

Trabajo de Fin de Grado
Grado en Ingeniería Informática
Ingeniería de Software

Título del trabajo

Martín López de Ipiña Muñoz

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

23 de mayo de 2025

Resumen

Índice de contenidos

Índice de contenidos	III
Índice de figuras	VII
Índice de tablas	VIII
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	7
2.4. Estado del arte en arquitecturas de agentes LLM	9
2.4.1. Arquitectura RAG	9
2.4.2. Arquitecturas de interacción entre agentes	10
2.5. Agentes LLM en proyectos software	11
2.6. Ajuste de modelos para agentes LLM	13
2.6.1. Limitaciones de los modelos actuales	13
2.6.2. Enfoques de ajuste fino para agentes específicos	13
2.6.3. Técnicas de entrenamiento eficiente	14
3 Planificación	15
3.1. Alcance	15
3.1.1. Objetivos concretos del proyecto	15
3.1.2. Requisitos	16

3.1.3. Fases del proyecto	16
3.1.4. Descomposición de tareas	17
3.2. Periodos de realización de tareas e hitos	20
3.2.1. Dependencias entre tareas	20
3.2.2. Diagrama de Gantt	21
3.2.3. Hitos	21
3.3. Gestión del tiempo	22
3.3.1. Estimación de cada tarea	22
3.4. Gestión de riesgos	22
3.5. Gestión de Comunicaciones e Información	24
3.5.1. Sistema de información	24
3.5.2. Sistema de comunicación	24
3.6. Herramientas disponibles	24
4 Captura de requisitos	29
4.1. Requisitos principales	29
4.1.1. Requisitos funcionales	29
4.1.2. Requisitos no funcionales	30
4.2. Dominios de conocimiento	31
4.2.1. Anotación de preguntas	31
4.2.2. Metodología empresarial	32
4.3. Recursos a utilizar	33
4.3.1. Proyecto software	33
4.3.2. Recursos generados	33
5 Diseño del sistema	35
5.1. Estructura del proyecto	35
5.2. Diseño de agentes	37
5.2.1. Diagrama de agentes	37
5.3. Servidores MCP utilizados	40
5.3.1. Servidores SSE	40
5.3.2. Servidores STDIO	42
5.4. Cliente MCP	44
6 Agentes especializados	49
6.1. Estructura SpecializedAgent	49
6.1.1. Gestión de herramientas	51
6.1.2. Sistema de citas	51
6.2. Agentes implementados	53
6.2.1. Agente código	53
6.2.2. Agente Google Drive	56

6.2.3. Agente Sistema de Ficheros	57
6.2.4. Agente Confluence	57
6.2.5. Agente GitLab	58
7 Orquestación de agentes	61
7.1. Agente Principal	61
7.1.1. Respuesta estructurada	62
7.1.2. Agente Planificador	63
7.2. Agente orquestador	64
7.3. Variaciones de orquestación	64
7.3.1. Caso práctico	66
8 Mejoras introducidas	67
8.1. Prompting Few-Shot	67
8.2. Memoria persistente	68
8.2.1. Agrupación de memoria	69
8.3. Diseño adaptativo	70
8.3.1. Criterios de clasificación	71
8.3.2. Aumento de datos	72
8.3.3. Clasificación de preguntas	72
9 Evaluación	75
9.1. Sistema de evaluación	75
9.1.1. Métricas de evaluación	75
9.1.2. Implementación de evaluación	76
9.1.3. Dataset anotado	78
9.2. Resultados obtenidos	79
A Definición de términos técnicos	81
A.1. Diseño General	81
A.2. Conjunto de datos etiquetados	82
A.3. Entrenamiento de redes neuronales	82
A.4. Distancia coseno	82
A.5. Tokenizador	82
A.6. Pool de conexiones asíncronas	83
B Elicitación de preguntas	85
C Actas de reuniones	95
A.1. Acta de reunión TFG (25/02/2025)	95
A.1.1. Orden del día	95
A.1.2. Resumen de la reunión	96

A.1.3. Estado del proyecto	96
A.1.4. Decisiones	96
A.2. Acta de reunión TFG (12/03/2025)	97
A.2.1. Orden del día	97
A.2.2. Resumen de la reunión	97
A.2.3. Estado del proyecto	98
A.2.4. Decisiones	98
A.3. Acta de reunión TFG (20/03/2025)	99
A.3.1. Orden del día	99
A.3.2. Resumen de la reunión	99
A.3.3. Estado del proyecto	99
A.3.4. Decisiones	100
A.4. Acta de reunión TFG (02/04/2025)	101
A.4.1. Orden del día	101
A.4.2. Resumen de la reunión	101
A.4.3. Estado del proyecto	101
A.4.4. Decisiones	102
A.5. Acta de reunión TFG (04/04/2025)	103
A.5.1. Orden del día	103
A.5.2. Resumen de la reunión	103
A.5.3. Decisiones	103
A.6. Acta de reunión TFG (15/04/2025)	105
A.6.1. Orden del día	105
A.6.2. Resumen de la reunión	105
A.6.3. Estado del proyecto	105
A.6.4. Decisiones	106
A.7. Acta de reunión TFG (29/04/2025)	107
A.7.1. Orden del día	107
A.7.2. Resumen de la reunión	107
A.7.3. Estado del proyecto	108
A.7.4. Decisiones	108
D Detalles de entrenamiento	109
A.1. Configuración del Dataset y Arquitectura del Modelo	109
A.2. Estrategias de Entrenamiento	110
A.3. Metodología Experimental	110
A.4. Resultados	111
Bibliografía	113

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
3.3. Diagrama de Gantt del proyecto	21
3.1. Estructura de Descomposición de Trabajo (EDT) del proyecto	26
3.2. Dependencias entre tareas del proyecto	27
3.4. Estimación horaria de cada tarea	28
5.1. Estructura de directorios principales del proyecto	36
5.2. Diagrama de clases UML de del sistema de agentes.	38
6.1. Grafo de ejecución de agentes especializados	50
6.2. Diagrama de funcionamiento del sistema de citas	52
6.3. Diagrama relacional de la base de datos con el código fuente del proyecto software . .	54
6.4. Algoritmo de chunking considerando definiciones de funciones y clases en el código fuente	55
6.5. Comparación de agente Confluence estándar y enfocado en el cacheo del prompt . .	58
7.1. Flujo operativo del agente planificador	63
7.2. Variaciones de orquestación principales experimentadas	65
8.1. Flujo operativo del sistema de memoria de los agentes especializados	69
8.2. Agrupación de memoria por clústeres	70
8.3. Agrupación de 3 clústeres	71
9.1. Mecanismo de evaluación de agentes	77

Índice de tablas

3.1. Cronograma de Hitos del Proyecto	22
C.1. Tareas asignadas (25/02/2025)	96
C.2. Tareas asignadas (12/03/2025)	98
C.3. Tareas asignadas (20/03/2025)	100
C.4. Tareas asignadas (02/04/2025)	102
C.5. Tareas asignadas (04/04/2025)	104
C.6. Tareas asignadas (15/04/2025)	106
C.7. Tareas asignadas (29/04/2025)	108
D.1. Proceso de búsqueda de hiperparámetros y resultados óptimos	111

CAPÍTULO

1

Introducción

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 (conocidos también por el anglicismo Large Language Models o por sus siglas LLM), su arquitectura, 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 y las técnicas de ajuste de modelos.

2.1. Onboarding en proyectos software

El proceso de incorporación (onboarding) 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^[1].

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^[2].

En este contexto, los modelos de lenguaje de gran escala emergen como una alternativa prometedora para transformar el proceso de onboarding, ofreciendo orientación personalizada e

instantánea que podría reducir la dependencia de los desarrolladores senior, preservar la productividad global de los equipos y facilitar una incorporación más eficiente y menos disruptiva[3].

2.1.1. Trabajo previo

La Universidad del País Vasco, en colaboración con LKS NEXT¹, 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[4]. 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 multi-agente que organiza diversos componentes especializados para proporcionar asistencia contextualizada durante la incorporación de nuevos desarrolladores[5].

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.

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 [6].

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.

¹LKS NEXT: <https://www.lksnext.com/es/>

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 **D** de los LLM 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 **A.2**, 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 ChatGPT²) de OpenAI³, Claude Sonnet de Anthropic⁴, y Llama-Instruct de Meta⁵.

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 es típicamente alternante. Tras realizar la llamada a la herramienta, la salida de esta se utilizará como entrada para el siguiente mensaje del modelo. La Figura 2.1 ilustra el esquema de un agente con acceso a una API del clima. Como el modelo carece

²ChatGPT:<https://chatgpt.com>

³OpenAI:<https://openai.com/>

⁴Anthropic:<https://www.anthropic.com/>

⁵Meta: <https://about.meta.com/es/>

de información climática 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.

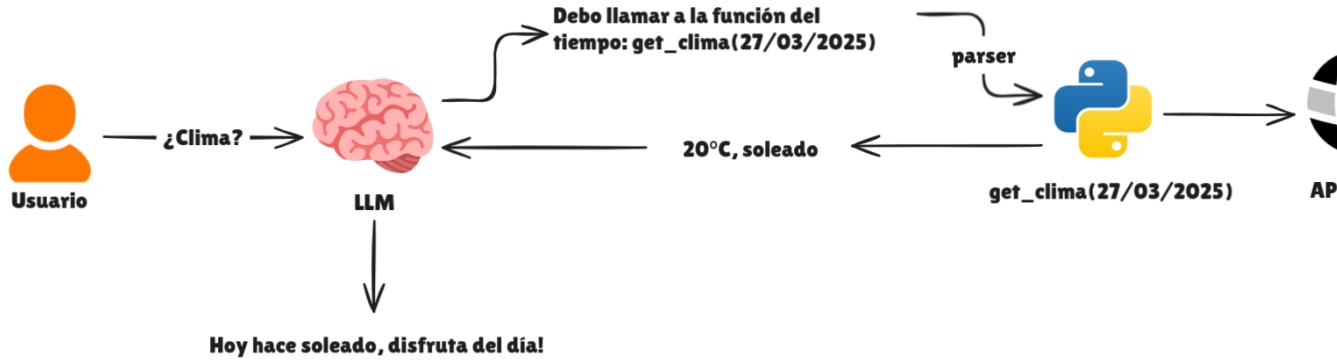


Figura 2.1: Ejemplo de interacción de un modelo LLM con una herramienta externa.

2.2.2.1. Patrón ReAct

El agente Reasoning and Act (ReAct) constituye uno de los patrones más utilizados[7]. 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 específicos A.5 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 fundamentalmente LangChain⁶ para la gestión de llamadas a APIs de modelos y prompting, LangGraph⁷ para la orquestación de flujos agénticos, y LangSmith⁸ para el seguimiento de trazas de llamadas a modelos y herramientas. Para la gestión de los conjuntos de datos y entrenamiento de modelos utilizaremos HuggingFace.⁹

2.3. Model Context Protocol

El Model Context Protocol (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[8].

La Figura 2.2 ilustra el esquema operativo del protocolo. Los desarrolladores de Jira¹⁰ y GitHub¹¹ 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.

⁶LangChain: <https://www.langchain.com/>

⁷LangGraph: <https://www.langchain.com/langgraph>

⁸LangSmith: <https://www.langchain.com/langsmith>

⁹HuggingFace: <https://huggingface.co/>

¹⁰Jira: <https://www.atlassian.com/es/software/jira>

¹¹GitHub: <https://github.com/>

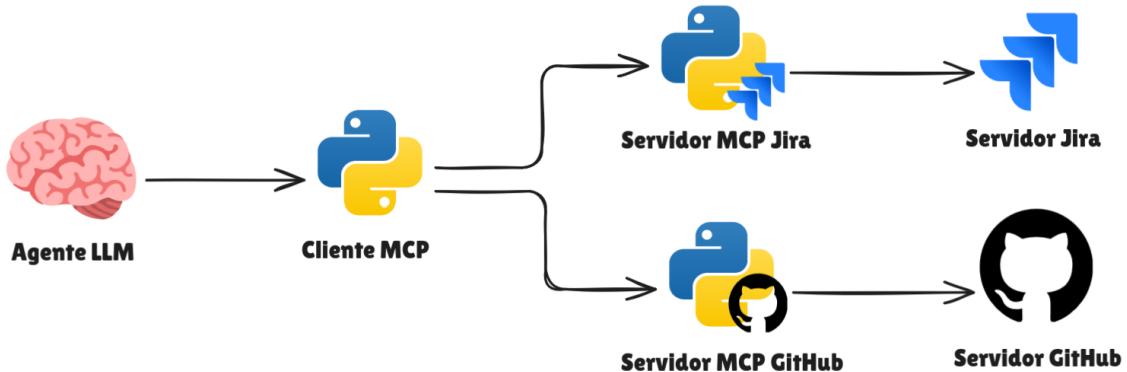


Figura 2.2: Esquema de funcionamiento del Model Context Protocol.

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 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 de escritorio Claude Desktop¹² de Anthropic constituye un reflejo del potencial del protocolo. Esta plataforma ofrece la posibilidad de interactuar con servidores mediante una configuración mínima. Implementando el protocolo STDIO, la aplicación ejecuta los servidores distribuidos por terceros a través de gestores de paquetes como uv¹³, npx¹⁴ o Docker¹⁵. Al incorporar un cliente MCP en la aplicación, consigue integrar las herramientas disponibles en la interfaz de chat con los modelos de Anthropic.

¹²Claude Desktop: <https://claude.ai/download>

¹³uv: <https://astral.sh/blog/uv>

¹⁴npx: <https://docs.npmjs.com/cli/v8/commands/npx>

¹⁵Docker: <https://www.docker.com/>

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, la arquitectura 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.

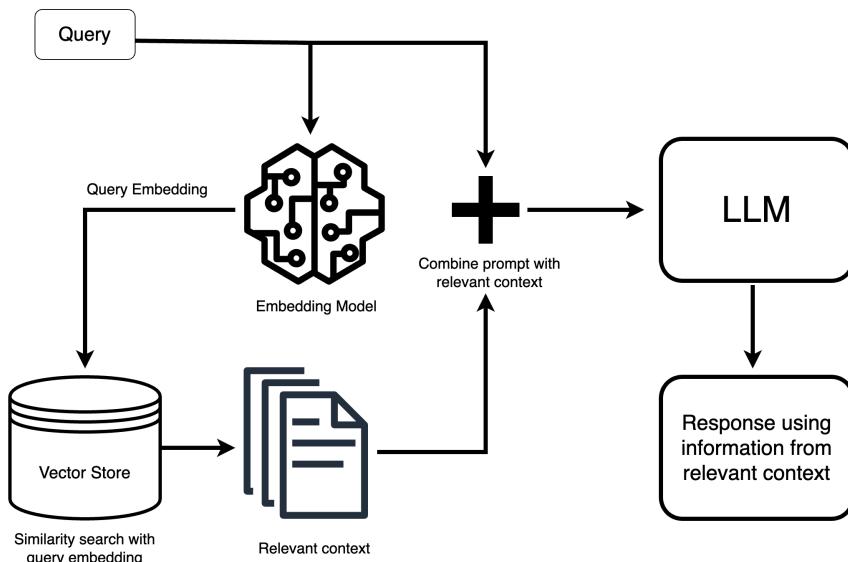


Figura 2.3: Esquema de funcionamiento de la arquitectura RAG en un LLM [Fuente](#).

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, 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 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 A.4.

2.4.1.1. Estrategias RAG avanzadas

La optimización del rendimiento en arquitecturas RAG ha sido ampliamente estudiada [9][10], enfocándose en tres áreas principales: el procesado de documentos, los sistemas de recuperación y la mejora del flujo de generación.

