentos relevantes deberían estar incluidos en el prompt de entrada, mientras que en los más complejos debería buscar información adicional: ¿Puedes mostrarme la jerarquía completa de llamadas para el método invoke_rag_with_repo en ModelTools?

9.2. Resultados obtenidos

En esta sección se presentan los resultados obtenidos mediante el sistema implementado. Inicialmente, se exponen los resultados de evaluación correspondientes a la versión inicial de cada agente, contrastados con una segunda versión mejorada. A continuación se evalúan los distintos sistemas de orquestación desarrollados. Finalmente se analizan las mejoras introducidas en el sistema. Los datos completos de todas las evaluaciones realizadas están públicamente disponibles en el repositorio del proyecto en GitHub para su consulta y reproducibilidad. Dichos datos corresponden a archivos CSV exportados directamente desde la plataforma LangSmith, que incluyen la traza completa de respuestas y los valores de precisión obtenidos para cada métrica.

Cabe destacar que los LLM utilizados no son deterministas: una misma pregunta puede generar respuestas diferentes. Además, al evaluar estas respuestas mediante otro modelo, se introduce una segunda capa de indeterminismo. Esto implica que el sistema presenta una tasa de error variable.

Los resultados mostrados corresponden al promedio de al menos dos ejecuciones. Si bien esto no garantiza certeza absoluta, sí permite afirmar que una evaluación más favorable tiene mayor probabilidad de ser una mejor solución. Esta metodología permite mejorar el sistema de manera objetiva, ya que las mejoras que generen resultados consistentemente superiores en las evaluaciones tienen mayor probabilidad de constituir avances reales.

9.2.1. Mejora de versión inicial

Primero se mejoraron los agentes especializados que presentaron deficiencias en la evaluación inicial, incorporándose posteriormente el prompting few-shot en el orquestador y planificador.

9.2.1.1. Agentes especializados

La Figura 9.2 ilustra el rendimiento de los agentes especializados según las métricas de juez, citación y alucinación, así como su gasto promedio por cada ejemplo de ejecución en el dataset.

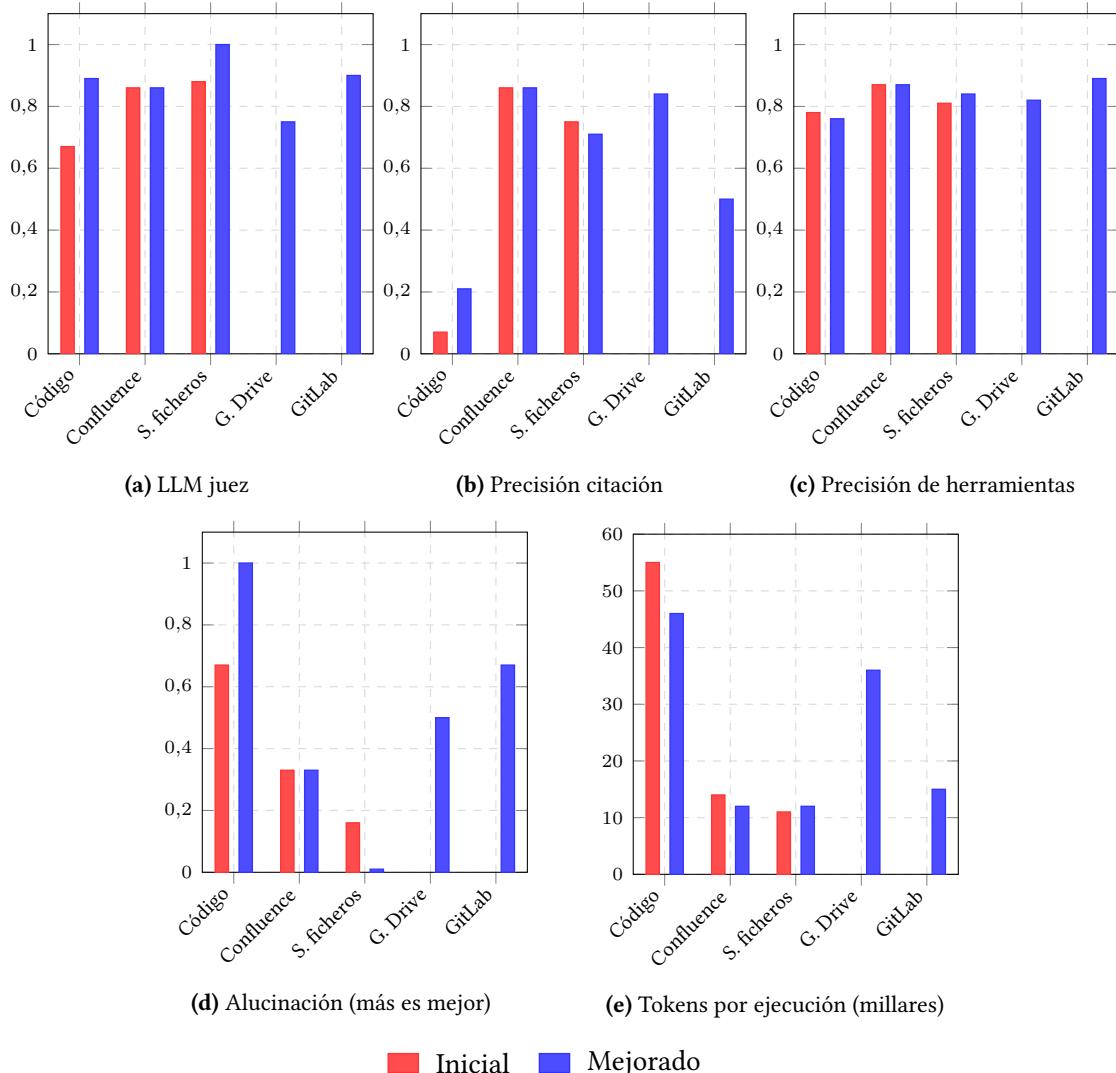


Figura 9.2: Comparación de métricas entre versión inicial y mejorada de agentes especialistas

Agente de código Este presentó un gasto considerablemente superior debido a las múltiples consultas RAG realizadas para una misma pregunta. Se ajustó el prompt y la cantidad de fragmentos extraídos, reduciendo así el coste asociado a ejecuciones prolongadas.

Agente Confluence Se identificó que la tarea de este agente consistía invariablemente en elegir qué fichero leer y responder basándose en su contenido. Se incorporó el enfoque de prompt caching, ya que al limitarse exclusivamente a leer y responder, el acceso directo a todos los documentos podría reducir los errores derivados de no leer los documentos necesarios. Aunque la precisión en todas las métricas resultó similar, este enfoque redujo el consumo de tokens y el gasto económico.

Agente sistema de ficheros Al requerir la selección entre múltiples ficheros, cuyos nombres en ciertas situaciones no describían adecuadamente todo su contenido, este agente fallaba al identificar siempre los necesarios. Por ello, la mejora implementada consistió en la inclusión de la herramienta RAG. Cuando existe incertidumbre sobre cuál fichero leer, la herramienta RAG proporciona varios fragmentos relacionados junto al nombre del fichero, facilitando así la localización de la información requerida.

9.2.1.2. Planificador y orquestador

El principal problema identificado en estos agentes radicaba en su incumplimiento del comportamiento esperado. Mientras que el agente planificador tenía a realizar pasos innecesarios en el plan, el agente orquestador invocaba un número excesivo de agentes especializados. La mejora implementada consistió en incorporar varios ejemplos del comportamiento esperado en los prompts correspondientes.

La Figura 9.3 ilustra la comparación del agente planificador y orquestador tras incorporar el prompting few-shot. Los resultados muestran una mejora considerable en el rendimiento de ambos agentes.

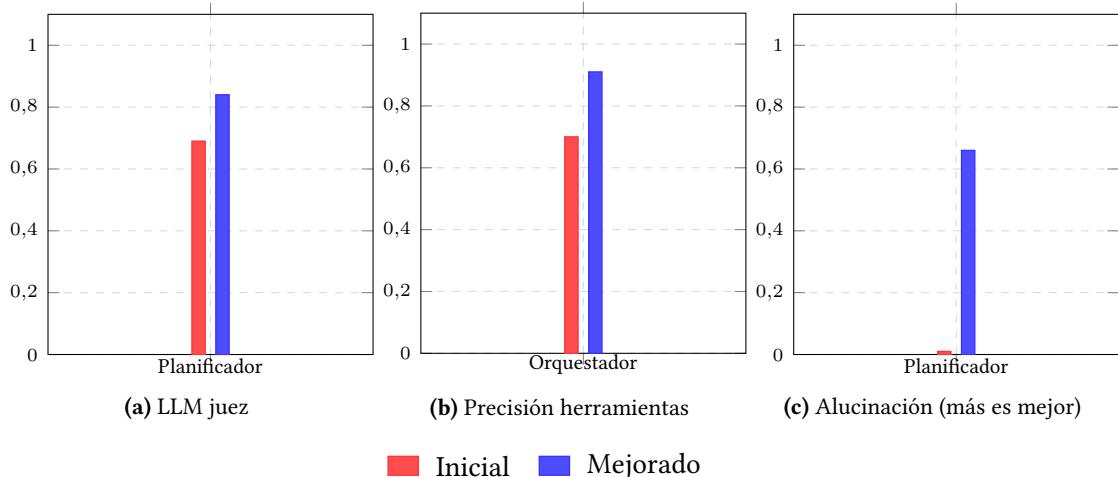


Figura 9.3: Evaluación de agentes orquestador y planificador antes y después de prompting few-shot

9.2.2. Variaciones de orquestación

Esta evaluación (Figura 9.4) compara los diferentes enfoques de orquestación: planificación dividida, planificación unificada, el sistema sin planificación y el sistema adaptativo. Adicionalmente se incluye la evaluación del sistema con planificación dividida previo a las mejoras implementadas en la sección anterior.

Los resultados muestran que las mejoras implementadas incrementaron significativamente el rendimiento del sistema mejorado en comparación con la versión inicial. El prompting few-shot en el orquestador y planificador redujo la cantidad de agentes utilizados, optimizando notablemente el costo total.

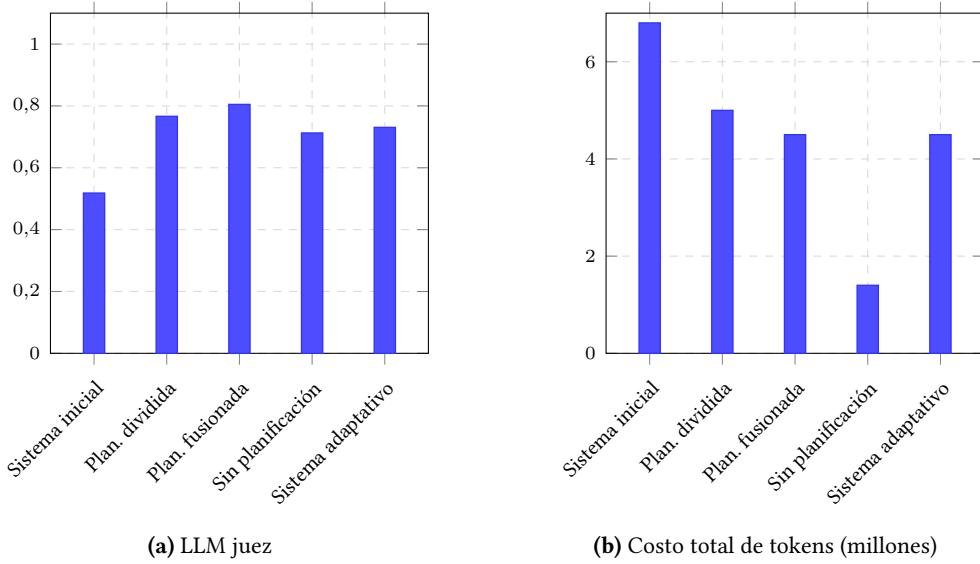


Figura 9.4: Comparación de rendimiento y costo entre diferentes variaciones de orquestación

En cuanto a la planificación, los resultados confirman que esta etapa mejora en ambos casos el sistema. Específicamente, la planificación unificada supera a la dividida. Esto indica que, en este caso de uso, la división lógica del paso de planificación no resulta eficiente, siendo preferible que el agente planificador disponga directamente de la máxima información posible para fundamentar su plan.

Finalmente, los resultados del enfoque adaptativo no cumplieron las expectativas. Aunque el rendimiento se situó en el rango esperado —superior al sistema sin planificación pero inferior a la planificación unificada— y el coste resultó menor que la planificación tradicional (al no emplear modelos razonadores pese al uso similar de tokens), la mejora económica resultó mínima mientras que la pérdida de rendimiento fue considerable. Estos resultados indican que el enfoque adaptativo no es adecuado para este caso de uso, siendo la planificación unificada la opción más eficiente.

9.2.3. Integración de memoria

Para evaluar este componente, se dividió el dataset del agente principal en un conjunto de entrenamiento (80 %) y de evaluación (20 %). Se analizó si las memorias acumuladas sobre el conjunto de entrenamiento mejoran el rendimiento en el conjunto de evaluación, ejecutando el sistema secuencialmente sobre ambos conjuntos y comparando el rendimiento con y sin memoria.

La Figura 9.5a muestra que incluir memorias de entrenamiento mejoró la precisión en el conjunto de evaluación respecto al sistema sin memoria. Sin embargo, la Figura 9.5b presenta resultados menos prometedores al evaluar repetidamente el conjunto de evaluación añadiendo memorias de los mismos ejemplos de evaluación. Esto indica que, en este caso, la memoria funciona mejor como información adicional que como memoria caché.

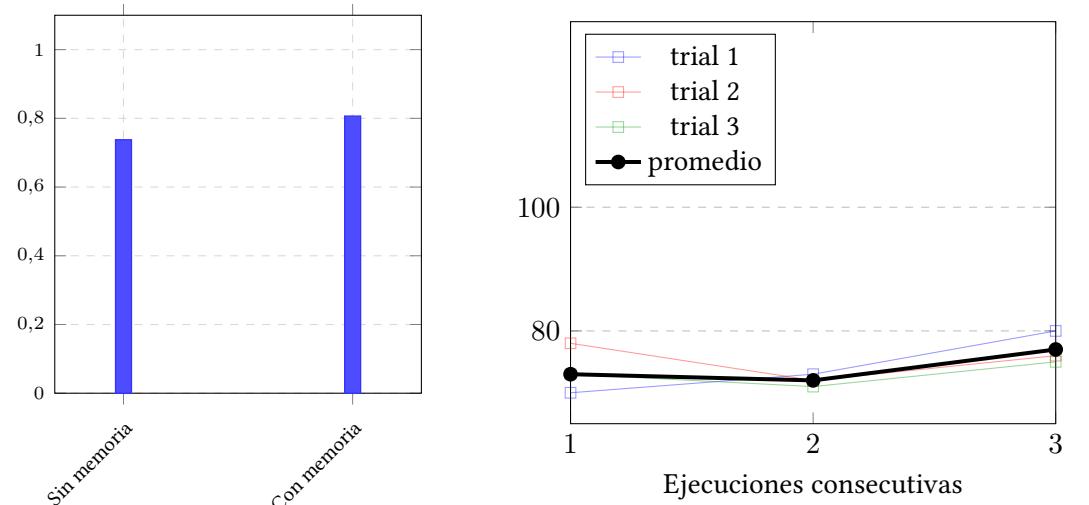


Figura 9.5: Resultados de evaluación para el sistema con y sin memoria

10

CAPÍTULO

Seguimiento y control

Este capítulo presenta el seguimiento y control del proyecto, abordando la gestión del alcance, la evaluación de riesgos y el control temporal de las actividades desarrolladas.

10.1. Gestión del alcance

Durante el desarrollo del proyecto, el alcance ha experimentado modificaciones mediante la supresión e incorporación de tareas, resultado del seguimiento bisemanal realizado con los directores del proyecto. Dado el carácter exploratorio del trabajo y el alcance inicialmente ambiguo, el proyecto se diseñó considerando dichas posibles modificaciones, siguiendo la metodología iterativo-incremental establecida en el Capítulo 3. Concretamente, las mejoras de la última iteración eran susceptibles a cambios, y se modificaron estratégicamente para no comprometer los objetivos del proyecto. A continuación se presenta un resumen de los cambios efectuados:

- **Limitación de mejoras incorporadas:** el alcance inicial contemplaba dos mejoras técnicas reducidas por limitaciones de recursos. Primero, la implementación de un sistema RAG avanzado con recuperadores especializados y su posterior evaluación. La validación de las ventajas de este sistema frente a alternativas más sencillas habría demandado una recopilación considerable de datos, por lo que el proyecto se limitó a implementar mecanismos RAG más simples. Segundo, el ajuste de un modelo de lenguaje para selección de herramientas fue sustituido por el entrenamiento de un modelo clasificador más simple, debido a la limitada dedicación horaria restante.
- **Modificación de arquitecturas de interacción alternativas:** la interacción directa entre agentes especialistas inicialmente planteada se sustituyó por una exploración más profunda de las arquitecturas de orquestación. Esta decisión se tomó al observarse que las cadenas

excesivas de agentes pueden derivar en degradación progresiva de la información y resultar excesivamente sofisticadas para el caso de uso del proyecto.

- **Sistema de citación:** esta funcionalidad no estaba contemplada en el alcance inicial, siendo incorporada al identificarse como mejora idónea para permitir al usuario conocer las fuentes de información utilizadas por los agentes en sus respuestas.

10.2. Gestión de riesgos

En este apartado se enumeran los riesgos que se materializaron en incidencias durante el transcurso del proyecto, detallando su impacto y las medidas correctivas implementadas.

10.2.1. R1-Concurrencia exploratoria

Se presentó con el hallazgo de la publicación Onboarding Buddy [14] (véase Sección 2.1.1), que proponía un enfoque similar para la creación de un sistema de agentes con una fase de planificación dinámica.

- **Impacto:** el impacto de este riesgo resultó beneficioso, dado que surgió en la fase inicial del proyecto y permitió incorporar algunos conceptos.
- **Medida correctiva:** se analizó la implementación de dicho trabajo y se incorporó un enfoque similar en el agente planificador.

10.2.2. R2-Variabilidad del alcance

Se materializó durante la fase de captura de requisitos. El desarrollo de dicha captura requirió mayor esfuerzo del previsto, ya que definir el alcance del proyecto resultó más desafiante de lo esperado.

- **Impacto:** la prolongación de esta fase inicial comprometió la dedicación horaria disponible para las fases posteriores del proyecto.
- **Medida correctiva:** se decidió recortar la fase de mejoras propuestas para evitar que el proyecto se alargara más de lo previsto o conllevara un sobrecoste horario significativo, sin comprometer los objetivos del proyecto.

10.2.3. R4-Filtrado de credenciales

Se manifestó al incluir accidentalmente la clave de acceso de OpenAI en el repositorio del proyecto.

- **Impacto:** el impacto fue nulo gracias a la segunda capa de seguridad implementada, que consistía en mantener el repositorio privado.
- **Medida correctiva:** se anuló la clave de acceso comprometida y se creó una nueva por precaución ante una posible publicación futura del repositorio.

10.2.4. R5-Pérdida de recursos

Ocurrió al inicio del proyecto cuando, por error humano, fue necesario formatear el equipo de desarrollo sin previo respaldo.

- **Impacto:** el impacto fue mínimo debido a las copias de seguridad periódicas implementadas en la nube.
- **Medida correctiva:** se recuperaron todos los recursos desde las copias de seguridad de GitHub y Google Drive.

10.3. Gestión del tiempo

Como se ha mencionado previamente, las desviaciones temporales iniciales fueron gestionadas mediante la redistribución de tareas entre iteraciones. Estos ajustes implicaron cambios tanto en las fechas como en las dedicaciones horarias.

10.3.1. Gestión de fechas

La desviación inicial supuso un retraso de aproximadamente dos semanas respecto a la planificación original. La Tabla 10.1 muestra los cronogramas de hitos reales del proyecto, destacando el fin tardío de la primera iteración. Este retraso se redujo a 10 días al finalizar la iteración 4, mientras que la última iteración concluyó según la planificación inicial.