- **Procesado de documentos:** la calidad de la indexación documental determina la eficacia del sistema. Entre las estrategias destacadas figuran la eliminación de ruido textual, el ajuste del tamaño óptimo de los segmentos indexados, y la técnica de ventana solapada. Esta última superpone fragmentos para preservar los datos situados en las intersecciones de los segmentos.
- **Sistemas de recuperación:** los recuperadores densos basados en representaciones, ilustrados en la Figura 2.3, precalculan las representaciones vectoriales de los documentos mediante una indexación independiente de las consultas. Por otro lado, los recuperadores basados en interacción, proponen calcular el vector para cada documento junto al vector de consulta durante el tiempo de ejecución, obteniendo así una representación más rica que captura las relaciones contextuales específicas entre ambos elementos[11][12].

Para contrarrestar el sobrecoste computacional de este enfoque, se han desarrollado técnicas de indexación híbridas que combinan ambos métodos. Estas técnicas permiten realizar una búsqueda inicial utilizando representaciones vectoriales y posteriormente refinar los resultados mediante la extracción interactiva[13].

- **Optimización del flujo de generación:** para tareas que requieren reflexión documentada, se han desarrollado sistemas de salto múltiple que alternan entre recuperación y generación[14][15][16][17][18]. Estos flujos permiten al sistema examinar críticamente la información inicialmente recuperada, identificar lagunas informativas, y formular las consultas correspondientes para subsanar dichas carencias.

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[19].

- **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 de interacciones pasadas de forma resumida[20][21][22]. Esta memoria puede consultarse posteriormente mediante RAG, permitiendo recuperar los elementos más relevantes según el contexto[23].

Algunos módulos se inspiran en la estructura de la memoria humana[24], incorporando mecanismos que almacenan información con distintas temporalidades y niveles de relevancia[19][25].

- **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)[26], 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[22] lo amplían mediante la generación de múltiples cadenas de razonamiento independientes y la posterior selección de la respuesta óptima entre ellas[27][28].

De manera complementaria, las técnicas de reflexión[29][30][31] 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[32][33][34]. Esta técnica permite definir planes de alto nivel que posteriormente se desglosan en acciones específicas ejecutables por los agentes[35][36][37][38]. Adicionalmente, la definición de interdependencias entre estas acciones permite verificar la validez de los planes generados[39][40][41].

Los modelos razonadores como o1 de OpenAI o DeepSeek-R1 incorporan estas técnicas de forma nativa[42]. Estos han sido entrenados con datos que incluye ejemplos de razonamiento y planificación, lo que les permite generar respuestas que incluyen dichas estructuras.

- **Interacción entre agentes:** los sistemas multi-agente implementan una arquitectura donde diversos agentes especializados son coordinados por un componente central denominado orquestador[43][44]. 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[45][46]. Por ejemplo, ChatDev[47] establece un sistema de colaboración entre agentes programadores, testers y gestores para abordar problemas de ingeniería de software. MetaGPT[48] refina esta propuesta al implementar un protocolo de comunicación basado en el patrón publicador/suscriptor entre los agentes, permitiéndoles difundir información de forma selectiva.

2.5. 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 Copilot¹⁶ o Cursor¹⁷ ofrecen la integración de un modelo LLM ajustado para generación de código directamente al entorno del desarrollador. De esta forma, el desarrollador recibe sugerencias de autocompletado en tiempo real mientras escribe el código.

Este tipo de herramientas han demostrado ser eficaces para aumentar la productividad de los desarrolladores en tareas de programación repetitivas[49]. Sin embargo, debido a su enfoque en la rapidez de respuesta, presentan limitaciones en la aplicación de razonamiento profundo.

- **Asistentes de programación:** principalmente en aplicaciones en forma de chat, herramientas como GitHub Copilot Chat, Cursor o Cody¹⁸ de SourceGraph¹⁹ ofrecen un sistema de chat interactivo que permite al desarrollador realizar preguntas específicas sobre el código[50].

Cabe destacar el proyecto SourceGraph, el cual al ser inicialmente de código abierto ha publicado una explicación de su funcionamiento. Proporciona un sistema avanzado de navegación de código que permite la integración del agente LLM denominado Cody.

La arquitectura de este sistema opera en dos etapas: en primera instancia, los servidores de lenguaje específicos para cada lenguaje de programación analizan la estructura del código fuente, generando un grafo de dependencias que representa las relaciones jerárquicas entre los distintos elementos del código (funciones, clases, métodos, interfaces, entre otros)[51]. Posteriormente, el sistema implementa un mecanismo RAG basado en trigramas sobre el grafo del proyecto, cuyos fragmentos son suministrados al agente Cody, que procesa esta información estructurada y elabora respuestas contextualizadas a las consultas del desarrollador[52].

- **Agentes autónomos:** con un enfoque más ambicioso, soluciones como Aider²⁰ o DevinAI²¹ persiguen la automatización del ciclo completo de desarrollo de software. Debido a la complejidad inherente a dicha tarea, estas herramientas permanecen en una fase incipiente[53].

El funcionamiento de Aider presenta similitudes con SourceGraph[54]. En primera instancia, analiza el código fuente mediante la biblioteca Tree-sitter²², generando un grafo de dependencias que representa las definiciones (funciones, clases, interfaces, etc.) y las referencias

¹⁶GitHub Copilot: <https://github.com/features/copilot>

¹⁷Cursor: <https://www.cursor.com/>

¹⁸Cody: <https://sourcegraph.com/cody>

¹⁹SourceGraph: <https://sourcegraph.com/>

²⁰Aider: <https://aider.chat/>

²¹DevinAI: <https://devin.ai/>

²²Tree-sitter: <https://tree-sitter.github.io/tree-sitter/>

entre estas. Posteriormente, implementa un algoritmo de clasificación de grafos para determinar la relevancia de cada nodo respecto al contexto actual. De este modo, incorpora dicho contexto junto con un mapa de los ficheros del proyecto en la entrada del agente LLM.

2.6. Ajuste de modelos para agentes LLM

2.6.1. Limitaciones de los modelos actuales

Los modelos *instruct* del estado del arte poseen la capacidad de resolver tareas que requieren comportamiento agéntico gracias a su amplio conocimiento general. No obstante, su eficiencia es cuestionable, puesto que para tareas específicas pueden carecer de ciertos conocimientos particulares, mientras que disponen de una cantidad considerable de información superflua. Se estima que los modelos más avanzados en la actualidad, como GPT-4o o Claude 3.7 Sonnet, contienen cientos de miles de millones de parámetros^[55], lo que los convierte en prohibitivamente costosos para su ejecución en entornos locales.

Debido a estas limitaciones y por tratarse de modelos propietarios, estos están disponibles únicamente a través de API. Este modelo de acceso requiere enviar los datos a servidores externos, planteando así preocupaciones sobre la privacidad y la seguridad de la información. Esta restricción es especialmente relevante en el ámbito del desarrollo de software, donde los datos pueden incluir información sensible o confidencial.

2.6.2. Enfoques de ajuste fino para agentes específicos

Como consecuencia de estas limitación, diversos proyectos de investigación han propuesto el ajuste fino de modelos de menor dimensión para tareas agénticas específicas. Mediante esta aproximación, es posible obtener modelos con mayor precisión en las tareas requeridas, pero con una fracción del coste.

Enfoques como FireAct^[56] y AgentTuning^[57] proponen el entrenamiento de dichos modelos mediante una estrategia de destilación. Esto consiste en previamente utilizar un modelo *instruct* de gran tamaño para generar un conjunto de datos sintéticos para las respuestas del agente. Posteriormente, se ajusta un modelo de menor tamaño utilizando este conjunto de datos, de forma que el conocimiento del modelo grande se destila en el modelo pequeño.

Adicionalmente, Agent-FLAN^[58] propone la división de dichos datos de entrenamiento en varios conjuntos de aprendizaje. Debido a que los modelos aprenden antes a interpretar el formato de entrada que a razonar sobre la acción a realizar, se propone modificar parte de las trazas de entrenamiento para que estas tengan una estructura conversacional. De esta forma, al ajustar el modelo con las trazas originales, este aprende la estructura de entrada y llamada de herramientas, mientras que en las trazas conversacionales, el modelo aprende a razonar sobre la acción a realizar.

2.6.3. Técnicas de entrenamiento eficiente

El entrenamiento de estos modelos se realiza mediante el ajuste de los pesos de la red neuronal. Debido a que el ajuste fino del modelo completo es costoso, en contextos de bajo coste se propone el uso de la técnica de entrenamiento Low Rank Adaptation (LoRA)[59]. Esta técnica consiste en ajustar únicamente un subconjunto de los pesos de la red neuronal, lo que permite reducir significativamente el coste computacional del ajuste fino.

CAPÍTULO 3

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, con el propósito de 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 las arquitecturas propuestas en un proyecto software corporativo, con el propósito de 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. Requisitos

Los requisitos del proyecto se detallan en profundidad en la Sección 4

3.1.3. 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. 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. La quinta iteración se

dedicará a la implementación de arquitecturas de agentes exploratorias, evaluando su rendimiento mediante las métricas previamente establecidas.

Finalmente, la sexta iteración contemplará el ajuste fino de un modelo LLM para su integración en un agente específico del sistema, así como su evaluación contextualizada dentro del entorno funcional del proyecto.

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.

3.1.4. Descomposición de tareas

La Estructura de Descomposición de Trabajo (EDT) del proyecto se ha creado considerando el ciclo de vida iterativo-incremental del proyecto. La Figura 3.1 ilustra un diagrama de esta.

Se divide en los siguientes paquetes:

■ 1^a iteración:

- **Captura de requisitos (CR):** actividades orientadas a la definición de los requisitos necesarios para el desarrollo del proyecto.
 - **Definición de requisitos principales:** identificación y documentación de los requisitos del proyecto, validados con los directores al inicio del mismo.
 - **Elicitación de requisitos:** recopilación de preguntas potenciales para el sistema desarrollado mediante cuestionario electrónico y análisis posterior de las respuestas.
- **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:

- **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:**

- **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:**

- **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:**

- **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.
- **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.

■ **6^a iteración:**

- **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 mediante un modelo de alto rendimiento.
 - **Entrenamiento del modelo:** diseño y ejecución del ciclo de entrenamiento del modelo LLM seleccionado.
- **Sistema de agentes (SA):** actividades centradas en el diseño e implementación de un sistema de agentes LLM para asistencia en proyectos software.
 - **Incorporación del modelo ajustado:** desarrollo de adaptadores e integración del modelo en el sistema existente.
- **Pruebas y evaluación (P):** actividades destinadas a verificar y analizar el rendimiento de los módulos implementados.
 - **Evaluación comparativa:** análisis del rendimiento del sistema con el modelo ajustado frente a versiones anteriores.

■ **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 una primera iteración centrada en la captura de requisitos, invertiendo un tiempo significativo en alinear los objetivos con las necesidades empresariales. Esta fase comprende la captura de requisitos (CR), la recopilación de recursos disponibles (RR) y la definición del alcance y requisitos del sistema de agentes (DA).

Las iteraciones intermedias desarrollan progresivamente el paquete Sistema de agentes (SA) mediante fases complementarias. La segunda iteración implementa su diseño conceptual, agentes especializados y un sistema de comunicación mínima. Posteriormente, en la tercera iteración, el sistema evoluciona incorporando mecanismos de evaluación y captura de datos. La cuarta iteración optimiza los agentes existentes y explora variaciones en la orquestación, mientras que la quinta implementa arquitecturas de interacción alternativas, módulos de memoria y agentes de procesamiento avanzado.

La sexta iteración introduce el paquete Ajuste de modelo (AM), que comprende la selección del agente candidato, extracción de datos y entrenamiento especializado. Simultáneamente, el paquete SA aborda actividades de integración del modelo ajustado en la arquitectura existente.

Transversalmente, se establecen los paquetes de Gestión (G), que abarca actividades de planificación y seguimiento mediante reuniones bisemanales, y Trabajo académico (TA), orientado a la elaboración de la memoria y preparación de la defensa.

Cabe destacar que a partir de la segunda iteración, cada fase incorpora el paquete de Pruebas y evaluación (P), destinado a verificar y analizar el rendimiento de los módulos implementados. Este paquete evoluciona progresivamente desde pruebas automatizadas básicas hasta evaluaciones comparativas que contrastan el rendimiento entre iteraciones sucesivas.

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.

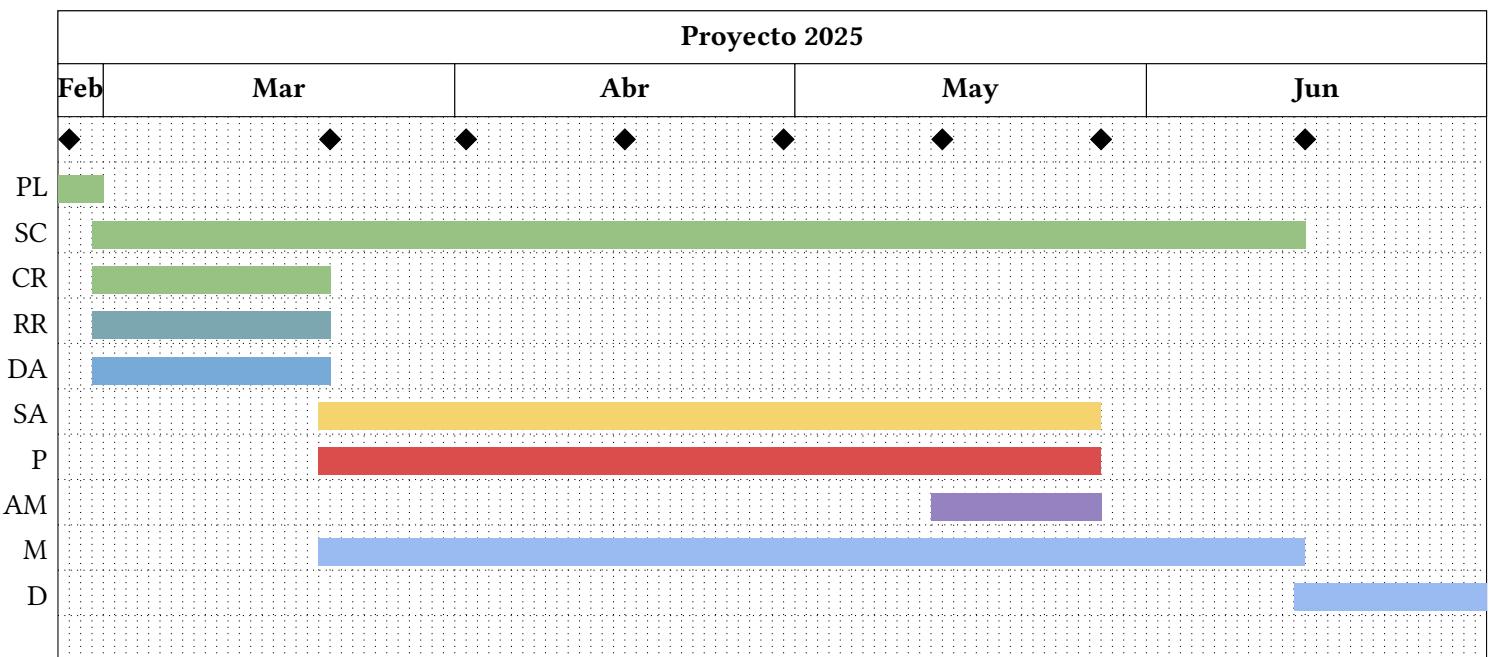


Figura 3.3: Diagrama de Gantt del proyecto

3.2.3. Hitos

En la Tabla 3.1 se detallan los hitos establecidos para el desarrollo 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	13/05/2025
Fin Iteración 6	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

3.4. Gestión de riesgos

Debido al enfoque exploratorio del proyecto, la gestión de riesgos cobra especial importancia. Se han identificado los siguientes riesgos con su correspondiente plan de contingencia:

- **R1- Concurrencia exploratoria:** dado que el proyecto está enfocado en tecnologías emergentes, existe la posibilidad de que durante el período de desarrollo emerjan iniciativas paralelas que aborden la misma problemática.