Hito	Fecha estimada	Fecha real
Inicio del proyecto	25/02/2025	25/02/2025
Fin Iteración 1	20/03/2025	2/04/2025
Fin Iteración 2	01/04/2025	15/04/2025
Fin Iteración 3	15/04/2025	29/04/2025
Fin Iteración 4	29/04/2025	09/05/2025
Fin Iteración 5	27/05/2025	30/05/2025
Fin memoria	14/06/2025	Por determinar
Defensa del proyecto	Por determinar	Por determinar

Tabla 10.1: Cronograma de Hitos del Proyecto

Aunque en esta última iteración habría sido posible implementar mejoras adicionales, se decidió centrar los esfuerzos en la redacción de la presente memoria, dado que los objetivos del proyecto habían sido alcanzados y el tiempo invertido en tareas técnicas era ya significativo.

10.3.2. Dedicaciones

La Figura 10.1 ilustra la dedicación horaria final de cada tarea. Destacan dos sobrecostes principales: en la primera y cuarta iteración. Estas desviaciones motivaron la modificación del alcance, disminuyendo la dedicación en la quinta iteración para compensar el exceso. El sobrecoste final se atribuye a una dedicación superior a la estimada para la redacción de la memoria, que resultó considerablemente más costosa de lo previsto.

El exceso horario en la primera iteración se debe principalmente a la dificultad para definir el alcance del sistema, lo cual implicó 16 horas adicionales o un 45 % de incremento.

La segunda y tercera iteraciones transcurrieron según lo planificado. Es destacable que la implementación del sistema de evaluación fue más sencilla de lo previsto, con 7 horas o un 35 % menor de dedicación, dado que el marco de evaluación de la librería LangSmith resultó conveniente e intuitivo. En cambio, la captura de datos de evaluación fue más compleja de lo esperado, requiriendo 8 horas adicionales o el 160 % de la tarea original.

La cuarta iteración fue también más compleja de lo esperado (+22 horas, +62 %), ya que se requirieron más correcciones del comportamiento inicial de lo previsto, necesitando 10 horas adicionales (60 %). La evaluación del sistema experimentó una desviación significativa de 10 horas o el 200 % de lo estimado, debido a que, aunque la evaluación estaba automatizada, fue necesario analizar las ejecuciones para sacar conclusiones y determinar los puntos a mejorar.

La dedicación a la última iteración fue aproximadamente la mitad de lo esperado debido a la reducción del alcance. El exceso de dedicación para la memoria ascendió a 30 horas o el 50 %, considerado razonable para la redacción de un documento que satisfaga los estándares académicos.

El sobrecoste total ha sido de 19 horas, equivalente al 5 % del proyecto.

Iteración	Paquete de trabajo	Tarea	Dedicación estimada (h)	Dedicación real (h)	Diferencia
1ª Iteración	Captura de requisitos(CR)	Definición de requisitos principales	7	15	+8
	Recopilación de recursos disponibles (RR)	Elicitación de requisitos	3	3	0
	Definición del alcance del sistema de agentes (DA)	Selección del proyecto software	5	7	+2
		Generación de recursos	5	6	+1
		Investigación de arquitecturas del estado del arte	10	15	+5
		Búsqueda de proyectos parecidos	5	5	0
SUBTOTAL			35	51	+16
2ª Iteración	Sistema de agentes (SA)	Diseño del sistema	10	10	0
	Pruebas y evaluación (P)	Implementación de agentes especializados	35	40	+5
		Implementación de sistema de comunicación mínima	10	7	-3
		Pruebas automatizadas	5	6	+1
		Integración continua	3	3	0
		SUBTOTAL	63	66	+3
3ª Iteración	Sistema de agentes (SA)	Sistema de evaluación	20	13	-7
	Pruebas y evaluación (P)	Captura de datos de evaluación	5	13	+8
		Evaluación del sistema mínimo	5	7	+2
		SUBTOTAL	30	33	+3
	Sistema de agentes (SA)	Mejora de agentes	15	25	+10
	Pruebas y evaluación (P)	Variaciones de orquestación	15	20	+5
SUBTOTAL			35	57	+22
5ª Iteración	Sistema de agentes (SA)	Arquitecturas de interacción alternativas	15	0	-15
	Pruebas y evaluación (P)	Módulos de memoria	15	15	0
		Agentes RAG avanzados	15	0	-15
		Ajuste de modelo	40	15	-25
		Evaluación comparativa	10	10	0
		SUBTOTAL	95	40	-55
	Gestión (G)	Planificación (PL)	15	15	0
		Seguimiento y control (SC)	15	15	0
		SUBTOTAL	30	30	0
	Trabajo académico (T)	Memoria (M)	60	90	+30
		Defensa (D)	10	10	0
		SUBTOTAL	70	100	+30
Horas totales:			358	377	+19

Figura 10.1: Comparación de dedicación horaria estimada y real

CAPÍTULO

11

Retos

Habiendo finalizado la implementación del sistema, este capítulo expone los principales desafíos que el proyecto ha presentado, abordando tanto retos de análisis y diseño como consideraciones técnicas.

11.1. Análisis y diseño

Los agentes LLM ofrecen múltiples aplicaciones posibles, con numerosos trabajos explorando técnicas diversas. Dado este extenso espectro, uno de los principales retos consistió en determinar qué objetivos concretos perseguir dentro del ámbito del onboarding y cómo abordar su desarrollo.

Este desafío se manifestó inicialmente al establecer el alcance, donde se delimitó el objetivo específico en torno a la asistencia con las diversas cuestiones que puede enfrentar un nuevo desarrollador. Posteriormente, surgió al seleccionar las arquitecturas: aunque el director empresarial estableció la necesidad de un agente coordinador que delegase tareas a componentes especializados, la elección del enfoque concreto formaba parte del problema a resolver. La solución implicó explorar diferentes enfoques que combinaran técnicas de orquestación y planificación, junto con la implementación de un sistema adaptativo y la integración de memoria de ejecución.

Diseño del software Con el fin de desarrollar un sistema extensible, se estableció un diseño pre-meditado antes de proceder con cualquier implementación. Destacan especialmente la arquitectura distribuida para superar las limitaciones de ventana de contexto mediante agentes especializados, el enfoque híbrido que combina composición mediante grafos de LangGraph con herencia orientada a objetos para evitar duplicación de código, y el sistema de citas dinámico que garantiza la validez de los documentos referenciados.

11.2. Comportamiento de agentes

Deducir la causa del comportamiento de los agentes tras su evaluación constituyó otra dificultad. A diferencia de las excepciones tradicionales, donde un error se localiza en un punto específico del código, la conducta de un LLM resulta de múltiples variables.

Aunque es posible inferir las causas específicas mediante el análisis del estado interno del modelo, los LLM utilizados son de caja negra. Por tanto, el proceso de depuración se asemejó más a un análisis forense, requiriendo reflexión ante comportamientos inesperados: ¿por qué no citas las fuentes?, ¿por qué llamas a esa herramienta?, ¿por qué NO llamas a esa herramienta?, ¿por qué ignoras la memoria?

Respecto a la última cuestión, al evaluar el mecanismo de memoria, ciertos ejemplos no mostraban mejoras. El análisis reveló que el agente ignoraba las memorias incluidas en el prompt del sistema junto a las demás instrucciones, presumiblemente porque la extensión del prompt reducía la atención hacia estos fragmentos específicos. Para solucionar este problema, se reimplementaron las memorias como mensajes individuales, aprovechando así la optimización conversacional de estos modelos.

11.3. Desafíos técnicos

Se han utilizado una variedad de tecnologías que han conllevado también desafíos de implementación, dado que el uso de este conjunto en específico en el grado universitario es más bien limitado.

Manejo de conexiones La gestión de múltiples conexiones simultáneas presentó complejidades técnicas. Para la base de datos, se evaluaron varias alternativas tecnológicas, optando por un enfoque híbrido: el ORM SQLAlchemy para consultas complejas con PGVector, y las abstracciones de LangChain para operaciones básicas de lectura e inserción. Se descartaron tecnologías web como Flask al determinar que estas capacidades no eran necesarias para los objetivos del proyecto.

El protocolo MCP introdujo retos adicionales al requerir el análisis de las modalidades de comunicación SSE y STDIO para diferentes servidores. La sincronización entre las conexiones de base de datos y las conexiones MCP demandó una arquitectura coordinada, implementada mediante un contexto asíncrono global que gestiona de forma centralizada tanto el pool de conexiones de PostgreSQL como las sesiones MCP, optimizando el uso de recursos compartidos.

Contexto asíncrono El costo computacional de los modelos LLM implica una latencia que requiere la optimización de ejecuciones. De este modo, en ciertas ocasiones han sido necesarios varios niveles de concurrencia asíncrona. Por ejemplo, a la hora de evaluar el sistema, se ejecutan 10 evaluaciones paralelas, en las cuales el orquestador puede decidir utilizar varios agentes concurrentes, los cuales a su vez pueden ejecutar varias herramientas a la vez.

12

CAPÍTULO

Conclusiones y líneas futuras

Completado el ciclo de desarrollo del proyecto, este capítulo ofrece una síntesis reflexiva del trabajo realizado. Se presentan las conclusiones principales y el repaso de los objetivos, junto con las lecciones aprendidas durante el proceso. Finalmente, se abordan las líneas futuras relacionadas con el proyecto.

12.1. Conclusiones

En este apartado se exponen las conclusiones respecto al uso de agentes LLM en el proceso de incorporación a proyectos software. Primero se presenta una valoración global, para posteriormente extraer conclusiones acerca del protocolo MCP y los mecanismos de orquestación utilizados.

12.1.1. Líneas generales

En términos generales, los agentes han demostrado la capacidad de responder con información valiosa a las preguntas formuladas, obteniendo en ocasiones una precisión superior al 80 % sobre el dataset derivado de la elicitation de preguntas. Constituyen una herramienta eficaz para buscar y sintetizar grandes cantidades de información dispersa en múltiples fuentes (código fuente, documentación, sistemas de gestión de tareas), tarea de gran importancia en procesos de incorporación que requieren una fase de aprendizaje.