Para abordar este riesgo, se realizará una previa investigación de soluciones existentes antes de implementar cada módulo del sistema, así como un análisis periódico que permita identificar mejoras inspiradas en avances externos.

- **R2- Variabilidad del alcance:** al ser un proyecto exploratorio, con un alcance inicialmente ambiguo, podría sufrir alteraciones no planificadas. Estos contratiempos podrían suponer un sobrecoste horario, lo que obligaría a replanificar el alcance para no sobrepasar ampliamente los recursos disponibles.

Para mitigar este riesgo, se implementará un seguimiento y control mediante reuniones quincenales que permitirán evaluar la correcta evolución de las actividades y, en caso

necesario, adoptar medidas correctivas 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, tanto por los modelos de lenguaje accedidos a través de APIs como por los servicios proporcionados por los servidores MCP. Cualquier alteración o interrupción en estos servicios podría comprometer la funcionalidad del sistema implementado.

Para prevenir este riesgo, se diseñará un sistema con un manejo de excepciones robusto que garantice la continuidad operativa incluso cuando alguno de los módulos experimente fallos. Complementariamente, se adoptará una arquitectura de desarrollo modular que facilite la integración o desacoplamiento de componentes según las necesidades evolutivas del proyecto.

- **R4- Filtrado de credenciales:** el acceso a recursos externos sobre el que se desarrolla el proyecto requiere de claves secretas. Es necesario considerar que existen algoritmos de rastreo automatizados que analizan plataformas públicas en busca de credenciales expuestas para su uso fraudulento. La publicación inadvertida de estas claves en entornos públicos podría ocasionar pérdidas económicas considerables o comprometer la seguridad del sistema corporativo.

Para contrarrestar este riesgo, se implementará una política de gestión de credenciales mediante variables de entorno, evitando su inclusión directa en el código fuente o su transferencia a plataformas en la nube. Adicionalmente, se mantendrán todos los repositorios y sistemas posibles en modo de visibilidad privada.

- **R5- Pérdida de recursos:** el desarrollo del proyecto se fundamenta en múltiples recursos esenciales, incluyendo el código fuente del sistema, la documentación de requisitos, los diversos artefactos generados durante el proceso de desarrollo y la propia memoria académica. La pérdida de cualquiera de estos elementos debido a fallos técnicos o incidentes fortuitos podría ocasionar un retraso significativo.

Para mitigar este riesgo, se han implementado los sistemas de información descritos en la Sección 3.5.1 que garantizan el mantenimiento de copias de seguridad actualizadas diariamente en la nube. Mediante esta estrategia preventiva, cualquier eventualidad que afecte al dispositivo principal de desarrollo supondría, como máximo, la pérdida del trabajo correspondiente a una jornada.

3.5. Gestión de Comunicaciones e Información

3.5.1. 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.

3.5.2. Sistema de comunicación

La comunicación eficaz entre alumno, director empresarial 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 Meet:** plataforma destinada a la realización de reuniones telemáticas entre los participantes.
- **Google Chat:** herramienta complementaria para resolución de consultas rápidas con el director empresarial.

3.6. Herramientas disponibles

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

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

- **IDE PyCharm²:** 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 al modelo Claude, 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.

²PyCharm: <https://www.jetbrains.com/es-es/pycharm/>

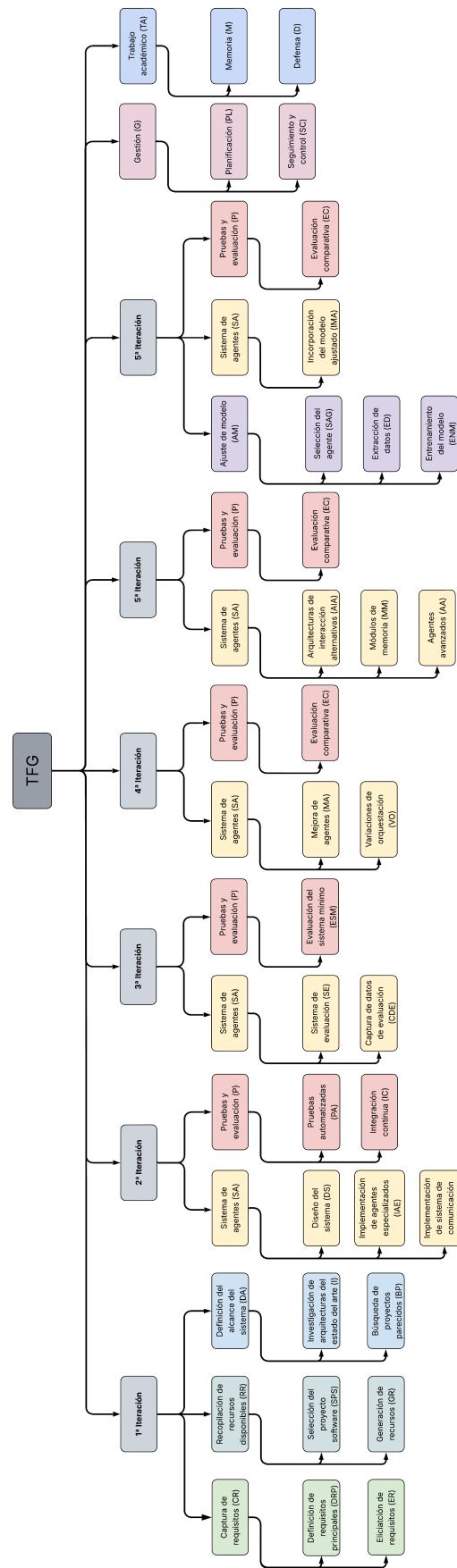
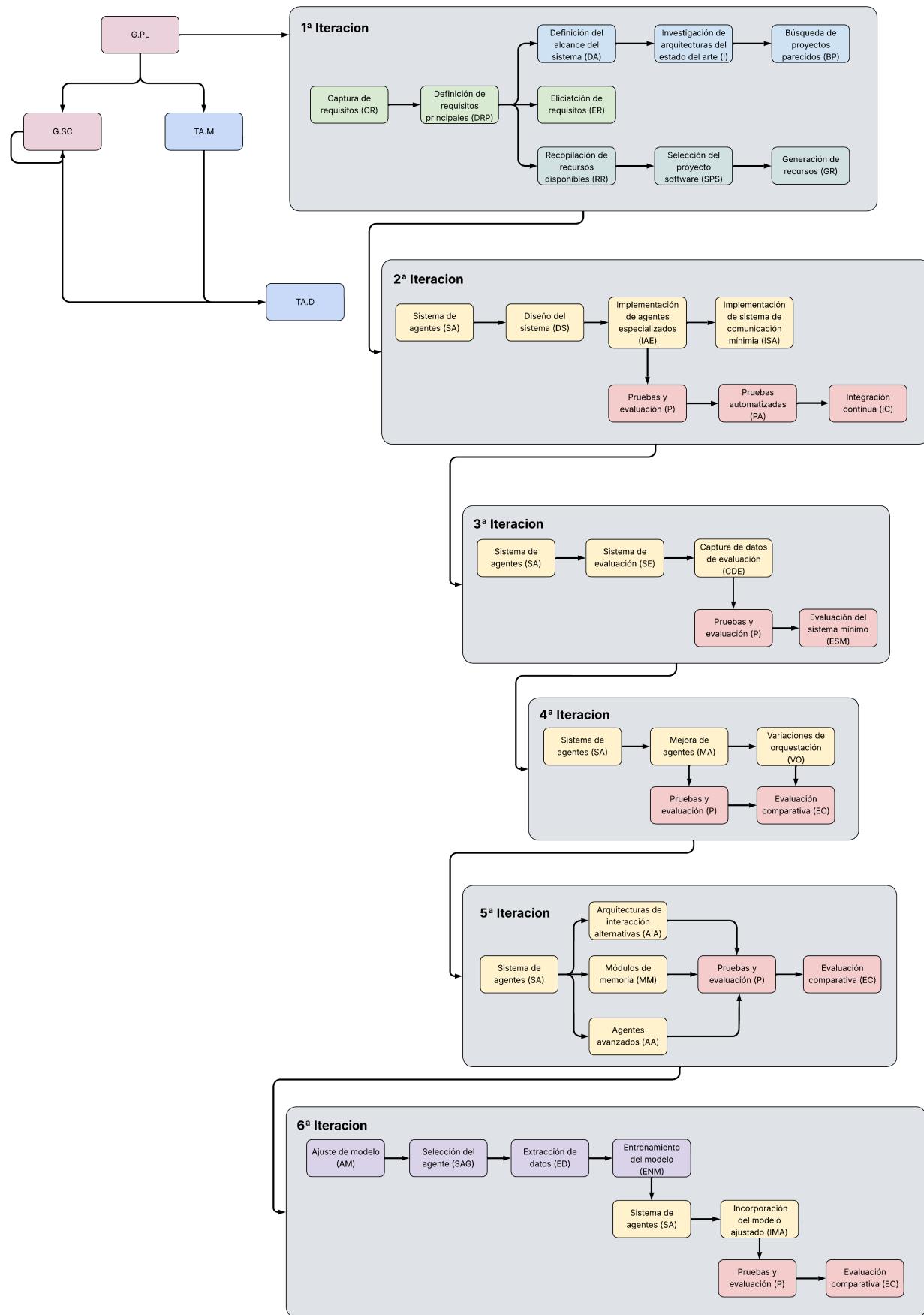


Figura 3.1: Estructura de Descomposición de Trabajo (EDT) del proyecto

**Figura 3.2:** Dependencias entre tareas del proyecto

Iteración	Paquete de trabajo	Tarea	Dedicación estimada (H)	Horas totales
1 ^a Iteración	Captura de requisitos(CR)	Definición de requisitos principales	7	361
		Elicitación de requisitos	3	
	Recopilación de recursos disponibles (RR)	Selección del proyecto software	5	
		Generación de recursos	5	
	Definición del alcance del sistema de agentes (DA)	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	
		Implementación de agentes especializados	35	
		Implementación de sistema de comunicación mínima	10	
	Pruebas y evaluación (P)	Pruebas automatizadas	5	
		Integración continua	3	
		SUBTOTAL	63	
3 ^a Iteración	Sistema de agentes (SA)	Sistema de evaluación	20	
		Captura de datos de evaluación	5	
	Pruebas y evaluación (P)	Evaluación del sistema mínimo	5	
		SUBTOTAL	30	
4 ^a Iteración	Sistema de agentes (SA)	Mejora de agentes	15	
		Variaciones de orquestación	15	
	Pruebas y evaluación (P)	Evaluación comparativa	5	
		SUBTOTAL	35	
5 ^a Iteración	Sistema de agentes (SA)	Arquitecturas de interacción alternativas	15	
		Módulos de memoria	15	
		Agentes avanzados	15	
	Pruebas y evaluación (P)	Evaluación comparativa	10	
		SUBTOTAL	55	
6 ^a Iteración	Ajuste de modelo (AM)	Selección del agente	3	
		Extracción de datos	10	
		Entrenamiento del modelo	20	
	Sistema de agentes (SA)	Incorporación del modelo ajustado	5	
		Evaluación comparativa	5	
	SUBTOTAL			43
	Gestión (G)	Planificación (PL)	15	
		Seguimiento y control (SC)	15	
		SUBTOTAL	30	
	Trabajo académico	Memoria (M)	60	
		Defensa (D)	10	
	SUBTOTAL			70

Figura 3.4: Estimación horaria de cada tarea

CAPÍTULO 4

Captura de requisitos

Una vez establecido el objetivo general del proyecto en el Capítulo 1 y asentados los conceptos con los que se trabajará en el Capítulo 2, 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 empresarial:

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 objetiva 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 especializadas: 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).

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.
- **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 dirigidas al sistema, 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. 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 Listado B.1, donde se han identificado los siguientes dominios de conocimiento:

En este párrafo tengo dudas con los tiempos verbales. He puesto "se recabaron", "fueron ampliadas", "se procedió", y luego el pretérito perfecto compuesto "se han identificado". Es como mejor me suena, pero no sé si el cambio contrasta demasiado.

- **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.
- **Recursos adicionales:** recursos externos al proyecto como formaciones y documentación de librerías.

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

- **Arquitectura del sistema:** cuestiones de diseño del código fuente sobre todos los niveles de abstracción del modelo C4[60]. 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

4.2.2. Metodología empresarial

La metodología implementada ha sido consultada directamente con los desarrolladores de LKS Next. Al tratarse de una empresa consultora, la metodología aplicada presenta 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 el GitLab³ propio de LKS Next o GitHub.

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 A.5, se ha documentado el siguiente proceso:

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.

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

³GitLab: <https://about.gitlab.com/>

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

⁵Figma: <https://www.figma.com/>

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 de cliente 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 fuentes 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 orientados directamente a clientes. En este caso, el desarrollador principal es Aritz Galdos (director empresarial 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 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.

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

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[61]. 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 ??.

Como he usado la documentación API para indexar los chunks de código, creo que lo mejor es explicarlo en esa sección.

CAPÍTULO

5

Diseño del sistema

Tras establecer los parámetros necesarios correspondientes a la gestión del proyecto, este capítulo presenta el diseño del sistema implementado. Para ello se introducirá la estructura de directorios utilizada, el diseño del software desarrollado y la gestión de las herramientas con el protocolo MCP.

5.1. Estructura del proyecto

El repositorio se estructura en cuatro componentes independientes, cada uno con su propio entorno virtual de Python (venv¹). La Figura 5.1 ilustra la organización jerárquica de los directorios principales.

El componente `sistema_agentes` alberga todos los agentes desarrollados junto con los clientes MCP. La implementación de los agentes se ubica en el directorio `src`, mientras que `static` contiene los prompts utilizados, las descripciones de los agentes y la documentación generada. El archivo `.env` almacena variables secretas como las claves de acceso a APIs, y el archivo `config.py` gestiona variables globales como el modelo predeterminado utilizado en los agentes. El archivo `main.py` integra la lógica de ejecución principal, mientras que `src/main_agent/main_graph.py` implementa el agente principal `MainAgent`.

Los directorios `servidor_mcp_bd_codigo`, `servidor_mcp_confluence` y `servidor_mcp_google_drive` contienen cada uno un servidor MCP.