No obstante, estas herramientas presentan limitaciones que impiden considerarlas fuentes de información irrefutables. Las evaluaciones revelan que en determinadas ocasiones, los agentes no logran identificar todas las fuentes de información disponibles, además de ocasionalmente generar alucinaciones. Por este motivo, el presente proyecto propone utilizar dichas herramientas como sistemas de búsqueda y fuentes de información complementarias, requiriendo siempre la supervisión y validación humana.

12.1.2. Protocolo MCP

Esta especificación ha demostrado ser de gran utilidad para el proyecto, por la flexibilidad que proporciona emplear un protocolo común para todas las herramientas, su eficacia para integrar herramientas desarrolladas por terceros, y su facilidad de uso mediante un SDK que permite desarrollar clientes y servidores con una dedicación mínima. Dichos beneficios explican su rápida integración en el sector desde el inicio del presente proyecto^{1,2,3}, consolidándose como un estándar robusto cuya adopción resulta recomendable.

Por otro lado, al tratarse de un protocolo relativamente reciente, la disponibilidad de servidores MCP para herramientas específicas puede resultar limitada, requiriendo en ocasiones el desarrollo de adaptadores personalizados.

12.1.3. Orquestación de agentes

Los resultados de evaluación permiten reflexionar sobre la división de la comunicación entre agentes. Parece evidente que los agentes cuya función consiste en extraer información sobre una temática concreta no deberían disponer de información sobre la perspectiva general de la consulta pertinente. Sin embargo, la cuestión radica en determinar cuánta información deberían disponer los agentes encargados de distribuir la ejecución del sistema.

El presente proyecto ha concluido que cuanta más información contenga el agente decisor, más fundamentadas resultarán sus decisiones. No obstante, al degradarse la precisión de los LLM conforme aumenta la información de entrada, esta estrategia no resulta viable para los casos más complejos. Queda pendiente explorar, entonces, el punto óptimo donde dividir la fase de orquestación o decisión en varias etapas independientes.

12.2. Objetivos del proyecto

A continuación se enumeran los objetivos establecidos al inicio del proyecto junto a su grado de cumplimiento:

- **Estudio de arquitecturas agénticas:** este objetivo comprendió el análisis de diversas arquitecturas presentadas en la Sección 2.4, evaluando varias implementaciones de RAG, diferentes variaciones de planificación y orquestación, y un módulo adicional de memoria.
- **Desarrollo de sistema de Onboarding:** el sistema propuesto ha sido implementado y evaluado satisfactoriamente. Para ello, se ha seguido en la medida de lo posible la metodología

¹Google recomienda MCP: <https://cloud.google.com/blog/products/ai-machine-learning/build-multilingual-chatbots-with-gemini-gemma-and-mcp>

²MCP en Windows: <https://blogs.windows.com/windowsexperience/2025/05/19/securing-the-model-context-protocol-building-a-safer-agentic-future-on-windows/>

³MCP en OpenAI Agents SDK: <https://openai.github.io/openai-agents-python/mcp/>

de trabajo de LKS Next, utilizando los recursos propietarios disponibles y la elicitación de preguntas realizada para su evaluación.

- **Integración del Model Context Protocol:** la implementación fue diseñada para la incorporación de dicho protocolo, desarrollando varios servidores y clientes MCP, y utilizándolo para las herramientas de los agentes especializados.
- **Evaluación de agentes:** se ha desarrollado el sistema de evaluación automatizado expuesto en el capítulo 9, mediante la librería LangSmith y un conjunto de datos anotado manualmente, incorporando varias métricas personalizadas.
- **Valoración de ajuste de agentes:** fue cumplido ajustando y evaluando un modelo específico para su integración en el flujo agéntico (véase Sección 8.3). Dicha valoración ha proporcionado un ejemplo de integración sencilla que permite entrever el potencial detrás de este tipo de optimizaciones.

12.3. Lecciones aprendidas

El desarrollo del presente proyecto ha proporcionado una valiosa lección sobre la gestión de proyectos. Esta se fundamenta en la distribución eficiente del tiempo para focalizar los esfuerzos en los aspectos más relevantes para el cumplimiento de objetivos.

Es habitual considerar un proyecto como una obra personal y, por tanto, buscar optimizar todos sus aspectos para realizar el mejor trabajo posible. Dicha filosofía puede conllevar centrar un esfuerzo excesivo en detalles que, aunque importantes, no tienen el impacto final que otros aspectos sí poseen. Por ello, los esfuerzos deben centrarse en las tareas que resulten óptimas para la consecución de objetivos, ya que, independientemente de la dedicación horaria, los recursos disponibles son siempre limitados.

Por ejemplo, en este proyecto se invirtió una cantidad considerable de tiempo investigando e implementando un agente de código cuya funcionalidad RAG considerase la estructura del código fuente. Si bien esto contribuyó al cumplimiento de los objetivos del proyecto, desde una perspectiva general, un agente más simple podría haber realizado la misma labor exploratoria.

12.4. Líneas futuras

A continuación se enumeran algunas posibles direcciones de investigación en el ámbito de los agentes LLM aplicados al Onboarding:

Evaluación en diferentes proyectos software Una evolución fundamental del proyecto consistiría en evaluar la eficacia del sistema implementado aplicándolo a otros proyectos software con características diferentes al utilizado en este trabajo. Esta validación permitiría determinar la

generalización del enfoque propuesto, identificar posibles limitaciones relacionadas con el dominio específico, y ajustar la metodología para optimizar su aplicabilidad en diversos contextos.

Exploración de embedders especializados Este proyecto ha realizado una indexación del código junto a su contexto y documentación para una captura semántica contextualizada. Sin embargo, otras estrategias proponen el uso de sistemas de vectorización especializados. Por ejemplo, recientemente Mistral ha presentado un codificador específicamente ajustado para la captura semántica de código [74].

El ajuste fino de un modelo de codificación para su uso agéntico resulta más preciso que uno genérico [75, 76, 77], pero requiere de un trabajo adicional. Una exploración subsiguiente podría comparar la estrategia del proyecto actual, un embedder especializado en código, y un embedder especializado en tareas de Onboarding.

Optimización de orquestación En la Sección 12.1.3 se ha especulado sobre un punto de inflexión donde separar la fase de planificación en varias etapas conllevaría beneficios. Otro proyecto podría explorar un sistema más complejo, con preguntas más amplias, que genere reportes detallados. Este enfoque seguiría el modelo de sistemas como Deep Research⁴ de OpenAI, aplicándolo sobre repositorios empresariales que involucren diferentes proyectos, o incluso toda la información disponible de la empresa.

Ajuste de agentes En una línea de investigación complementaria, otro trabajo podría explorar el ajuste de modelos para tareas agénticas especializadas en el Onboarding. Esto incluiría entrenar un modelo para buscar en la base de conocimientos de la empresa, o incluso entrenarlo directamente sobre dicha base de conocimientos de modo que incorpore la información sin requerir búsquedas externas.

⁴Deep Research: <https://openai.com/index/introducing-deep-research/>

Definición de términos técnicos

A.1. Conjunto de datos etiquetados

Un conjunto de datos etiquetado es una colección estructurada de información donde cada elemento o instancia está asociado a una o más categorías, clases o valores objetivo, denominados etiquetas. Estas etiquetas representan la información que se desea predecir o clasificar mediante un modelo de aprendizaje automático.

En el contexto del aprendizaje supervisado, estos conjuntos constituyen la base para el entrenamiento de modelos, ya que proporcionan ejemplos concretos de la relación entrada-salida que el algoritmo debe aprender a generalizar.

A.2. Entrenamiento de redes neuronales

El entrenamiento de una red neuronal consiste en un proceso iterativo de modificación de los pesos de las conexiones entre neuronas artificiales. Estos ajustes permiten que la red aprenda a generalizar a partir de los datos de entrenamiento, extrayendo patrones subyacentes que podrá aplicar posteriormente a datos no observados.

En el aprendizaje supervisado, específicamente durante el ajuste fino, los pesos se modifican comparando las predicciones del modelo con los datos de referencia. Esta comparación se cuantifica mediante una función de pérdida, cuyos gradientes, calculados mediante la regla de la cadena, indican cómo deben ajustarse los pesos para minimizar el error. Este mecanismo de retropropagación permite que la red optimice progresivamente su capacidad predictiva.

A.3. Distancia coseno

La distancia coseno es una medida que cuantifica la similitud entre dos vectores basándose en el coseno del ángulo que forman, independientemente de sus magnitudes. Matemáticamente se expresa como:

$$\text{Similitud_coseno}(x, y) = \frac{x \cdot y}{|x||y|}$$

El valor 1 indica vectores perfectamente alineados (máxima similitud), 0 representa vectores perpendiculares (sin similitud) y -1 señala vectores en direcciones opuestas (máxima disimilitud).

A.4. Tokenizador

Un tokenizador es el componente algorítmico encargado de segmentar el texto en unidades mínimas procesables (tokens), implementando reglas específicas de división basadas en espacios, puntuación, subpalabras o patrones predefinidos según el modelo de lenguaje.

A.5. Pool de conexiones asíncronas

Un pool de conexiones asíncronas en PostgreSQL constituye un mecanismo de gestión eficiente que mantiene un conjunto predefinido de conexiones a la base de datos. Cuando una aplicación requiere conectarse a la base de datos, en lugar de crear una nueva conexión, solicita una al pool, que le proporciona una de las conexiones ya establecidas y disponibles. Su naturaleza asíncrona permite ejecutar operaciones sin bloquear el hilo principal, reduciendo la sobrecarga asociada al establecimiento repetitivo de conexiones.

A.6. Tabla de cierre (*Closure Table*)

Una tabla de cierre constituye un patrón de diseño de base de datos utilizado para representar y consultar eficientemente estructuras jerárquicas. Este enfoque almacena todas las relaciones ancestro-descendiente posibles en una tabla auxiliar, incluyendo tanto las relaciones directas (padre-hijo) como las indirectas (ancestro-descendiente a cualquier nivel).

En este proyecto, la tabla `Ancestor` implementa este patrón para representar la jerarquía del sistema de ficheros. Incluye las columnas `ancestor_id` y `descendant_id` que referencian entradas de la tabla `FileSystem`, almacenando todas las relaciones jerárquicas entre directorios y ficheros. Esta estructura permite consultar eficientemente todos los ficheros contenidos dentro de cualquier subdirectorio mediante una simple operación de filtrado por `ancestor_id`.

Las ventajas principales de este patrón incluyen la posibilidad de ejecutar consultas de subárboles completos mediante operaciones SQL directas, eliminando la necesidad de recursión costosa

y permitiendo la obtención de rutas jerárquicas con complejidad temporal constante. Como contrapartida, este enfoque requiere mayor espacio de almacenamiento debido a la redundancia de relaciones.

Elicitación de preguntas

Los índices numéricos han sido establecidos en base a una captura de datos que fue inicialmente ampliada y posteriormente sometida a un proceso de depuración manual, con el objetivo de incorporar exclusivamente preguntas no redundantes.

La presencia del carácter “e” en el índice denota preguntas de carácter específico vinculadas a ejemplos concretos, esto es, aquellas que precisan de un contexto adicional para su adecuada interpretación. Las preguntas identificadas con el carácter “a” en su índice corresponden a interrogantes anotadas directamente del cuestionario electrónico, preservando su formulación original sin ningún tipo de alteración.

Listing B.1: Listado de elicitation de preguntas procesadas y clasificadas

```
1 {  
2   "informacion_general": {  
3     "intencion_del_proyecto": {  
4       "1": "¿Cuál es el objetivo principal y la finalidad del proyecto?",  
5       "2": "¿Qué problema específico o necesidad resuelve este proyecto?"  
6     },  
7     "funcionalidades_del_proyecto": {  
8       "3": "¿Cuáles son las funcionalidades principales que incluye el  
9         proyecto?",  
10      "4": "¿Qué funcionalidades están explícitamente fuera del alcance del  
11        proyecto?"  
12    },  
13    "estructura_del_proyecto": {  
14      "5": "¿Cuál es la estructura organizativa general del proyecto a nivel  
15        de repositorios o subproyectos?",  
16    }  
17  }  
18}
```

```
13     "79a": "¿Cómo están organizados los módulos, componentes y paquetes  
14     dentro del proyecto?"  
15     },  
16     "contexto_de_negocio": {  
17         "6": "¿Cuáles son los requisitos funcionales detallados del proyecto?",  
18         "7": "¿Cuáles son los requisitos no funcionales (rendimiento, seguridad,  
19             escalabilidad, etc.) del proyecto?",  
20         "8": "¿Existe documentación formal del modelo de negocio o dominio? ¿Dó  
21             nde se encuentra?",  
22             "80ae": "¿Dónde puedo encontrar los requerimientos funcionales  
23             documentados para entender el problema a resolver?"  
24             },  
25             "repositorio_codigo": {  
26                 "140a": "¿Cuál es la URL completa del repositorio de código y cómo puedo  
27                     acceder a él?"  
28             }  
29         },  
30         "entorno_y_despliegue": {  
31             "guias_existentes": {  
32                 "9": "¿Existen guías o manuales de despliegue para el proyecto? ¿Dónde  
33                     puedo encontrarlas?"  
34             },  
35             "entornos_disponibles": {  
36                 "10a": "¿Qué entornos están configurados o están disponibles para el  
37                     proyecto (desarrollo, pruebas, preproducción, producción, etc.)?",  
38                 "11a": "¿Qué credenciales o permisos necesito para acceder a cada  
39                     entorno (VPN, usuarios, certificados, etc.)?"  
40             },  
41             "configuracion_entorno_desarrollo": {  
42                 "12": "¿Cuál es el proceso paso a paso para configurar mi entorno de  
43                     desarrollo local (IDE, herramientas, plugins)?",  
44                 "13a": "¿Cómo compilo y ejecuto el proyecto en mi entorno local? ¿Qué  
45                     comandos debo utilizar?",  
46                 "84a": "¿Qué IDE o editor es recomendado para este proyecto y qué  
47                     configuraciones específicas requiere?",  
48                 "85a": "¿Cómo configuro mi entorno de desarrollo para integrarlo con los  
49                     sistemas corporativos?",  
50                 "86a": "¿Cuál es el proceso completo para compilar el proyecto y  
51                     verificar que funciona correctamente?"  
52             },  
53             "despliegue": {
```

```

41     "14": "¿Qué sistema o plataforma se utiliza para el despliegue de
42     aplicaciones?",  

43     "15": "¿Cuál es el proceso detallado de despliegue, incluyendo
44     configuraciones, tecnologías y herramientas utilizadas?",  

45     "87a": "¿Existen pipelines DevOps implementados para la compilación y
46     despliegue en los diferentes entornos?"  

47   },  

48   "gestion_del_proyecto": {  

49     "equipo_comunicacion_coordinacion": {  

50       "comunicacion": {  

51         "16": "¿Cuáles son los canales oficiales de comunicación del equipo (chat,
52               email, videollamadas)?",  

53         "17": "¿Cómo puedo contactar a cada miembro del equipo y cuál es su
54               rol o área de responsabilidad?"  

55       },  

56       "roles": {  

57         "18": "¿Quién es el líder del proyecto o responsable final de las
58               decisiones?",  

59         "19": "¿Quiénes son los responsables de cada subsistema o área del
60               proyecto y cuáles son sus responsabilidades específicas?"  

61     },  

62     "reuniones_ceremonias": {  

63       "21": "¿Qué reuniones periódicas o ceremonias están establecidas en el
64               proyecto y cuál es su propósito?",  

65       "88a": "¿Con qué frecuencia se realizan reuniones de equipo o
66               seguimiento del proyecto?",  

67       "89a": "¿Cuál es la periodicidad de las reuniones (diarias, semanales,
68               etc.) y su duración estimada?",  

69       "90a": "¿Cuáles son los objetivos y entregables esperados para cada
70               tipo de reunión?"  

71     },  

72   },  

73   "metodologia_contribucion": {  

74     "23": "¿Dónde puedo encontrar las guías oficiales de contribución al
75               proyecto?",  

76     "24": "¿Cuál es el proceso completo para contribuir código al proyecto,
77               desde la asignación hasta la integración?",  

78     "25": "¿Existen tareas marcadas como 'good first issues' para nuevos
79               contribuyentes? ¿Dónde puedo encontrarlas?",  

80     "91a": "¿Cuál es el procedimiento detallado para entregar una tarea
81   }

```

```
completada (revisión, validación, merge)?"
},
"gestion_tareas_requisitos": {
  "sistema_gestion_tareas": {
    "26": "¿Qué herramienta específica se utiliza para gestionar las tareas del proyecto (Jira, Trello, GitHub Projects, etc.)?",  

    "27": "¿En qué ubicación exacta dentro del sistema de gestión están descritas las tareas pendientes?",  

    "28": "¿Cómo se identifican y clasifican las tareas por prioridad o urgencia?",  

    "92a": "¿Qué herramienta de gestión de tareas se utiliza en el proyecto y cómo accedo a ella?",  

    "93a": "¿Dónde puedo encontrar la descripción detallada de las tareas asignadas o disponibles?",  

    "94ae": "¿Qué tareas específicas tengo asignadas actualmente o debo realizar?",  

    "95a": "¿Cómo puedo identificar cuáles son las tareas más urgentes o prioritarias en este momento?"
  },
  "gestion_requisitos": {
    "29": "¿En qué sistema o plataforma se documentan y gestionan los requisitos del proyecto?",  

    "30": "¿Cuál es el proceso establecido para analizar, validar y aprobar nuevos requisitos?",  

    "96a": "¿Dónde están registrados formalmente los requisitos del proyecto (Jira, Confluence, documentos, hojas de cálculo)?"
  }
},
"informacion_cliente": {
  "97a": "¿Quién es el cliente final o usuario principal de esta aplicación y cuál es su contexto de uso?",  

  "98a": "¿Qué nivel de participación tiene el cliente en el proceso de desarrollo y toma de decisiones?",  

  "99ae": "¿Es necesario consultar directamente al cliente para resolver dudas sobre determinadas funcionalidades o requisitos?"
},
"estandares_y_practicas": {
  "estandares_nomenclatura_organizacion": {
    "33": "¿Cuáles son los estándares definidos para la nomenclatura y gestión de branches, commits y pull requests?",
```

```

94     "34": "¿Cuál es el proceso completo para entregar una tarea, desde su
95     finalización hasta la aprobación?",  

96     "100a": "¿Existe un estándar documentado para la nomenclatura de
97     branches, commits y otros elementos del repositorio?"  

98     },  

99     "estandares_codigo": {  

100       "35": "¿Para qué lenguajes de programación existen estándares de
101       codificación definidos en el proyecto?",  

102       "101a": "¿Se utiliza alguna guía de estilo o formato de código específico
103       para cada lenguaje del proyecto?",  

104       "102a": "¿Existe una estructura o arquitectura de código predefinida que
105       deba seguirse?"  

106     },  

107     "estandares_diseno": {  

108       "36": "¿Existe un sistema de diseño o guía de estilos para la interfaz
109       de usuario?",  

110       "103a": "¿Hay diseños en Figma u otra herramienta para las nuevas
111       pantallas o componentes a desarrollar?",  

112       "104a": "¿Dónde puedo encontrar la documentación sobre el diseño visual
113       y la experiencia de usuario a implementar?",  

114       "106a": "¿Cuál es el procedimiento a seguir si no existen diseños
115       definidos para un componente o pantalla?"  

116     },  

117     "practicas": {  

118       "ci_cd": {  

119         "37": "¿Qué procesos de Integración Continua y Despliegue Continuo (CI
120         /CD) están implementados?",  

121         "107a": "¿Cuál es el flujo completo de integración continua, desde el
122         commit hasta la validación?",  

123         "108a": "¿Qué herramientas específicas se utilizan para los procesos
124         de integración y despliegue continuo?"  

125       },  

126       "ci": {  

127         "38": "¿Cómo funciona detalladamente el proceso de integración
128         continua y qué validaciones incluye?"  

129       },  

130       "cd": {  

131         "39": "¿Cómo se ejecuta el proceso de despliegue continuo y qué
132         entornos abarca?"  

133       }  

134     },  

135   },