¹venv: <https://docs.python.org/3/library/venv.html>

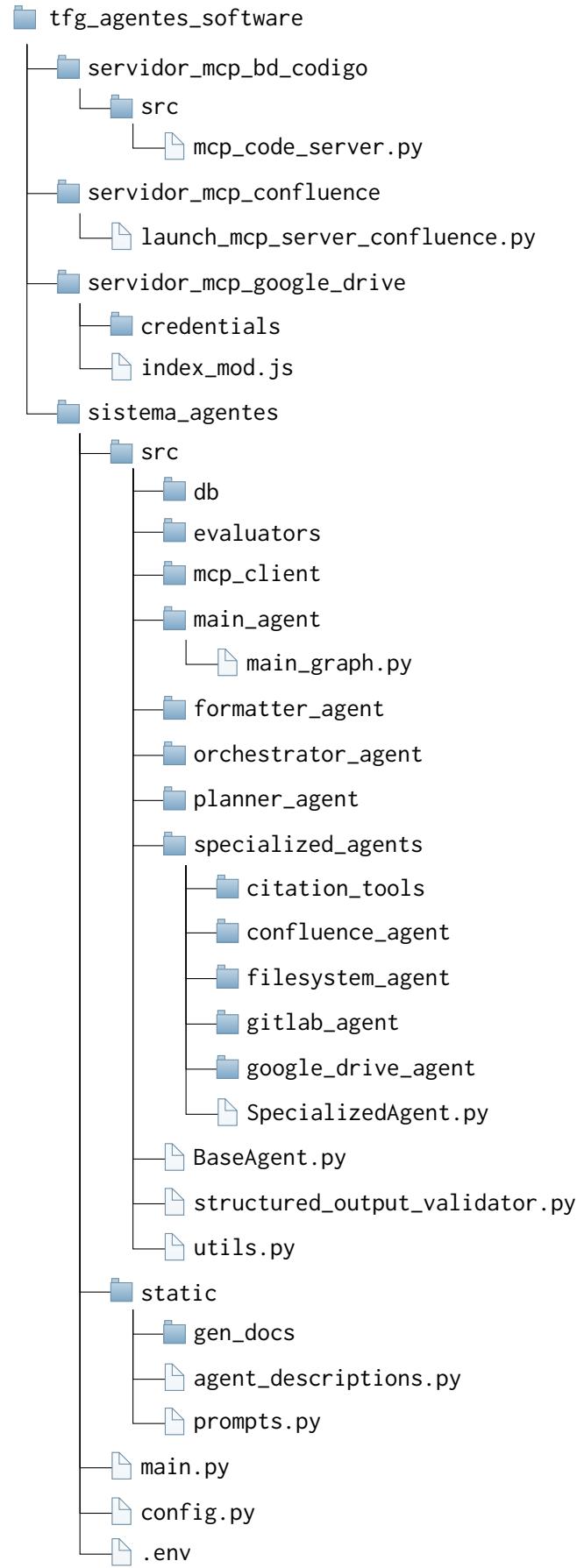


Figura 5.1: Estructura de directorios principales del proyecto

5.2. Diseño de agentes

LangGraph proporciona un marco de trabajo estructurado para la creación de flujos de ejecución en forma de grafo. Inicialmente se construye un grafo mediante el objeto StateGraph, el cual establece la lógica de enrutamiento entre diversos nodos definidos como funciones de Python. Estos grafos utilizan un denominado estado, representado por un diccionario tipado, para almacenar el estado de ejecución durante el proceso.

Bajo este paradigma, un agente puede representarse como un grafo compilado, en el cual uno o varios nodos definen la lógica de invocación a los modelos LLM, mientras que otros nodos se encargan de interpretar el resultado. En una implementación tradicional, el estado conservaría los mensajes generados entre llamadas, lo que posibilita un proceso iterativo donde el nodo de invocación al LLM se ejecuta repetidamente, añadiendo un nuevo mensaje al estado en cada ciclo de ejecución.

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 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.

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.

5.2.1. Diagrama de agentes

La Figura 5.2 ilustra el diagrama de clases (UML) general del sistema, donde se puede observar que 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

Debería referenciar en el pie de página la documentación de las clases de las librerías que menciono? Me ha dicho Claude que sobrecargaría demasiado la memoria.

Estos dos párrafos quizás quedarían mejor en una breve sección en antecedentes?

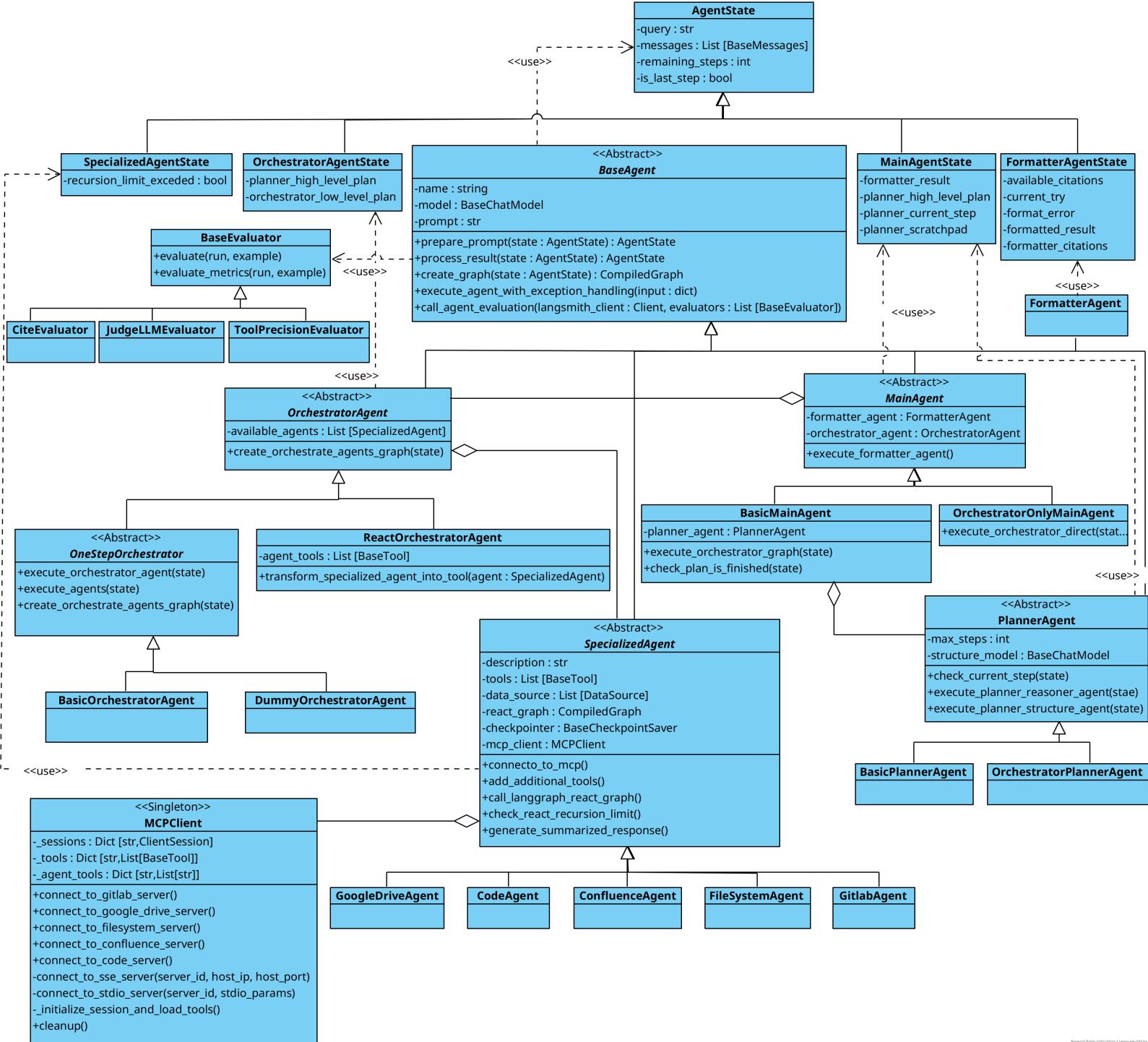


Figura 5.2: Diagrama de clases UML de del sistema de agentes.

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 de manera estática el resultado de ejecución del agente para un estado de ejecución determinado. Dependiendo del agente concreto, este resultado puede comprender todos los mensajes que satisfagan determinados criterios establecidos.
- **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 ?? 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 la Sección ?? . 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 ?? .
- **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 ?? .
- **PlannerAgent:** define la estrategia de planificación a seguir para una consulta determinada, elaborando planes secuenciales dinámicamente adaptables. Este proceso se detalla en la Sección ?? .
- **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. Consulte la Sección ?? .
- **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 ?? .

Cada agente implementa su propia clase de estado para gestionar los atributos vinculados a una ejecución específica. Estas clase 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 implementar esta estructura, se utiliza la implementación de mensajes de LangChain: `SystemMessage` para las instrucciones

que instruyen al agente, `HumanMessage` para las consultas o mensajes del usuario, y `AIMessage` para incorporar comunicaciones de agentes precedentes en el contexto de entrada.

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 Atlassian². El componente `servidor_mcp_confluence` incorpora un script de lanzamiento `launch_mcp_server_confluence.py` (detallado en el listado 5.1), un entorno virtual `.venv` y un archivo `.env` destinado al almacenamiento de credenciales de Confluence.

El script inicializa el servidor mediante el gestor de paquetes `uv`, implementando el paquete `mcp-atlassian`³ disponible en PyPI⁴. Atlassian integra el Kit de desarrollo de software (SDK) del protocolo MCP⁵ en este paquete para desarrollar un servidor de Interfaz de Pasarela de Servidor Asíncrona (ASGI) con capacidades de enrutamiento de herramientas que interactúan con su API. Tras la extracción de variables de entorno desde `.env`, la biblioteca `subprocess`⁶ ejecuta el comando `uvx` en el sistema, requiriendo que `uvx` esté previamente instalado en el entorno virtual del componente.

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

```

1      # Obtener variables de entorno
2      load_dotenv() # <- Carga las variables desde el fichero .env
```

²Repositorio de servidor MCP Atlassian: <https://github.com/sooperset/mcp-atlassian>

³Paquete `mcp-atlassian`: <https://pypi.org/project/mcp-atlassian/>

⁴PyPI: <https://pypi.org/>

⁵SDKs del protocolo MCP: <https://github.com/modelcontextprotocol>

⁶Librería `subprocess`: <https://docs.python.org/3/library/subprocess.html>

```

3   confluence_url = os.getenv('CONFLUENCE_URL')
4   ...
5
6   # Construir el comando para uvx
7   command = ["uvx", "mcp-atlassian"]
8
9   command.extend(["--transport", mcp_transport])
10  command.extend(["--port", mcp_port])
11  command.extend(["--confluence-url", confluence_url])
12  command.extend(["--confluence-username", confluence_username])
13  command.extend(["--confluence-token", confluence_token])
14
15  # Ejecutar el comando
16  try:
17      subprocess.run(command)
18  ...

```

- **Servidor Código:** proporciona acceso de lectura al repositorio software, implementando herramientas para la consulta de su código fuente. Su desarrollo se ha realizado directamente mediante el SDK de MCP para Python⁷, localizado en el fichero `src/mcp_code_server.py` del componente `servidor_mcp_bd_codigo`. La Sección ?? detalla la implementación de dichas herramientas.

El sistema emplea la clase `FastMCP`, que integra internamente un servidor ASGI mediante la librería `Starlette`⁸ 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 desde el cliente MCP.

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

```

1  # Instanciar clase
2  mcp = FastMCP("postgre")
3
4  # Vincular herramientas al servidor
5  @mcp.tool()
6  async def get_all_repository_files_list() -> TextContent:
7      """
8          Devuelve una lista en formato string serializable a JSON de todos los
9          ficheros en el repositorio respecto a su ruta relativa
10         """

```

⁷SDK de MCP para Python: <https://github.com/modelcontextprotocol/python-sdk>

⁸Librería Starlette: <https://www.starlette.io/>

```

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.

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 # Crear comando y argumentos
2 server_command = "npx"
3 server_args = ["-y", "@modelcontextprotocol/server-gitlab"]
4
5 # Obtener las credenciales desde las variables de entorno
6 server_env = {
7     "GITLAB_PERSONAL_ACCESS_TOKEN": os.getenv('GITLAB_PERSONAL_ACCESS_TOKEN'),
8     "GITLAB_API_URL": GITLAB_API_URL
9 }
10
11 # Crear instancia StdioServerParameters indicando las credenciales de GitLab como
12 # variables de entorno
13 server_params = StdioServerParameters(
14     command=server_command,
15     args=server_args,
16     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 FileSystem⁹, ejecutándolo mediante el comando npx con el paquete del registro npm¹⁰ @modelcontextprotocol/server-filesystem¹¹.
- **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-gitlab¹². Debido a que no ofrece capacidades específicas para la lectura de ciertos recursos (limitándose a operaciones de escritura), se han incorporado herramientas complementarias mediante la función add_additional_tools() en el agente SpecializedAgent, como se detalla en la Sección ??.
- **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, por lo que se ha empleado una modificación desarrollada en JavaScript¹³. En consecuencia, en lugar de especificar el paquete a utilizar en la instancia StdioServerParameters, se indica la ruta del script a ejecutar, como se muestra en el Listado 5.4.

Listing 5.4: mcp_multi_client.py: StdioServerParameters para el servidor MCP de Google Drive

```

1 # Construir el comando indicando el script a ejecutar
2 server_script_path = f"[server_path]/index_mod.js"
3 server_command = "node"
4 server_args = [server_script_path]
5
6 server_params = StdioServerParameters(
7     command=server_command,
8     args=server_args,
9     env=env
10 )

```

Adicionalmente, se ha modificado la herramienta de listar ficheros para mostrar recursivamente todos los ficheros dentro de los directorios listados como se muestra en el Listado 5.5

Este fragmento de código quizás no es necesario?

Listing 5.5: index_mod.js: herramienta gdrive_list_files utilizando la API de Google Drive

```

1 // Utilizar librería de google drive para interactuar con la api
2 import { google } from "googleapis";

```

⁹FileSystem: <https://github.com/modelcontextprotocol/servers/tree/main/src/filesystem>

¹⁰Npm: <https://www.npmjs.com/>

¹¹Server-filesystem: <https://www.npmjs.com/package/@modelcontextprotocol/server-filesystem>

¹²Server-gitlab: <https://www.npmjs.com/package/@modelcontextprotocol/server-gitlab>

¹³Servidor MCP Google Drive: <https://github.com/felores/gdrive-mcp-server/blob/main/index.ts>

```

3   const drive = google.drive("v3");
4
5   // Consultar archivos en la carpeta actual
6   const res = await drive.files.list({
7       q: '${folderId}' in parents,
8       pageSize: 1000,
9       fields: "files(id, name, mimeType, modifiedTime, size)"
10      \});
11
12  // Procesar cada archivo/carpeta
13  for (const file of res.data.files || []) {
14      allFiles.push({
15          ...
16      });
17
18      // Si es una carpeta, listar su contenido recursivamente
19      if (file.mimeType === 'application/vnd.google-apps.folder') {
20          const subFiles = await listFilesRecursively(file.id, depth + 1,
21              maxDepth);
22          allFiles = allFiles.concat(subFiles);
23      }
24  }

```

Para obtener acceso al directorio de Google Drive mediante la API, se ha configurado una aplicación en Google Cloud¹⁴ siguiendo las directrices establecidas en el servidor MCP de Google Drive oficial¹⁵. 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 adicionalmente de autenticación mediante el protocolo OAuth2.0¹⁶ 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`. Este proceso utilizará la biblioteca `google-cloud/local-auth`¹⁷, la cual iniciará el protocolo 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.

¹⁴Google Cloud: <https://cloud.google.com/?hl=es>

¹⁵Directrices de autenticación: <https://github.com/modelcontextprotocol/servers/tree/main/src/gdrive>

¹⁶OAuth2.0: <https://oauth.net/2/>

¹⁷Local-auth: <https://www.npmjs.com/package/@google-cloud/local-auth>

Se ha implementado un cliente MCP mediante el patrón Singleton¹⁸ 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 5.6.

Listing 5.6: `mcp_multi_client.py`: clase Singleton MCPClient

```

1  class MCPClient:
2      _instance = None
3
4      # Sesiones con servidores, diccionario de server_id, Session
5      _sessions: Dict[str, ClientSession] = {}
6      # Herramientas disponibles por cada servidor
7      _tools: Dict[str, List[BaseTool]] = {}
8      # Nombres de las herramientas requeridas por cada agente
9      _agent_tools: Dict[str, List[str]] = {}

```

Para acceder a las herramientas de un agente, se debe seguir un procedimiento secuencial que consiste en registrar inicialmente el agente para indicar las herramientas que este necesita, establecer la conexión con el servidor correspondiente, y posteriormente obtener las herramientas mediante el cliente, según se ilustra en el Listado 5.7.

Quizás no es necesario este fragmento de código?

Listing 5.7: `google_drive_agent_graph.py`: función connect_to_mcp en agente Google Drive

```

1  async def connect_to_mcp(self):
2      self.mcp_client = MCPClient.get_instance()
3      await self.mcp_client.connect_to_google_drive_server()
4
5      self.mcp_client.register_agent(self.name, self.tools_str)
6      self.tools = self.mcp_client.get_agent_tools(self.name)

```

Las operaciones anteriormente descritas invocarán la función de conexión con el servidor correspondiente, siguiendo dos patrones distintos según el tipo de conexión: para conexiones STDIO, como en el caso de `connect_to_google_drive_server()` (Listado 5.8), se generarán los parámetros STDIO conforme al procedimiento detallado en la Sección 5.3.2, y se ejecutará la función `connect_to_stdio_server()` (Listado 5.9); mientras que para conexiones SSE, la función correspondiente obtendrá la dirección del servidor para posteriormente establecer la conexión mediante la función `connect_to_sse_server()`, como se ilustra en el Listado 5.10.

Las conexiones implementan el SDK de MCP para Python, generando objetos `ClientSession` dentro de una pila de salida asíncrona (`AsyncExitStack`) denominada `global_exit_stack`, definida a nivel global para gestionar la liberación ordenada de recursos de todos los agentes. Esta estructura permite registrar múltiples gestores de contexto asíncronos que serán cerrados automáticamente

¹⁸Patrón Singleton: <https://refactoring.guru/es/design-patterns/singleton>

en orden inverso. Al finalizar la ejecución, la función `cleanup()` del cliente MCP cierra esta pila de salida, terminando así todas las conexiones establecidas de manera controlada, tal como se detalla en el Listado 5.11.

Listing 5.8: `mcp_multi_client.py`: función `connect_to_google_drive_server` en el cliente MCP

```

1  async def connect_to_google_drive_server(self):
2      server_id = "google_drive"
3      ...
4      await self.connect_to_stdio_server(server_id, server_params)

```

Listing 5.9: `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
7          # Crear la sesión
8          session = await global_exit_stack.enter_async_context(ClientSession(stdio,
9              write))
10         self._sessions[server_id] = session
11
12         await self._initialize_session_and_load_tools(server_id)

```

Listing 5.10: `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)

```

Listing 5.11: `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         ...
```

En ambos protocolos de conexión se invoca finalmente la función `_initialize_session_and_load_tools()`, ilustrada en el Listado 5.12. Esta función inicializa la sesión de la conexión y adapta las herramientas MCP a objetos BaseTool de LangChain mediante la función `load_mcp_tools()`. Adicionalmente, la función `patch_tool_with_exception_handling()` constituye otro adaptador que incorpora un bloque de manejo de excepciones para impedir que los errores en las herramientas se propaguen a la ejecución del agente.

Listing 5.12: 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
4         # Adaptar herramientas a BaseTool de LangChain
5         tools = await load_mcp_tools(self._sessions[server_id])
6         # Añadir control de excepciones a las herramientas
7         wrapped_tools = [patch_tool_with_exception_handling(tool) for tool in tools]
8
9         self._tools[server_id] = wrapped_tools
```


CAPÍTULO 6

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 especialistas que extienden esta base.

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 6.1 ilustra dicho flujo de ejecución.

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.

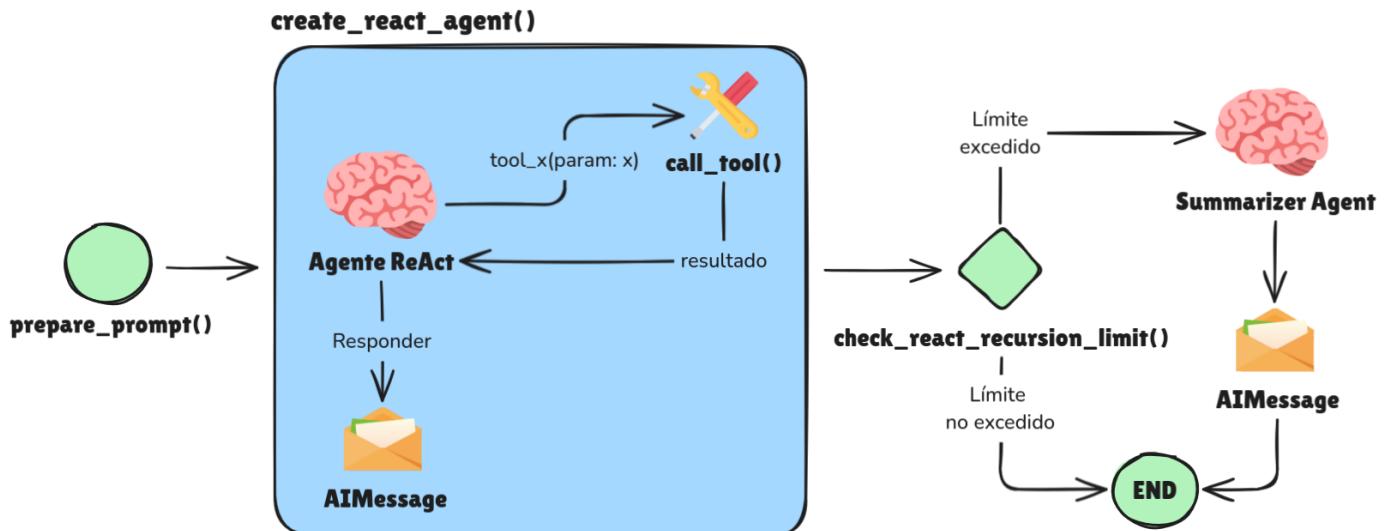


Figura 6.1: Grafo de ejecución de agentes especializados

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

```

21     graph_builder.add_conditional_edges("react", self.check_react_recursion_limit)
22
23     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. En la inicialización del agente se crea un objeto AsyncPostgresSaver, vinculado a la base de datos PostgreSQL¹ mediante un pool de conexión asíncrono A.6 y al contexto de cierre asíncrono global_exit_stack. De este modo, todos los estados de ejecución se almacenan en una colección de PostgreSQL y son accesibles mediante su identificador.

Ese último párrafo creo que queda un poco denso. ¿Debería sustituirlo por este otro que omite conceptos pero es más simple?:

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 debe indicar 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() 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 consultados. Un sistema sencillo podría solicitar en el prompt que el agente indique la ruta o nombre del archivo, 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 6.2.

El sistema proporciona a cada agente una herramienta para citar documentos dinámicamente construida según las fuentes a las que tiene acceso. Se implementa mediante la clase DataSource (Listado 6.2), que se inicializa utilizando herramientas específicas del servidor MCP para listar todos

¹PostgreSQL: <https://www.postgresql.org/>

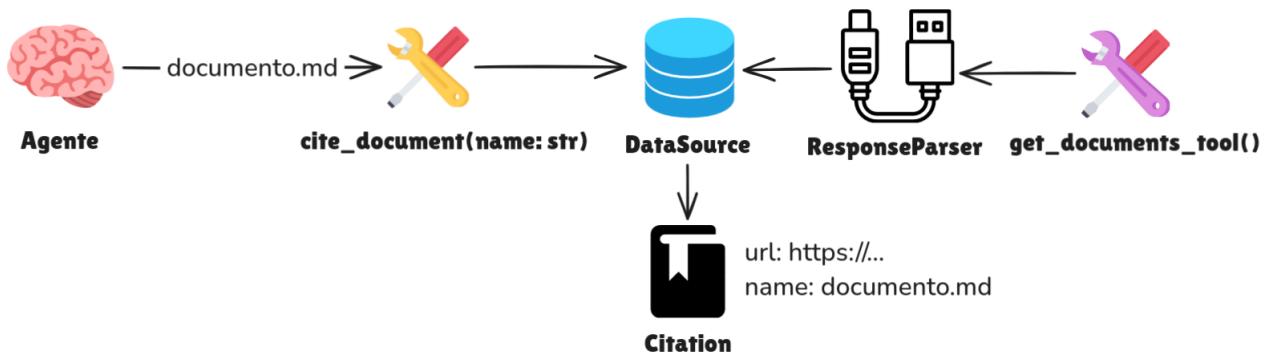


Figura 6.2: Diagrama de funcionamiento del sistema de citas

los documentos citables, como `confluence_search()` para la fuente de datos de Confluence. Durante esta inicialización, un ResponseParser personalizado adapta la salida de estas herramientas para extraer y catalogar los documentos en el DataSource.

Cada DataSource almacena la información necesaria para crear las rutas de los documentos, ensamblando automáticamente la URL completa cuando un agente necesita citar alguno de ellos. Por ejemplo, en el caso de Confluence, cada documento tiene asociado un prefijo constituido por un 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.

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
  
```

Por lo tanto, 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, como se explica en la Sección ??.

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 siguiente proceso: mediante `prepare_prompt()` se incluye en el prompt del sistema el árbol de directorios del repositorio obtenido con la herramienta `get_repository_tree_tool()` y fragmentos de código, denominados *chunks*, relevantes para la consulta actual obtenidos mediante la herramienta `get_code_repository_rag_docs_from_query_tool()`. Tras concatenar la consulta al agente mediante un `HumanMessage`, este debe decidir si buscar chunks adicionales sobre un subdirectorio concreto indicando otra consulta, buscar archivos específicos con la herramienta `get_file_from_repository_tool()` indicando su ruta relativa, o responder directamente a la consulta.

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 referenciaen 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 PGVector² 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 6.3, utilizando el ORM SQLAlchemy³ 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 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 join con la

²PGVector: <https://github.com/pgvector/pgvector>

³SQLAlchemy: <https://www.sqlalchemy.org/>

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 subdirectorio 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)
13 results = self.db_session.execute(consulta).all()

```

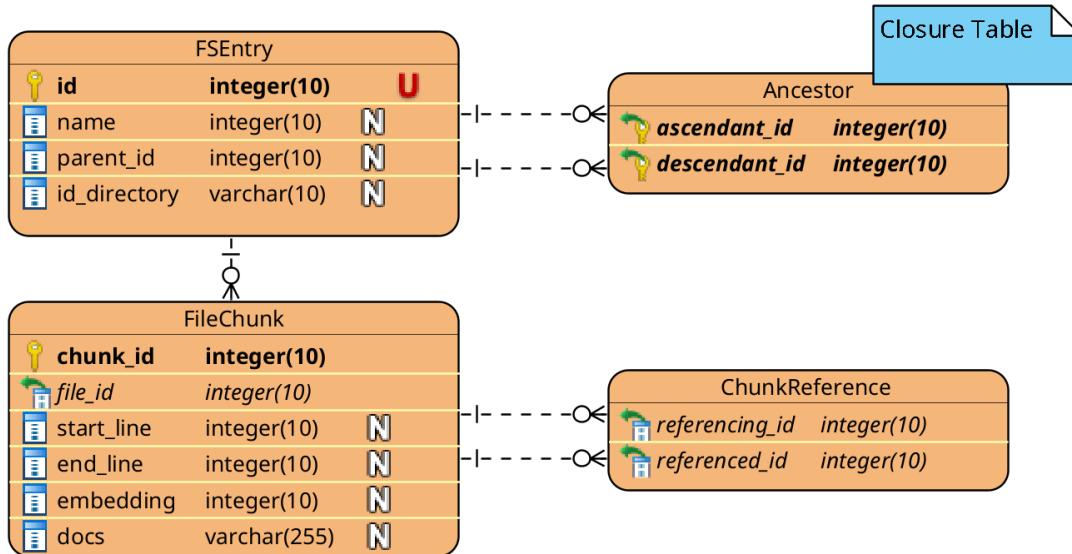


Figura 6.3: Diagrama relacional de la base de datos con el código fuente del proyecto software

Procedimiento de chunking Para dividir los ficheros en *chunks*, se han considerado las definiciones del código fuente (clases y funciones), utilizando la librería grep-ast⁴ para identificar definiciones y referencias en cada archivo.

⁴grep-ast: <https://github.com/Aider-AI/grep-ast>

La segmentación emplea un patrón State (Figura 6.4) 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.

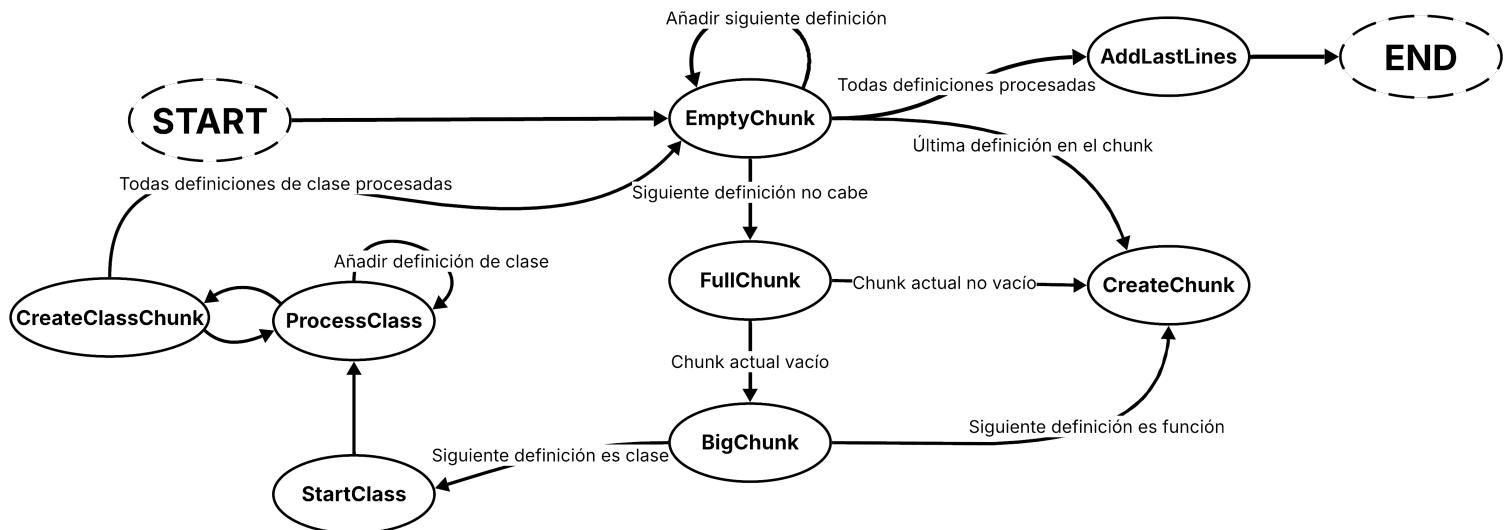


Figura 6.4: Algoritmo de chunking considerando definiciones de funciones y clases en el código fuente

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.

La fiabilidad del procedimiento se ha verificado mediante nueve tests unitarios de caja negra con pytest⁵, ejecutados automáticamente en cada push a las ramas develop o master mediante un flujo de integración continua configurado en el archivo .github/workflows/build.yml e integrado con SonarCloud⁶.