```

```
121     "aspectos_legales": {  
122         "40": "¿Qué licencias de software se utilizan en el proyecto y sus  
123             dependencias?",  
124             "41": "¿Cuáles son las consideraciones legales específicas que deben  
125                 tenerse en cuenta durante el desarrollo?",  
126                 "110a": "¿Cuál es el protocolo para gestionar adecuadamente las  
127                     licencias de componentes externos?"  
128             },  
129             "estandares_seguridad": {  
130                 "42": "¿Qué herramientas o procesos se utilizan para identificar  
131                     vulnerabilidades de seguridad en el código?",  
132                     "44": "¿Cómo se gestionan y auditán las dependencias desde la  
133                         perspectiva de seguridad?",  
134                         "112a": "¿Cómo verifico que las dependencias externas que utilice son  
135                             seguras y se mantienen actualizadas?",  
136                             "113a": "¿Cuál es el procedimiento a seguir si descubro una  
137                                 vulnerabilidad crítica en una dependencia?",  
138                                 "114a": "¿Cuáles son las mejores prácticas de seguridad establecidas que  
139                                     debo aplicar en mi código para este proyecto?"  
140             },  
141             "pruebas_calidad": {  
142                 "45": "¿Qué tipos de pruebas se realizan en el proyecto (unitarias,  
143                     funcionales, integración, rendimiento)?",  
144                     "46": "¿Cuál es la política establecida para la ejecución de pruebas ( por  
145                         commit, por merge request, al final del sprint)?",  
146                         "47a": "¿Cómo se automatizan las pruebas y qué herramientas y tecnologí  
147                             as se utilizan para ello?",  
148                             "48": "¿Cuál es el porcentaje actual de cobertura de pruebas y cuál es  
149                                 el objetivo establecido?",  
150                                 "115a": "¿Existen pruebas unitarias implementadas para el código actual?  
151                                     ¿Dónde se encuentran?",  
152                                     "116a": "¿Qué clases o métodos tienen pruebas unitarias documentadas y  
153                                         cuáles necesitan implementación?"  
154             },  
155             "documentacion": {  
156                 "49a": "¿Qué fuentes de documentación existen para el proyecto y dónde  
157                     puedo encontrarlas (API, guías, licencias, estándares)?",  
158                     "50": "¿Cuál es la estructura organizativa de la documentación del  
159                         proyecto?",  
160                         "51e": "¿Qué documentación específica debo consultar para realizar esta
```

```
tarea concreta?",  
146 "52": "¿Cuál es el proceso para modificar o actualizar la documentación  
del proyecto?",  
147 "53": "¿Cuál es el procedimiento establecido para documentar cambios en el  
código?",  
148 "118a": "¿Existe un espacio de documentación centralizado como Confluence  
para el proyecto?",  
149 "120a": "¿Es obligatorio documentar los cambios realizados en el código? ¿  
Qué nivel de detalle se requiere?"  
150 },  
151 "recursos_adicionales": {  
152 "54e": "¿Existen preguntas frecuentes o discusiones en StackOverflow  
relacionadas con este tema o tecnología?",  
153 "124ae": "¿Dónde puedo encontrar la documentación técnica actualizada para  
las tecnologías o herramientas específicas que necesito utilizar?",  
154 "123a": "¿Qué herramientas o utilidades podrían ayudarme a ejecutar mis  
tareas de manera más eficiente?",  
155 "formaciones": {  
156 "55e": "¿Qué recursos formativos están disponibles sobre estas tecnologí  
as y cuáles son más relevantes para mi tarea actual?",  
157 "122ae": "¿Dónde puedo encontrar material de formación específico para  
las tecnologías utilizadas en este proyecto?"  
158 }  
159 },  
160 "arquitectura_del_sistema": {  
161 "tecnologias_y_plataformas": {  
162 "stack_tecnologico": {  
163 "56a": "¿Cuáles son todas las tecnologías, frameworks y lenguajes  
utilizados en el proyecto?",  
164 "57": "¿Para qué componente o funcionalidad específica se utiliza cada  
tecnología del proyecto?",  
165 "129a": "¿Qué herramientas específicas se utilizan para gestionar las  
migraciones de esquemas de base de datos?"  
166 },  
167 "plataformas_disponibles": {  
168 "58": "¿Qué plataformas o herramientas de soporte están disponibles  
para el proyecto (diseño, colaboración, monitoreo)?"  
169 }  
170 },  
171 "dependencias": {  
172 "126a": "¿Qué herramientas o procesos se utilizan para gestionar las
```

```
    dependencias en este proyecto?",  
173     "127a": "¿Cuál es el procedimiento para revisar, actualizar o reemplazar  
    dependencias vulnerables o desactualizadas?"  
174 },  
175 "modelo_c4": {  
176     "nivel_1_contexto_sistema": {  
177         "actores": {  
178             "63": "¿Quiénes son los actores o usuarios que interactúan con el  
            sistema?",  
179             "64": "¿De qué manera específica interactúa cada tipo de actor con  
            el sistema?",  
180             "65": "¿Cuáles son los niveles de permiso o roles definidos para  
            cada tipo de actor en el sistema?"  
181         },  
182         "sistemas_externos": {  
183             "66": "¿Qué sistemas externos se integran o comunican con este  
            sistema?",  
184             "67": "¿Mediante qué protocolos, estándares o interfaces se conectan  
            los sistemas externos?"  
185         },  
186     },  
187     "nivel_2_contenedores": {  
188         "68": "¿Qué aplicaciones, servicios o componentes principales  
        conforman el sistema y cuál es la función de cada uno?",  
189         "105a": "¿La aplicación está diseñada para funcionar en múltiples  
        plataformas o dispositivos? ¿Cuáles?",  
190         "130a": "¿Qué estrategias o patrones se aplican para optimizar el  
        rendimiento de las consultas a bases de datos?",  
191         "comunicacion_entre_contenedores": {  
192             "69": "¿Qué protocolos, patrones o estándares se utilizan para la  
            comunicación entre los diferentes contenedores?"  
193         },  
194         "dependencias_entre_contenedores": {  
195             "70": "¿Cómo se gestionan las dependencias y el orden de inicio  
            entre los diferentes contenedores?"  
196         },  
197         "tecnologias": {  
198             "71": "¿Qué tecnologías, frameworks o bibliotecas específicas  
            utiliza cada contenedor o servicio?"  
199         },  
200     },
```

```

201     "evaluacion_y_mejora": {
202         "138a": "¿El enfoque de desarrollo actual es óptimo o existen
203         oportunidades de mejora identificadas?", 
204         "139a": "¿Cuál sería el costo en tiempo y recursos para optimizar el
205         proceso de desarrollo actual?" 
206     },
207     "nivel_3_diagrama_componentes": {
208         "72": "¿Cuáles son los componentes internos de cada contenedor y cuá
209         l es la función específica de cada uno?", 
210         "131a": "¿Cuál es la arquitectura de integración entre los
211         diferentes servicios o componentes del sistema?", 
212         "comunicacion_entre_componentes": {
213             "73e": "¿Qué patrones o protocolos de comunicación se utilizan
214         entre los componentes dentro de un mismo contenedor?" 
215     },
216     "dependencias_entre_componentes": {
217         "74": "¿Cómo se gestionan las dependencias y el ciclo de vida
218         entre los componentes de diferentes contenedores?" 
219     },
220     "tecnologias": {
221         "71": "¿Qué tecnologías, frameworks o bibliotecas específicas
222         utiliza cada componente?" 
223     },
224     "nivel_4_diagrama_codigo": {
225         "estructura_diagrama_clases": {
226             "75e": "¿Cuál es la estructura detallada de clases, interfaces y
227             objetos dentro de un componente específico?", 
228             "76e": "¿Cuál es la responsabilidad y función principal de cada
229             clase o interfaz dentro del componente?", 
230             "132ae": "¿Puedes proporcionar un diagrama de paquetes y clases para
231             entender la estructura del código?", 
232             "133ae": "¿Puedes mostrarme la jerarquía completa de llamadas para
233             este método específico?", 
234             "134ae": "¿Cuáles son los métodos más complejos o difíciles de
235             entender en el código y por qué?" 
236     },
237     "patrones_diseño": {
238         "77": "¿Qué patrones de diseño, estructuras de herencia o composició
239         n se implementan en el código?", 
240     }

```

```
229     "135a": "¿Qué patrones arquitectónicos o de diseño se utilizan en el  
230     proyecto (MVC, MVVM, etc.)?",  
231     "136ae": "¿Por qué se eligieron estos patrones arquitectónicos o de  
232     diseño específicos?",  
233     "137ae": "¿Qué arquitectura o patrones debo implementar para mi  
234     desarrollo actual?"  
235     },  
236     "principios_diseno": {  
237     "78": "¿Qué principios de diseño (SOLID, DRY) o buenas prácticas de  
238     código se aplican en el proyecto?"  
239     }
```

Actas de reuniones

A.1. Acta de reunión TFG (25/02/2025)

Fecha: 25/02/2025

Inicio de reunión: 14:00

Fin de reunión: 16:00

Lugar: Sala Zurriola

Tipo de reunión: Iniciación

Asistentes:

Maider Azanza
Aritz Galdos
Martín López de Ipiña
Beatriz Pérez Lamancha
Nerea Larrañaga
Jon Dorronsoro

A.1.1. Orden del día

1. Presentar el estado actual del proyecto Onboarding en LKS
2. Sesión de Brainstorming para el TFG

A.1.2. Resumen de la reunión

Se ha comenzado con una presentación del trabajo de Nerea en el proyecto de Onboarding en LKS. Se ha comentado la disponibilidad de una base de datos de itinerarios personalizados para el proceso de Onboarding.

Después, Aritz ha presentado sus requisitos para el TFG; una exploración de la implementación técnica de agentes LLM, para la posterior implementación en otros proyectos. A continuación, se ha procedido a una sesión de Brainstorming, donde las ideas destacadas son:

- Asistente ChatBot para la integración Onboarding en proyectos software. Se enfocaría en resolver cuestiones de dominio, arquitectura o prácticas software. Se descarta la generación de código dada su complejidad y la diversidad de herramientas existentes.
- Generador de ejercicios para formaciones de Onboarding usando agentes LLM.