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

⁵pytest: <https://docs.pytest.org/en/stable/>

⁶SonarCloud: <https://www.sonarsource.com/products/sonarcloud/>

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 Pipeline⁷ 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 treelib⁸ y la creación de documentación API con RepoAgent.
- **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)
7
8      ...

```

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().

⁷Patrón Pipeline: <https://medium.com/@bonnotguillaume/software-architecture-the-pipeline-design-pattern-from-zero-to-hero-b5c43d8a4e60>

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

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 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.aadd_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 6.5 compara ambos enfoques, donde las secciones resaltadas en rojo 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.

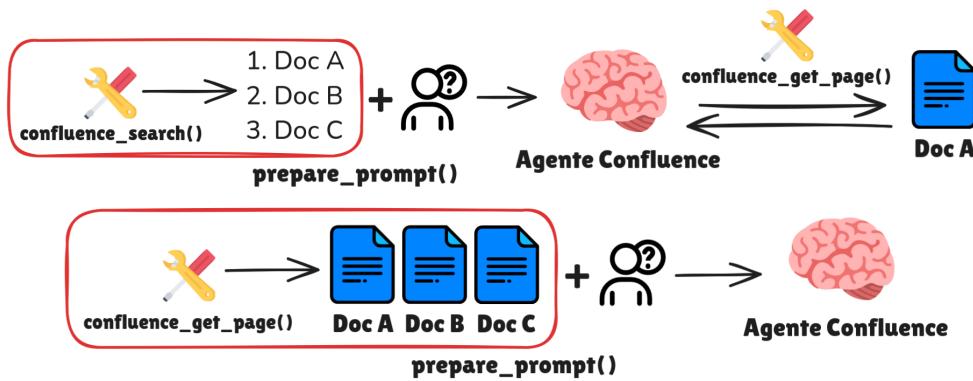


Figura 6.5: Comparación de agente Confluence estándar y enfocado en el cacheo del prompt

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`⁹. Como se detalla en el Listado 6.6, las herramientas se crean llamando a la API con las credenciales y la URL correspondiente.

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 def execute_gitlab_api_request(url: str, params: Dict[str, Any] = None) -> Response:
8     gitlab_token = os.getenv('GITLAB_PERSONAL_ACCESS_TOKEN')
9     request_url = f'{GITLAB_API_URL}/projects/{GITLAB_PROJECT_URL}/{url}'
10    headers = { "PRIVATE-TOKEN": gitlab_token }
11
12    return requests.get(request_url, headers=headers, params=params)
```

⁹requests: <https://pypi.org/project/requests/>

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.

Voy a añadir una sección al capítulo de “Diseño del sistema” donde se explique la lógica detrás de separar los agentes. He pensado en algo como el anexo A.1

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 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 (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 })

```

Esta sección la he puesto aquí pero no sé si es el lugar más adecuado.
Es donde primero menciono la salida estructurada, pero queda un poco raro en medio de “agente principal”.

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 según el formato requerido.

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

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      for _ in range(max_retries):
10         try:
11             response = await structured_model.ainvoke(prompt)
12             raw_response = response if isinstance(response, str) else None
13             # Si no sigue la estructura intentar convertirla directamente
14             if not isinstance(response, output_schema):
15                 response = output_schema.model_validate(response)
16
17         except Exception as e:
18             # Si no se convierte intentar parsear la respuesta con otro LLM
19             response = retry_parser.parse(raw_response)
20
21     ...

```

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 [62]. 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.

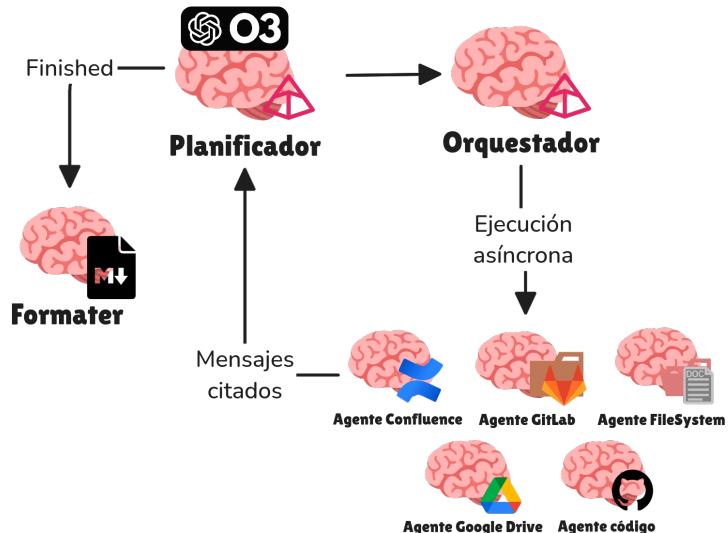
Para fundamentar su toma de decisiones, su prompt incorpora una descripción detallada de cada agente disponible, incluyendo una síntesis del tipo de información que gestiona cada uno. Este contexto se construye dinámicamente en función de los agentes integrados en el sistema, almacenando dicha caracterización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

7.3. Variaciones de orquestación

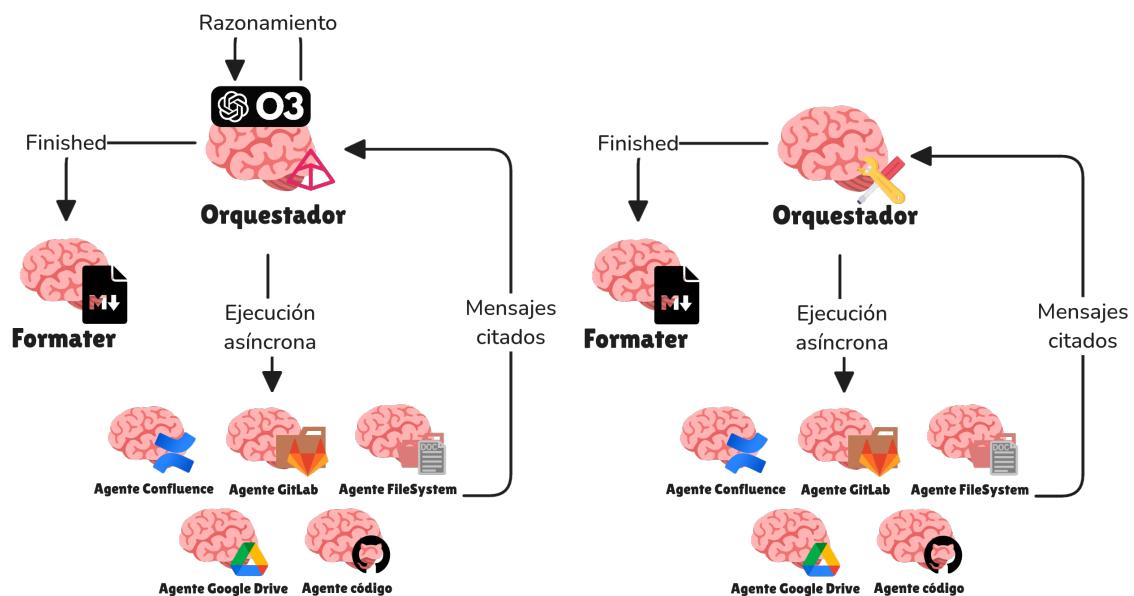
El sistema base sigue la estructura descrita en la sección 7.1, donde un orquestador selecciona los agentes a ejecutar para los pasos del plan generados por un planificador. Como se observa en la Figura 7.2a, este diseño establece una separación completa entre el agente planificador y el agente orquestador. 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.



(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

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 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 [63].

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 9.1 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.

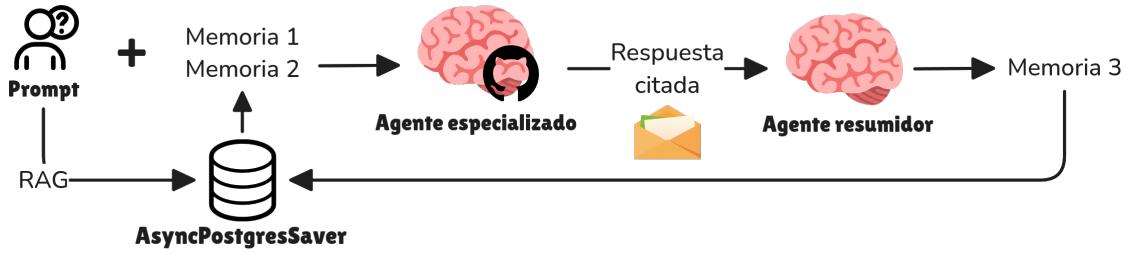


Figura 8.1: Flujo operativo del sistema de memoria de los agentes especializados

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.

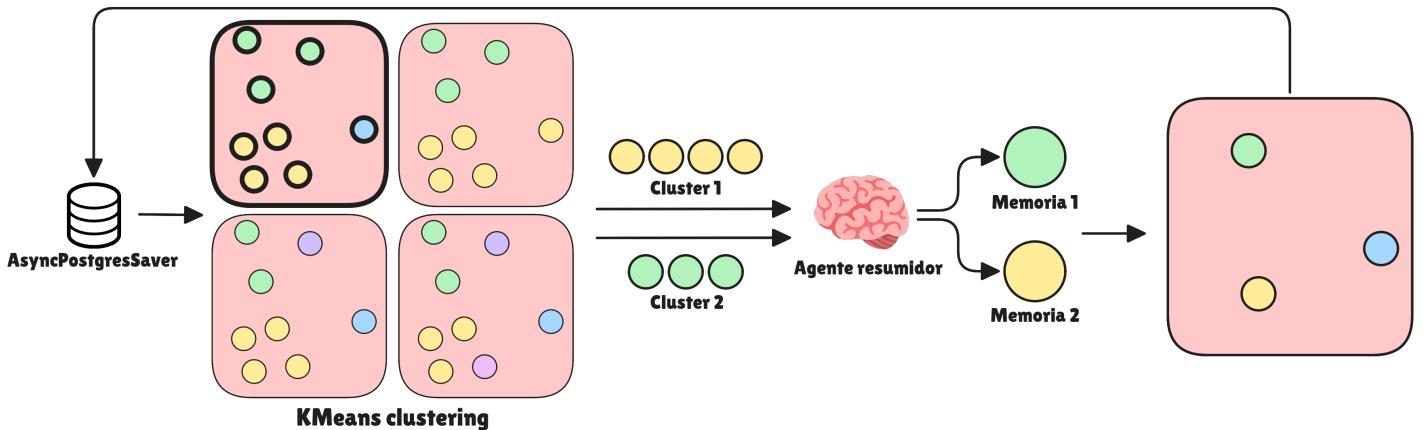
La extracción de las memorias se realiza mediante un RAG híbrido. En lugar de obtener únicamente las memorias más similares al prompt actual, se considera también las veces que estas han sido accedidas, favoreciendo las memorias más frecuentemente utilizadas. Este mecanismo actúa como un sistema de olvido, 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 los enfoques expuestos en el capítulo anterior (véase Sección ??), 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/>

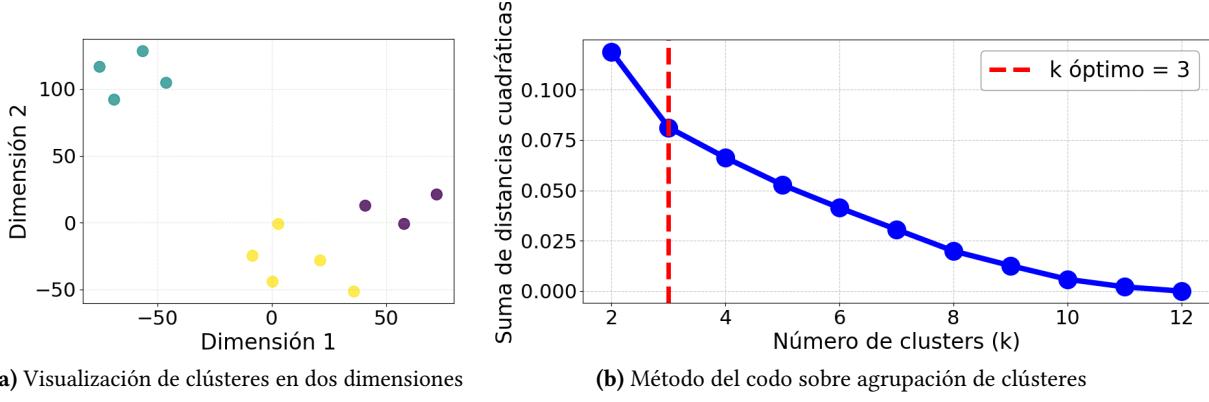


Figura 8.3: Agrupación de 3 clústeres

que un sistema híbrido surge como la solución ideal [64], 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 ???. 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) [65] 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 [66], en su versión entrenada con un corpus textual derivado de la biblioteca nacional de España [67]. 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>

Todo el tema del entrenamiento lo he sacado al anexo para no tener que explicar demasiados conceptos externos a la ingeniería del software. Supongo que podría omitirlo directamente también.

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.

CAPÍTULO

9

Evaluación

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

En esta línea, inicialmente se explica el sistema de evaluación implementado, tras lo que se exponen los resultados de evaluación y las conclusiones derivadas de dichos resultados.

9.1. Sistema de evaluación

Evaluar el sistema presenta un desafío inherente a su naturaleza, ya que se debe evaluar la capacidad de este para responder preguntas en lenguaje natural. Para resolver esta problemática, se han definido una serie de métricas utilizando para ello la metodología de evaluación ofrecida por la librería LangSmith sobre un dataset anotado manualmente que actúa como respuesta correcta o *ground truth*.

9.1.1. Métricas de evaluación

Siguiendo los principios establecidos en la Sección [\(sección antecedentes evaluación\)](#), se han definido las siguientes métricas para cada agente:

LLM juez Consiste en utilizar un agente evaluador cuya función sea valorar el resultado obtenido por el agente evaluado.

Esta métrica incorpora variabilidad de precisión adicional, ya que el agente evaluador también contiene una tasa de error. Para minimizar dicha tasa, se han anotado las respuestas esperadas como listas textuales con los conceptos necesarios a incluir en la respuesta del agente. De esta forma, el agente juez genera una respuesta estructurada que contiene un argumento booleano

para cada concepto requerido, indicando si se incluye en la respuesta generada, calculando así una precisión medible para la calidad de la respuesta.

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 perfecto, calculando 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.

Por 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 halucinación 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 contiene? , el agente no podrá responder correctamente porque no existe dicho flujo.

Para evaluar esto, un agente juez determina si la respuesta del agente realmente intenta responder a la pregunta o por el contrario indica que no se dispone de la suficiente información.

Precisión de citado Constituye la cantidad de documentos citados en la respuesta que estaban anotados como necesarios.

9.1.2. Implementación de evaluación

Se ha incorporado el sistema de evaluación de LangSmith en la arquitectura agéntica utilizada. Para ello, se ha creado un dataset en dicha plataforma para cada agente con los datos ground truth anotados. La Figura ?? muestra un esquema de dicho mecanismo

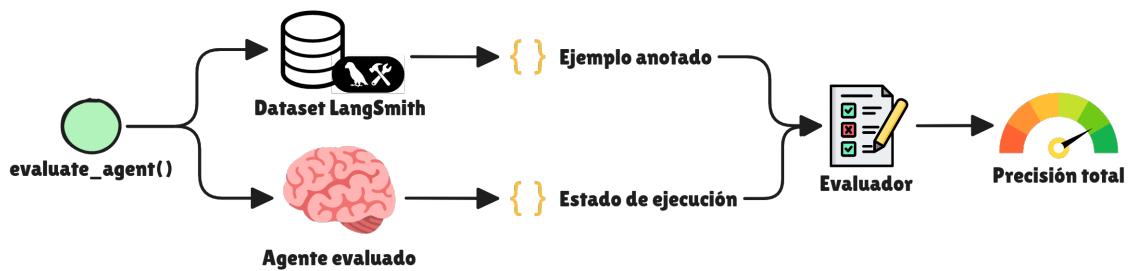


Figura 9.1: Mecanismo de evaluación de agentes

Se utiliza la función `evaluate_agent()` heredada por todos los agentes, la cual define una serie de instancias `BaseEvaluator` que se ejecutarán sobre el dataset del agente. El Listado 9.1 ilustra la evaluación de los agentes especializados, donde se declaran los evaluadores a utilizar.

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         JudgeLLMEvaluator(),
5         CiteEvaluator()
6     ]
7     result = await self.call_agent_evaluation(langsmith_client, evaluators)
8     return result
  
```

Esta función invoca `call_agent_evaluation()`, heredado por todos los agentes (Listado 9.2), que ejecuta la función de evaluación de LangSmith para aplicar las funciones de evaluación sobre cada estado final de ejecución y ejemplo anotado.

Listing 9.2: Llamada a evaluación de agentes