La principal inquietud radica en la gestión del alcance y la validación del sistema. Se ha mencionado el riesgo de proponer un proyecto demasiado ambicioso, resultando en fracaso por falta de recursos.

A.1.3. Estado del proyecto

El proyecto se encuentra en una fase inicial, el alcance no está definido.

A.1.4. Decisiones

A.1.4.1. Decisiones adoptadas

Dedicar un primer sprint para definir los requisitos y alcance del proyecto.

A.1.4.2. Asignación de tareas a realizar

Tarea	Responsable	Fecha límite
Realizar una propuesta de alcance del proyecto	Martín	12/03/25
Diseñar una propuesta de diseño para el sistema de agentes LLM	Martín	12/03/25

Tabla C.1: Tareas asignadas (25/02/2025)

Próxima reunión: 12/03/2025

A.2. Acta de reunión TFG (12/03/2025)

Fecha: 12/03/2025

Inicio de reunión: 14:00

Fin de reunión: 14:45

Lugar: Telemático

Tipo de reunión: Seguimiento

Asistentes:

Maider Azanza
Aritz Galdos
Martín López de Ipiña
Juanan Pereira

A.2.1. Orden del día

1. Presentar frameworks de agentes y caso de uso propuesto
2. Decidir el caso de uso

A.2.2. Resumen de la reunión

Se ha comenzado con una presentación de Martín de los Frameworks orientados a agentes y su caso de uso propuesto para la aplicación. Se ha propuesto orientarlo a la conexión entre el proyecto software y una formación proporcionada por la empresa, creando ejercicios específicos de la documentación y proyecto de la empresa. Tras esto, se ha comentado que la creación de ejercicios por agentes LLM ya ha sido ampliamente investigada.

Juanan ha aclarado que la parte interesante radica en vincular el usuario con los diferentes recursos de la empresa; proyecto software, guía de estilo y código, documentación del proyecto... También se ha mencionado la necesidad de generar datos para los casos en los que se va a utilizar el sistema.

Se han comentado varias ideas técnicas para el sistema de agentes:

- Un procesado del prompt del usuario para la aclaración de la tarea a realizar. Se puede complementar con el patrón human-in-the-loop para capturar mejor la intención del usuario, similar a deepresearch.
- División del sistema de agentes por especialistas. Cada uno tendría acceso a herramientas especializadas en un dominio, pudiendo ser las herramientas otro agente.

- Módulo de memoria para guardar la información relacionada con el usuario y sus conversaciones pasadas.
- Interacción entre los diferentes agentes especialistas. Podría ser mediante una conversación entre agentes, cada uno proponiendo su propuesta específica. También es posible un tablón de tareas, donde cada agente leería las cuestiones asignadas a su etiqueta.
- Planificación a alto y bajo nivel. Primero un agente compondría las ideas generales, mientras que otro definiría más concretamente qué pasos llevar a cabo. Un sistema de feedback para la planificación convendría también.

A.2.3. Estado del proyecto

El proyecto tiene un alcance más definido, pero todavía no se tiene captura de requisitos.

A.2.4. Decisiones

A.2.4.1. Decisiones adoptadas

Dedicar una semana más a la búsqueda del caso de uso específico. Buscar recursos disponibles en la empresa para el desarrollo del proyecto. Considerar varios frameworks a la hora del desarrollo. Se debe tener en cuenta el nuevo framework de OpenAI, langflow.

A.2.4.2. Asignación de tareas a realizar

Tarea	Responsable	Fecha límite
Proponer diferentes casos de uso específicos para el sistema de agentes	Martín	20/03/25
Definir los recursos disponibles en la empresa para el desarrollo del proyecto	Aritz	20/03/25

Tabla C.2: Tareas asignadas (12/03/2025)

Próxima reunión: 20/03/2025

A.3. Acta de reunión TFG (20/03/2025)

Fecha: 20/03/2025

Inicio de reunión: 12:30

Fin de reunión: 13:10

Lugar: Telemático

Tipo de reunión: Seguimiento

Asistentes:

Maider Azanza
Aritz Galdos
Juanan Pereira
Martín López de Ipiña

A.3.1. Orden del día

1. Presentar casos de uso propuestos
2. Decidir el caso de uso

A.3.2. Resumen de la reunión

Se ha comenzado con una presentación de Martín de los casos de uso propuestos para el agente. Se han propuesto 5 posibles preguntas para el sistema: información general del proyecto, gestión del proyecto, preguntas relacionadas sobre la documentación, preguntas sobre formaciones adicionales y arquitectura del proyecto. Martín ha propuesto enfocar el agente únicamente en la arquitectura del proyecto e información general del proyecto. También ha propuesto varios posibles proyectos open source donde aplicar el sistema dada la falta de documentación en los proyectos open source de la empresa.

Posteriormente, Juanan ha propuesto no descartar los demás casos de uso, y añadirlos con un enfoque iterativo incremental. También se ha decidido buscar más proyectos disponibles en la empresa, y generar la documentación faltante con LLMs.

A.3.3. Estado del proyecto

El alcance general del proyecto está definido, los requisitos generales están definidos.

A.3.4. Decisiones

A.3.4.1. Decisiones adoptadas

Refinar los diferentes tipos de preguntas en una taxonomía para poder definir un proceso iterativo. Buscar más proyectos disponibles en la empresa para el desarrollo del sistema.

A.3.4.2. Asignación de tareas a realizar

Tarea	Responsable	Fecha Límite
Crear la taxonomía de preguntas	Martín	02/04/2025
Buscar proyectos software disponibles en la empresa para la aplicación del sistema de agentes	Aritz	02/04/2025

Tabla C.3: Tareas asignadas (20/03/2025)

Próxima reunión: 02/04/2025

A.4. Acta de reunión TFG (02/04/2025)

Fecha: 02/04/2025

Inicio de reunión: 14:00

Fin de reunión: 14:45

Lugar: Telemático

Tipo de reunión: Seguimiento

Asistentes:

Maider Azanza
Aritz Galdos
Juanan Pereira
Martín López de Ipiña

A.4.1. Orden del día

1. Presentar taxonomía generada con el mailing
2. Presentar proyecto IA-core-tools propuesto
3. Presentar agente de código propuesto e implementación actual

A.4.2. Resumen de la reunión

Se ha comenzado con una presentación de Martín sobre la taxonomía de preguntas anotadas, el proyecto IA-core-tools y el agente de código propuesto. A continuación, Aritz ha expuesto que la idea es tener un sistema mínimo implementado, para posteriormente ir añadiendo agentes y mejoras de estos.

Se ha comentado la utilidad de realizar una pequeña investigación antes de crear cada agente. Podría ser más interesante incorporar uno ya implementado al sistema y evaluar su rendimiento.

Tras esto, Juanan ha comentado la posibilidad de utilizar búsquedas por expresiones regulares o palabras clave de forma complementaria a la búsqueda semántica.

A.4.3. Estado del proyecto

El primer sprint está casi finalizado, el agente de código está en proceso de implementación.

A.4.4. Decisiones

A.4.4.1. Decisiones adoptadas

Crear una implementación del sistema mínimo con el agente de código para el siguiente sprint. Dedicar algo de tiempo a buscar implementaciones ya existentes de agentes antes de implementarlos. Utilizar Jira para la gestión del proyecto.

A.4.4.2. Asignación de tareas a realizar

Tarea	Responsable	Fecha límite
Acabar la implementación del agente de código	Martín	15/04/2025
Crear una planificación en Jira con los issues y sprints	Martín	15/04/2025

Tabla C.4: Tareas asignadas (02/04/2025)

Próxima reunión: 15/04/2025

A.5. Acta de reunión TFG (04/04/2025)

Fecha: 04/04/2025

Inicio de reunión: 12:00

Fin de reunión: 12:30

Lugar: Telemático

Tipo de reunión: Captura de recursos

Asistentes:

Aritz Galdos
Martín López de Ipiña
Juan Carlos del Valle
Uxue Reino

A.5.1. Orden del día

1. Explicar el funcionamiento del equipo de diseño de interfaces de usuario en LKS
2. Exponer la idea del TFM de Uxue

A.5.2. Resumen de la reunión

Se ha comenzado con una breve explicación del TFG de Martín. Se han explicado los documentos que podrían ser de utilidad: guías de estilo, documentación de proceso, etc.

A continuación, Juan Carlos ha explicado el proceso general en su departamento. Se parte del requisito de un cliente, para lo que se crea un diseño con herramientas como figma. El diseño se exporta posteriormente a una maqueta en HTML, para lo que el equipo de desarrollo puede utilizar como base a seguir. En función del cliente se siguen ciertos estándares de documentación, por ejemplo, para Orona el proceso se documenta en un Confluence.

Finalmente, Uxue ha hecho una breve explicación del objetivo de su TFM. La idea es eliminar por completo las maquetas HTML y generar una base directamente en el framework frontend utilizado.

A.5.3. Decisiones

A.5.3.1. Decisiones adoptadas

Aprovechar los documentos compartidos por Juan Carlos y Uxue para generar la documentación del agente correspondiente. Definir un agente con acceso a Confluence como uno de los posibles agentes a implementar.

A.5.3.2. Asignación de tareas a realizar

Tarea	Responsable	Fecha límite
Incluir los recursos compartidos en la documentación disponible para los agentes	Martín	-

Tabla C.5: Tareas asignadas (04/04/2025)

A.6. Acta de reunión TFG (15/04/2025)

Fecha: 15/04/2025

Inicio de reunión: 10:30

Fin de reunión: 11:30

Lugar: Telemático

Tipo de reunión: Seguimiento

Asistentes:

Maider Azanza

Aritz Galdos

Juanan Pereira

Martín López de Ipiña

A.6.1. Orden del día

1. Presentar Sistema mínimo de agentes actual
2. Proponer siguiente iteración como evaluación y mejora del sistema mínimo

A.6.2. Resumen de la reunión

Se ha comenzado con una presentación de Martín sobre el sistema mínimo implementado actual. Contiene 5 agentes especializados, un orquestador, un planificador y un formateador de la respuesta. Se ha hecho una demostración del funcionamiento y se han comentado algunos detalles de implementación.

Martín ha comentado la necesidad de evaluar el sistema antes de implementar mejoras, para poder evaluar si las mejoras posteriores realmente mejoran el sistema. Aritz ha ofrecido el TFG de Mikel Lonbide como ayuda para la implementación, ya que este desarrolló un benchmark para la evaluación de modelos LLM.

Se ha propuesto que el agente formateador contenga la capacidad de citar las fuentes de su respuesta.

Aritz ha comentado que le gustaría probar un agente orquestador con la capacidad incorporada de pensamiento propio. Se ha decidido incluir esta posibilidad en las mejoras del sistema actual tras implementar el evaluador del sistema.

A.6.3. Estado del proyecto

La segunda iteración está finalizada, se debe empezar con la tercera.

A.6.4. Decisiones

A.6.4.1. Decisiones adoptadas

Crear un sistema de evaluación tanto para los agentes individuales como para el sistema completo. Se deben considerar dos métricas:

- Las llamadas a las herramientas son las adecuadas.
- La respuesta del agente contiene los puntos a incluir en los datos anotados.

Incluir una herramienta de referencias en el agente formateador. Incluir un orquestador con capacidad de decisión razonada en la fase de mejoras.

A.6.4.2. Asignación de tareas a realizar

Tarea	Responsable	Fecha límite
Empezar con la siguiente iteración para evaluar y mejorar los agentes actuales	Martín	28/04/2025

Tabla C.6: Tareas asignadas (15/04/2025)

Próxima reunión: 28/04/2025

A.7. Acta de reunión TFG (29/04/2025)

Fecha: 29/04/2025

Inicio de reunión: 10:30

Fin de reunión: 11:30

Lugar: Telemático

Tipo de reunión: Seguimiento

Asistentes:

Maider Azanza
Aritz Galdos
Juanan Pereira
Martín López de Ipiña

A.7.1. Orden del día

1. Presentar el sistema de evaluación y las mejoras implementadas
2. Decidir el enfoque del proyecto para la siguiente evaluación

A.7.2. Resumen de la reunión

Se ha comenzado con una presentación de Martín del trabajo realizado, explicado el sistema de evaluación, los resultados de las evaluaciones realizadas y las mejoras implementadas. Se han explicado las mejoras del sistema de citas, los prompts de los agentes orquestadores y planificadores y la optimización del costo de los agentes especializados. También se han definido las variaciones de orquestación posibles por implementar.

Tras esto, Aritz ha indicado especial interés en las variaciones de orquestación, ya que como su evaluación está automatizada su implementación es en un principio trivial.

Respecto a la iteración de mejora de interacción de agentes, se ha comentado la problemática de que un sistema de memoria sobre ejecuciones anteriores limitaría el conocimiento del agente al momento de su registro. Para evitar esto, Juanan ha indicado que se pueden utilizar sistemas de actualización de información periódicas.

La reunión ha finalizado con la decisión del enfoque del rumbo del proyecto. La dedicación horaria ha sido mayor de lo inicialmente planificada, por lo que es necesario ajustar algunas iteraciones. La complejidad técnica del proyecto es un principio suficiente, por lo que se ha decidido reducir en gran medida la dedicación horaria de las últimas dos iteraciones y centrar el enfoque en la redacción de la memoria.

A.7.3. Estado del proyecto

La tercera iteración ha finalizado, la cuarta iteración se encuentra aproximadamente por la mitad.

A.7.4. Decisiones

A.7.4.1. Decisiones adoptadas

- Reducir la dedicación horaria en lo relacionado a la implementación técnica en las últimas dos iteraciones.
- Centrar los esfuerzos en la redacción de la memoria.

A.7.4.2. Asignación de tareas a realizar

Tarea	Responsable	Fecha límite
Crear un índice de todos los capítulos a incluir en la memoria	Martín	4/05/2025
Acabar las mejoras y los sistemas de orquestación	Martín	9/05/2025

Tabla C.7: Tareas asignadas (29/04/2025)

Próxima reunión: 9/05/2025

Detalles de entrenamiento

Este anexo documenta el ajuste fino de RoBERTa-base para clasificación binaria. Se describen la configuración del dataset, la arquitectura del modelo, las estrategias de congelación paramétrica implementadas y la metodología experimental con búsqueda bayesiana de hiperparámetros. Finalmente se evalúan los resultados y la configuración óptima alcanzada.

A.1. Configuración del Dataset y Arquitectura del Modelo

El dataset consta de 2000 ejemplos distribuidos en dos clases de clasificación. Dada su dimensión reducida, se ha optado por una división avariciosa (75 % entrenamiento, 15 % validación y 15 % prueba) complementada con valores de regularización elevados para prevenir el sobreajuste.

El modelo base seleccionado es RoBERTa-base, que contiene aproximadamente 125 millones de parámetros estructurados en tres módulos principales:

- **Embedder:** Tabla de embeddings con 38 millones de parámetros que transforma los tokens de entrada en representaciones vectoriales.
- **Codificador:** Estructura de 85 millones de parámetros distribuidos en 12 capas de atención que procesa las representaciones contextuales.
- **Clasificador:** Sistema de medio millón de parámetros compuesto por 2 capas completamente conectadas que realiza la predicción final.

A.2. Estrategias de Entrenamiento

Para evaluar distintos enfoques de entrenamiento, se han implementado cinco niveles de congelación paramétrica:

- **Nivel 0:** entrenamiento del modelo completo sin parámetros congelados.
- **Nivel 1:** congelación exclusiva del módulo de embedding.
- **Nivel 2:** congelación del módulo de embedding y las 4 capas inferiores de atención.
- **Nivel 3:** congelación del módulo de embedding y las 8 capas inferiores de atención.
- **Nivel 4:** congelación de los módulos de embedding y codificador completo, ajustando únicamente el clasificador.

Respecto a la configuración del prompt, se han evaluado tres alternativas distintas. La primera configuración consistió en una implementación sin prompt como referencia. Posteriormente se implementó un prompt breve que incorporaba explícitamente el criterio de clasificación (mostrado en el Listado D.1), lo que resultó en una mejora significativa de la precisión. En un tercer experimento, se evaluó la incorporación de tokens especiales delimitadores al inicio y final de la consulta para indicar la posición de la pregunta, pero esta aproximación fue descartada tras registrar resultados inferiores.

Listing D.1: Plantilla del prompt para ejemplos del modelo clasificador

```

1 prompt_template = """Clasificar dificultad: {consulta}
2
3 FACIL: información general, bien documentada, fundamental
4 DIFICIL: implementaciones específicas, componentes concretos, personas responsables,
   acceso no evidente"""

```

A.3. Metodología Experimental

La experimentación se ha realizado en la plataforma Kaggle¹, utilizando dos GPU T4 con 16GB de VRAM cada una. La implementación se llevó a cabo mediante el Trainer de la librería HuggingFace, incorporando el prompt optimizado descrito previamente.

El proceso completo consumió aproximadamente 15 horas de computación, distribuidas en tres búsquedas bayesianas mediante la librería Optuna². Cada ensayo individual requirió aproximadamente entre 5 y 10 minutos para completar 6 épocas de entrenamiento.

¹Kaggle: <https://www.kaggle.com/>

²Optuna: <https://optuna.org/>

Para la optimización, se han evaluado cinco hiperparámetros: tasa de aprendizaje, decaimiento de pesos, tasa de desconexión (*dropout*), pasos de calentamiento y nivel de congelación. El entrenamiento se realizó con un tamaño de lote de 32 ejemplos, resultando en 44 pasos por época con los 1400 ejemplos disponibles. Siguiendo las recomendaciones establecidas para este modelo, se ha utilizado el optimizador `adamw_torch` con un planificador lineal.

Tabla D.1: Proceso de búsqueda de hiperparámetros y resultados óptimos

Hiperparámetro	Primera Búsqueda	Segunda Búsqueda	Tercera Búsqueda	Valores Óptimos
Tasa de aprendizaje	5.0e-06 a 1.0e-05	5.5e-06 a 9.5e-06	8.3e-06 a 9.9e-06	8.5e-06
Decaimiento de pesos	1.0e-06 a 0.09	1.0e-06 a 0.1	1.0e-05 a 2.0e-04	3.44e-05
Dropout	0.10 a 0.30	0.35 a 0.50	0.39 a 0.41	0.399
Pasos de calentamiento	0 a 25	10 a 20	14 a 16	16
Nivel de congelación	0, 1, 2, 3, 4	0, 1, 2	1	1
Épocas	4	6	6	3

A.4. Resultados

La optimización de hiperparámetros se desarrolló a través de tres búsquedas bayesianas secuenciales, cada una refinando los resultados de la anterior.

En la primera iteración, se estableció un rango de búsqueda amplio con 4 épocas por ensayo y 30 ensayos totales. Durante esta fase se exploraron todos los niveles de congelación (0-4) para identificar tendencias iniciales en el comportamiento del modelo.

Para la segunda iteración, se acotaron los rangos basándose en los resultados iniciales. Se incrementó el número de épocas a 6 por ensayo y se ejecutaron 40 ensayos totales. Los niveles de congelación 3 y 4 mostraban rendimiento inferior, por lo que fueron descartados. También se observó que valores más altos de dropout mejoraban el rendimiento, por lo que se aumentó este rango respecto a la primera iteración.

En la tercera y última iteración, se refinaron aún más los rangos de búsqueda manteniendo 6 épocas por ensayo y aumentando a 50 ensayos totales. El nivel 1 de congelación (preservando únicamente el módulo de embedding) ofrecía el mejor rendimiento, por lo que esta fase se centró exclusivamente en dicha configuración.

La configuración óptima de hiperparámetros, correspondiente al ensayo 20 de la tercera búsqueda, logró una precisión del 93,67 % en el conjunto de validación y 88 % en el conjunto de prueba. Los valores específicos pueden consultarse en la columna "Valores Óptimos" de la Tabla D.1, con una tasa de aprendizaje de 8.5e-06, un decaimiento de pesos de 3.44e-05, un dropout de 0.399, 16 pasos de calentamiento, y un nivel de congelación 1 con 3 épocas de entrenamiento.