```

1 async def call_agent_evaluation(self, langsmith_client: Client,
2                                 evaluators: List[BaseEvaluator], ...):
3     ...
4     run_function = self.execute_from_dataset
5     results = await aevaluate(
6         run_function,
7         data=data,
8         client=langsmith_client,
9         evaluators=evaluator_functions,
10        max_concurrency=max_conc,
11        experiment_prefix=evaluation_name,
12    )
13    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         ...
6         return EvaluationResults(
7             results=[
8                 EvaluationResult(
9                     key="cite_precision",
10                    score=StrictFloat(citation_score)
11                )
12            ]
13        )

```

9.1.3. Dataset anotado

La captura de dicho dataset se ha realizado en una hoja de cálculo de Google Drive. Esta hoja se ha descargado posteriormente en formato de valores separados por coma (CSV), y se ha subido como dataset de LangSmith.

La captura constituye de 46 ejemplos para el agente principal (el sistema completo), y aproximadamente 10 ejemplos para los agentes individuales. 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:** Se han identificado tres tipos de preguntas: aquellas que se pueden responder consultando una única fuente de datos, las que requieren múltiples fuentes de datos, y las que requieren múltiples fuentes de datos en un orden secuencial, dependiendo la entrada de uno de la salida del otro.
- **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, siendo este último la información de la que se dispone y el plan actual.

De esta forma, 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.

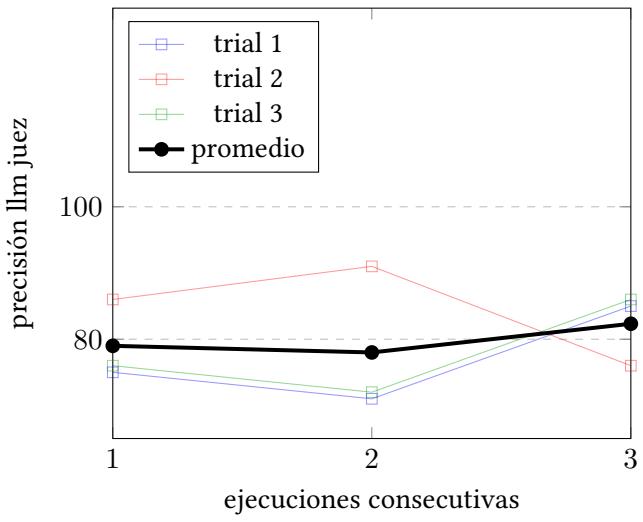
- **Orquestador:** Este agente se ha evaluado utilizando los agentes seleccionados para ejecutar como herramientas. Para ello se han incluido preguntas de todas las temáticas, evaluando qué agentes llamar en cada caso:

- **Tareas de agente único:** Se ha definido una consulta relacionada únicamente con cada agente especializado.
- **Tareas multi-agente:**
 - Información general: file_system_agent principalmente
 - Entorno y despliegue: file_system_agent + code_agent
 - Gestión del proyecto: file_system_agent + otros agentes según el caso
 - Estándares y prácticas: file_system_agent + confluence_agent si es frontend
 - Documentación: file_system_agent
 - Arquitectura del sistema: code_agent + file_system_agent en la mayoría
 - Tareas de frontend: confluence_agent + google_drive_agent

- **Agente sistema de ficheros, Confluence y Google Drive:** En los casos sencillos es necesario únicamente la herramienta de leer páginas. En los más desafiantes se requiere de las herramientas de búsqueda.
- **Agente GitLab:** Se han incluido ejemplos donde este debe llamar primero a la herramienta de obtener información sobre los usuarios en caso de necesitarlo para especificar el nombre de usuario en otras herramientas.
- **Agente código:** En los casos sencillos los chunks relevantes deberían estar incluidos en el prompt de entrada, mientras que en los más complejos debería buscar información adicional.

9.2. Resultados obtenidos

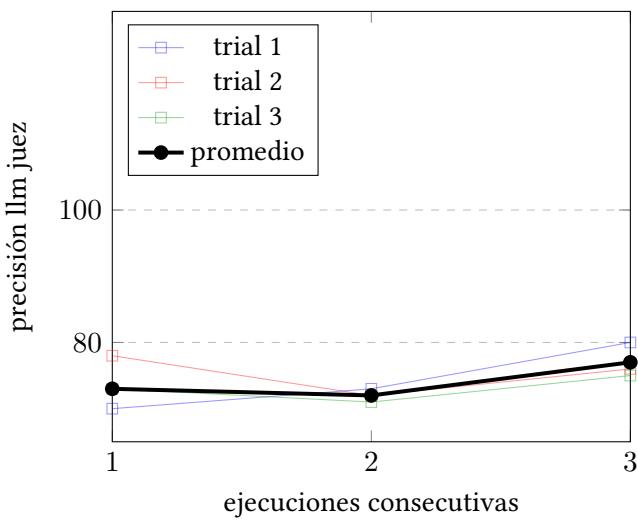
precisión en dataset test tras varias ejecuciones consecutivas



79, 78, 82,333

ejecuciones consecutivas

precisión en dataset test tras varias ejecuciones consecutivas, pasando las memorias como mensajes



media sin memorias: 76 con memorias train: 81, 83,78 media con memorias train: 80.66

Definición de términos técnicos

A.1. Diseño General

El sistema implementado explora arquitecturas aplicables a entornos con restricciones significativas 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 carecen de capacidad para procesar tal volumen textual.
- **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. 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 procesen detalles irrelevantes. El agente orquestador analiza una operación y determina qué especialistas deben intervenir según las fuentes más apropiadas.

Complementariamente, 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.

A.2. 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.3. 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.4. 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.5. 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.6. 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.

B

APÉNDICE

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 tecnologías  
147                             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
```

```
146 tarea concreta?",  
147 "52": "¿Cuál es el proceso para modificar o actualizar la documentación  
del proyecto?",  
148 "53": "¿Cuál es el procedimiento establecido para documentar cambios en el  
código?",  
149 "118a": "¿Existe un espacio de documentación centralizado como Confluence  
para el proyecto?",  
150 "120a": "¿Es obligatorio documentar los cambios realizados en el código? ¿  
Qué nivel de detalle se requiere?"  
151 },  
152 "recursos_adicionales": {  
153 "54e": "¿Existen preguntas frecuentes o discusiones en StackOverflow  
relacionadas con este tema o tecnología?",  
154 "124ae": "¿Dónde puedo encontrar la documentación técnica actualizada para  
las tecnologías o herramientas específicas que necesito utilizar?",  
155 "123a": "¿Qué herramientas o utilidades podrían ayudarme a ejecutar mis  
tareas de manera más eficiente?",  
156 "formaciones": {  
157 "55e": "¿Qué recursos formativos están disponibles sobre estas tecnologí  
as y cuáles son más relevantes para mi tarea actual?",  
158 "122ae": "¿Dónde puedo encontrar material de formación específico para  
las tecnologías utilizadas en este proyecto?"  
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
```

```
173     dependencias en este proyecto?",  
174     "127a": "¿Cuál es el procedimiento para revisar, actualizar o reemplazar  
175     dependencias vulnerables o desactualizadas?"  
176 },  
177 "modelo_c4": {  
178     "nivel_1_contexto_sistema": {  
179         "actores": {  
180             "63": "¿Quiénes son los actores o usuarios que interactúan con el  
181             sistema?",  
182             "64": "¿De qué manera específica interactúa cada tipo de actor con  
183             el sistema?",  
184             "65": "¿Cuáles son los niveles de permiso o roles definidos para  
185             cada tipo de actor en el sistema?"  
186         },  
187         "sistemas_externos": {  
188             "66": "¿Qué sistemas externos se integran o comunican con este  
189             sistema?",  
190             "67": "¿Mediante qué protocolos, estándares o interfaces se conectan  
191             los sistemas externos?"  
192     },  
193     "nivel_2_contenedores": {  
194         "68": "¿Qué aplicaciones, servicios o componentes principales  
195             conforman el sistema y cuál es la función de cada uno?",  
196         "105a": "¿La aplicación está diseñada para funcionar en múltiples  
197             plataformas o dispositivos? ¿Cuáles?",  
198         "130a": "¿Qué estrategias o patrones se aplican para optimizar el  
199             rendimiento de las consultas a bases de datos?",  
200     },  
201     "comunicacion_entre_contenedores": {  
202         "69": "¿Qué protocolos, patrones o estándares se utilizan para la  
203             comunicación entre los diferentes contenedores?"  
204     },  
205     "dependencias_entre_contenedores": {  
206         "70": "¿Cómo se gestionan las dependencias y el orden de inicio  
207             entre los diferentes contenedores?"  
208     },  
209     "tecnologias": {  
210         "71": "¿Qué tecnologías, frameworks o bibliotecas específicas  
211             utiliza cada contenedor o servicio?"  
212     },  
213 }
```

```

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     },
225     "nivel_4_diagrama_codigo": {
226         "estructura_diagrama_clases": {
227             "75e": "¿Cuál es la estructura detallada de clases, interfaces y
228             objetos dentro de un componente específico?", 
229             "76e": "¿Cuál es la responsabilidad y función principal de cada
230             clase o interfaz dentro del componente?", 
231             "132ae": "¿Puedes proporcionar un diagrama de paquetes y clases para
232             entender la estructura del código?", 
233             "133ae": "¿Puedes mostrarme la jerarquía completa de llamadas para
234             este método específico?", 
235             "134ae": "¿Cuáles son los métodos más complejos o difíciles de
236             entender en el código y por qué?" 
237         },
238         "patrones_diseño": {
239             "77": "¿Qué patrones de diseño, estructuras de herencia o composició
240             n se implementan en el código?", 
241         }
242     }
243 
```

```
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.

Bibliografía

- [1] S.E. Sim and R.C. Holt. The ramp-up problem in software projects: a case study of how software immigrants naturalize. In *Proceedings of the 20th International Conference on Software Engineering*, pages 361–370. ISSN: 0270-5257. URL: <https://ieeexplore.ieee.org/abstract/document/671389>, doi:10.1109/ICSE.1998.671389. Ver página 3.
- [2] Igor Steinmacher, Marco Aurelio Graciotto Silva, Marco Aurelio Gerosa, and David F. Redmiles. A systematic literature review on the barriers faced by newcomers to open source software projects. 59:67–85. URL: <https://www.sciencedirect.com/science/article/pii/S0950584914002390>, doi:10.1016/j.infsof.2014.11.001. Ver página 3.
- [3] Eva Ritz, Fabio Donisi, Edona Elshan, and Roman Rietsche. Artificial socialization? how artificial intelligence applications can shape a new era of employee onboarding practices. URL: <http://hdl.handle.net/10125/102648>, doi:10.24251/HICSS.2023.020. Ver página 4.
- [4] Maider Azanza, Juanan Pereira, Arantza Irastorza, and Aritz Galdos. Can LLMs facilitate onboarding software developers? an ongoing industrial case study. In *2024 36th International Conference on Software Engineering Education and Training (CSEE&T)*, pages 1–6. ISSN: 2377-570X. URL: <https://ieeexplore.ieee.org/document/10662989>, doi:10.1109/CSEET62301.2024.10662989. Ver página 4.
- [5] Andrei Cristian Ionescu, Sergey Titov, and Maliheh Izadi. A multi-agent onboarding assistant based on large language models, retrieval augmented generation, and chain-of-thought. URL: <http://arxiv.org/abs/2503.23421>, arXiv:2503.23421[cs], doi:10.48550/arXiv.2503.23421. Ver página 4.
- [6] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. URL: <http://arxiv.org/abs/1706.03762>, arXiv:1706.03762[cs], doi:10.48550/arXiv.1706.03762. Ver página 4.
- [7] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. URL: <http://arxiv.org/abs/2210.03629>, arXiv:2210.03629[cs], doi:10.48550/arXiv.2210.03629. Ver página 6.
- [8] Model context protocol (MCP). URL: <https://docs.anthropic.com/en/docs/agents-and-tools/mcp>. Ver página 7.
- [9] Fengbin Zhu, Wenqiang Lei, Chao Wang, Jianming Zheng, Soujanya Poria, and Tat-Seng Chua. Retrieving and reading: A comprehensive survey on open-domain question answering. URL: <http://arxiv.org/abs/2101.00774>, arXiv:2101.00774[cs], doi:10.48550/arXiv.2101.00774. Ver página 10.

- [10] Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, Meng Wang, and Haofen Wang. Retrieval-augmented generation for large language models: A survey. URL: <http://arxiv.org/abs/2312.10997>, arXiv:2312.10997[cs], doi:10.48550/arXiv.2312.10997. Ver página 10.
- [11] Xinbei Ma, Yeyun Gong, Pengcheng He, Hai Zhao, and Nan Duan. Query rewriting for retrieval-augmented large language models. Ver página 10.
- [12] Yoav Levine, Itay Dalmedigos, Ori Ram, Yoel Zeldes, Daniel Jannai, Dor Muhlgay, Yoni Osin, Opher Lieber, Barak Lenz, Shai Shalev-Shwartz, Amnon Shashua, Kevin Leyton-Brown, and Yoav Shoham. Standing on the shoulders of giant frozen language models. URL: <http://arxiv.org/abs/2204.10019>, arXiv:2204.10019[cs], doi:10.48550/arXiv.2204.10019. Ver página 10.
- [13] Omar Khattab, Christopher Potts, and Matei Zaharia. Relevance-guided supervision for Open-QA with ColBERT. URL: <http://arxiv.org/abs/2007.00814>, arXiv:2007.00814[cs], doi:10.48550/arXiv.2007.00814. Ver página 10.
- [14] Omar Khattab, Keshav Santhanam, Xiang Lisa Li, David Hall, Percy Liang, Christopher Potts, and Matei Zaharia. Demonstrate-search-predict: Composing retrieval and language models for knowledge-intensive NLP. URL: <http://arxiv.org/abs/2212.14024>, arXiv:2212.14024[cs], doi:10.48550/arXiv.2212.14024. Ver página 10.
- [15] Zhihong Shao, Yeyun Gong, Yelong Shen, Minlie Huang, Nan Duan, and Weizhu Chen. Enhancing retrieval-augmented large language models with iterative retrieval-generation synergy. In Houda Bouamor, Juan Pino, and Kalika Bali, editors, *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 9248–9274. Association for Computational Linguistics. URL: <https://aclanthology.org/2023.findings-emnlp.620/>, doi:10.18653/v1/2023.findings-emnlp.620. Ver página 10.
- [16] Peng Qi, Haejun Lee, Oghenetegiri "TG" Sido, and Christopher D. Manning. Answering open-domain questions of varying reasoning steps from text. URL: <http://arxiv.org/abs/2010.12527>, arXiv:2010.12527[cs], doi:10.48550/arXiv.2010.12527. Ver página 10.
- [17] Huaixiu Steven Zheng, Swaroop Mishra, Xinyun Chen, Heng-Tze Cheng, Ed H. Chi, Quoc V. Le, and Denny Zhou. Take a step back: Evoking reasoning via abstraction in large language models. URL: <http://arxiv.org/abs/2310.06117>, arXiv:2310.06117[cs], doi:10.48550/arXiv.2310.06117. Ver página 10.
- [18] Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. Interleaving retrieval with chain-of-thought reasoning for knowledge-intensive multi-step questions. URL: <http://arxiv.org/abs/2212.10509>, arXiv:2212.10509[cs], doi:10.48550/arXiv.2212.10509. Ver página 10.
- [19] Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, Wayne Xin Zhao, Zhewei Wei, and Jirong Wen. A survey on large language model based autonomous agents. 18(6):186345. doi:10.1007/s11704-024-40231-1. Ver páginas 10, 11.
- [20] Hongxin Zhang, Weihua Du, Jiaming Shan, Qinhong Zhou, Yilun Du, Joshua B. Tenenbaum, Tianmin Shu, and Chuang Gan. Building cooperative embodied agents modularly with large language models. URL: <http://arxiv.org/abs/2307.02485>, arXiv:2307.02485[cs], doi:10.48550/arXiv.2307.02485. Ver página 10.

- [21] Kevin A. Fischer. Reflective linguistic programming (RLP): A stepping stone in socially-aware AGI (SocialAGI). URL: <http://arxiv.org/abs/2305.12647>, arXiv:2305.12647[cs], doi:10.48550/arXiv.2305.12647. Ver página 10.
- [22] Xinnian Liang, Bing Wang, Hui Huang, Shuangzhi Wu, Peihao Wu, Lu Lu, Zeyun Ma, and Zhoujun Li. *Unleashing Infinite-Length Input Capacity for Large-scale Language Models with Self-Controlled Memory System*. doi:10.48550/arXiv.2304.13343. Ver páginas 10, 11.
- [23] Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. ExpeL: LLM agents are experiential learners. 38(17):19632–19642. Number: 17. URL: <https://ojs.aaai.org/index.php/AAAI/article/view/29936>, doi:10.1609/aaai.v38i17.29936. Ver página 10.
- [24] Wanjun Zhong, Lianghong Guo, Qiqi Gao, He Ye, and Yanlin Wang. MemoryBank: Enhancing large language models with long-term memory. 38(17):19724–19731. Number: 17. URL: <https://ojs.aaai.org/index.php/AAAI/article/view/29946>, doi:10.1609/aaai.v38i17.29946. Ver página 11.
- [25] Joon Sung Park, Joseph O'Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*, pages 1–22. ACM. URL: <https://dl.acm.org/doi/10.1145/3586183.3606763>, doi:10.1145/3586183.3606763. Ver página 11.
- [26] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. URL: <http://arxiv.org/abs/2201.11903>, arXiv:2201.11903[cs], doi:10.48550/arXiv.2201.11903. Ver página 11.
- [27] Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. Ver página 11.
- [28] Yancheng Wang, Ziyan Jiang, Zheng Chen, Fan Yang, Yingxue Zhou, Eunah Cho, Xing Fan, Xiaojiang Huang, Yanbin Lu, and Yingzhen Yang. RecMind: Large language model powered agent for recommendation. URL: <http://arxiv.org/abs/2308.14296>, arXiv:2308.14296[cs], doi:10.48550/arXiv.2308.14296. Ver página 11.
- [29] Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. Ver página 11.
- [30] Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. SELF-REFINE: Iterative refinement with self-feedback. Ver página 11.
- [31] Ning Miao, Yee Whye Teh, and Tom Rainforth. SelfCheck: Using LLMs to zero-shot check their own step-by-step reasoning. URL: <http://arxiv.org/abs/2308.00436>, arXiv:2308.00436[cs], doi:10.48550/arXiv.2308.00436. Ver página 11.
- [32] Bill Yuchen Lin, Yicheng Fu, Karina Yang, Faeze Brahman, Shiyu Huang, Chandra Bhagavatula, Prithviraj Ammanabrolu, Yejin Choi, and Xiang Ren. SWIFTSAGE: A generative agent with fast and slow thinking for complex interactive tasks. Ver página 11.
- [33] Wenlong Huang, Pieter Abbeel, Deepak Pathak, and Igor Mordatch. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. Ver página 11.

- [34] Zihao Wang, Shaofei Cai, Guanzhou Chen, Anji Liu, Xiaojian Ma, and Yitao Liang. Describe, explain, plan and select: Interactive planning with large language models enables open-world multi-task agents. URL: <http://arxiv.org/abs/2302.01560>, arXiv:2302.01560[cs], doi:10.48550/arXiv.2302.01560. Ver página 11.
- [35] Xizhou Zhu, Yuntao Chen, Hao Tian, Chenxin Tao, Weijie Su, Chenyu Yang, Gao Huang, Bin Li, Lewei Lu, Xiaogang Wang, Yu Qiao, Zhaoxiang Zhang, and Jifeng Dai. Ghost in the minecraft: Generally capable agents for open-world environments via large language models with text-based knowledge and memory. URL: <http://arxiv.org/abs/2305.17144>, arXiv:2305.17144[cs], doi:10.48550/arXiv.2305.17144. Ver página 11.
- [36] Chan Hee Song, Brian M. Sadler, Jiaman Wu, Wei-Lun Chao, Clayton Washington, and Yu Su. LLM-planner: Few-shot grounded planning for embodied agents with large language models. In 2023 IEEE/CVF International Conference on Computer Vision (ICCV), pages 2986–2997. IEEE. URL: <https://ieeexplore.ieee.org/document/10378628/>, doi:10.1109/ICCV51070.2023.00280. Ver página 11.
- [37] Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. URL: <http://arxiv.org/abs/2305.16291>, arXiv:2305.16291[cs], doi:10.48550/arXiv.2305.16291. Ver página 11.
- [38] Shunyu Liu, Yaoru Li, Kongcheng Zhang, Zhenyu Cui, Wenkai Fang, Yuxuan Zheng, Tongya Zheng, and Mingli Song. Odyssey: Empowering minecraft agents with open-world skills. URL: <http://arxiv.org/abs/2407.15325>, arXiv:2407.15325[cs], doi:10.48550/arXiv.2407.15325. Ver página 11.
- [39] Shreyas Sundara Raman, Vanya Cohen, Eric Rosen, Ifrah Idrees, David Paulius, and Stefanie Tellex. Planning with large language models via corrective re-prompting. Ver página 11.
- [40] LLM+p: Empowering large language models with optimal planning proficiency. URL: <http://arxiv.org/abs/2304.11477>, doi:10.48550/arXiv.2304.11477. Ver página 11.
- [41] Gautier Dagan, Frank Keller, and Alex Lascarides. Dynamic planning with a LLM. URL: <http://arxiv.org/abs/2308.06391>, arXiv:2308.06391[cs], doi:10.48550/arXiv.2308.06391. Ver página 11.
- [42] DeepSeek-r1/DeepSeek_r1.pdf at main · deepseek-ai/DeepSeek-r1. URL: https://github.com/deepseek-ai/DeepSeek-R1/blob/main/DeepSeek_R1.pdf. Ver página 11.
- [43] Ehud Karpas, Omri Abend, Yonatan Belinkov, Barak Lenz, Opher Lieber, Nir Ratner, Yoav Shoham, Hofit Bata, Yoav Levine, Kevin Leyton-Brown, Dor Muhlgay, Noam Rozen, Erez Schwartz, Gal Shachaf, Shai Shalev-Shwartz, Amnon Shashua, and Moshe Tenenholz. MRKL systems: A modular, neuro-symbolic architecture that combines large language models, external knowledge sources and discrete reasoning. URL: <http://arxiv.org/abs/2205.00445>, arXiv:2205.00445[cs], doi:10.48550/arXiv.2205.00445. Ver página 11.
- [44] Yingqiang Ge, Wenyue Hua, Kai Mei, Jianchao Ji, Juntao Tan, Shuyuan Xu, Zelong Li, and Yongfeng Zhang. OpenAGI: When LLM meets domain experts. Ver página 11.
- [45] Mingchen Zhuge, Haozhe Liu, Francesco Faccio, Dylan R. Ashley, Róbert Csordás, Anand Gopalakrishnan, Abdullah Hamdi, Hasan Abed Al Kader Hammoud, Vincent Herrmann, Kazuki Irie, Louis Kirsch, Bing Li, Guohao Li, Shuming Liu, Jinjie Mai, Piotr Piękos, Aditya Ramesh, Imanol Schlag, Weimin

- Shi, Aleksandar Stanić, Wenyi Wang, Yuhui Wang, Mengmeng Xu, Deng-Ping Fan, Bernard Ghanem, and Jürgen Schmidhuber. Mindstorms in natural language-based societies of mind. URL: <http://arxiv.org/abs/2305.17066>, arXiv:2305.17066[cs], doi:10.48550/arXiv.2305.17066. Ver página 11.
- [46] Yilun Du, Shuang Li, Antonio Torralba, Joshua B Tenenbaum, and Igor Mordatch. Improving factuality and reasoning in language models through multiagent debate. Ver página 11.
- [47] Chen Qian, Wei Liu, Hongzhang Liu, Nuo Chen, Yufan Dang, Jiahao Li, Cheng Yang, Weize Chen, Yusheng Su, Xin Cong, Juyuan Xu, Dahai Li, Zhiyuan Liu, and Maosong Sun. ChatDev: Communicative agents for software development. URL: <http://arxiv.org/abs/2307.07924>, arXiv:2307.07924[cs], doi:10.48550/arXiv.2307.07924. Ver página 11.
- [48] Sirui Hong, Mingchen Zhuge, Jiaqi Chen, Xiawu Zheng, Yuheng Cheng, Ceyao Zhang, Jinlin Wang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber. MetaGPT: Meta programming for a multi-agent collaborative framework. URL: <http://arxiv.org/abs/2308.00352>, arXiv:2308.00352[cs], doi:10.48550/arXiv.2308.00352. Ver página 11.
- [49] Eirini Kalliamvakou. Research: quantifying GitHub copilot's impact on developer productivity and happiness. URL: <https://github.blog/news-insights/research/research-quantifying-github-copilots-impact-on-developer-productivity-and-happiness/>. Ver página 12.
- [50] GitHub copilot features. URL: <https://docs-internal.github.com/en/copilot/about-github-copilot/github-copilot-features>. Ver página 12.
- [51] sourcegraph/scip. original-date: 2022-05-10T13:18:47Z. URL: <https://github.com/sourcegraph/scip>. Ver página 12.
- [52] sourcegraph/sourcegraph-public-snapshot. Code AI platform with code search & cody. URL: <https://github.com/sourcegraph/sourcegraph-public-snapshot>. Ver página 12.
- [53] Kamal Acharya. Devin: A cautionary tale of the autonomous AI engineer. URL: <https://medium.com/@lotussavy/devin-a-cautionary-tale-of-the-autonomous-ai-engineer-e1339ede8f8a>. Ver página 12.
- [54] Building a better repository map with tree sitter. URL: <https://aider.chat/2023/10/22/repomap.html>. Ver página 12.
- [55] Number of parameters in GPT-4 (latest data). URL: <https://explodingtopics.com/blog/gpt-parameters>. Ver página 13.
- [56] Baian Chen, Chang Shu, Ehsan Shareghi, Nigel Collier, Karthik Narasimhan, and Shunyu Yao. FireAct: Toward language agent fine-tuning. URL: <http://arxiv.org/abs/2310.05915>, arXiv:2310.05915[cs], doi:10.48550/arXiv.2310.05915. Ver página 13.
- [57] Aohan Zeng, Mingdao Liu, Rui Lu, Bowen Wang, Xiao Liu, Yuxiao Dong, and Jie Tang. AgentTuning: Enabling generalized agent abilities for LLMs. URL: <http://arxiv.org/abs/2310.12823>, arXiv:2310.12823[cs], doi:10.48550/arXiv.2310.12823. Ver página 13.
- [58] Zehui Chen, Kuikun Liu, Qiuchen Wang, Wenwei Zhang, Jiangning Liu, Dahua Lin, Kai Chen, and Feng Zhao. Agent-FLAN: Designing data and methods of effective agent tuning for large language models. URL: <http://arxiv.org/abs/2403.12881>, arXiv:2403.12881[cs], doi:10.48550/arXiv.2403.12881. Ver página 13.

- [59] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. URL: <http://arxiv.org/abs/2106.09685>, arXiv:2106.09685[cs], doi:10.48550/arXiv.2106.09685. Ver página 14.
- [60] C4 - abstractions. URL: <https://c4model.com/abstractions>. Ver página 32.
- [61] Qinyu Luo, Yining Ye, Shihao Liang, Zhong Zhang, Yujia Qin, Yaxi Lu, Yesai Wu, Xin Cong, Yankai Lin, Yingli Zhang, Xiaoyin Che, Zhiyuan Liu, and Maosong Sun. RepoAgent: An LLM-powered open-source framework for repository-level code documentation generation. URL: <http://arxiv.org/abs/2402.16667>, arXiv:2402.16667[cs], doi:10.48550/arXiv.2402.16667. Ver página 34.
- [62] Zhi Rui Tam, Cheng-Kuang Wu, Yi-Lin Tsai, Chieh-Yen Lin, Hung-yi Lee, and Yun-Nung Chen. Let me speak freely? a study on the impact of format restrictions on performance of large language models. URL: <http://arxiv.org/abs/2408.02442>, arXiv:2408.02442[cs], doi:10.48550/arXiv.2408.02442. Ver página 64.
- [63] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. URL: <http://arxiv.org/abs/2005.14165>, arXiv:2005.14165[cs], doi:10.48550/arXiv.2005.14165. Ver página 67.
- [64] Soyeong Jeong, Jinheon Baek, Sukmin Cho, Sung Ju Hwang, and Jong C. Park. Adaptive-RAG: Learning to adapt retrieval-augmented large language models through question complexity. URL: <http://arxiv.org/abs/2403.14403>, arXiv:2403.14403[cs], doi:10.48550/arXiv.2403.14403. Ver página 71.
- [65] Jason Wei and Kai Zou. EDA: Easy data augmentation techniques for boosting performance on text classification tasks. URL: <https://arxiv.org/abs/1901.11196v2>. Ver página 72.
- [66] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. RoBERTa: A robustly optimized BERT pretraining approach. URL: <http://arxiv.org/abs/1907.11692>, arXiv:1907.11692[cs], doi:10.48550/arXiv.1907.11692. Ver página 72.
- [67] Asier Gutiérrez-Fandiño, Jordi Armengol-Estabé, Marc Pàmies, Joan Llop-Palao, Joaquín Silveira-Ocampo, Casimiro Pio Carrino, Aitor Gonzalez-Agirre, Carme Armentano-Oller, Carlos Rodriguez-Penagos, and Marta Villegas. MarIA: Spanish language models. URL: <https://arxiv.org/abs/2107.07253v5>, doi:10.26342/2022-68-3. Ver página 72.