

Universidad Autónoma de Madrid

Escuela Politécnica Superior



Máster en Ciencia de Datos

TRABAJO FIN DE MÁSTER

Aplicación de Context Engineering en un asistente LLM para Nabsic-Enablon

Un enfoque de Ciencia de Datos aplicado a recuperación, generación y
evaluación de respuestas técnicas

Autor: Julio Carreras Bejerano

Tutor/a: Jesús García San Luis

Ponente/a: Alejandro Bellogin Kouki

Madrid, Junio 2026

Resumen

Este Trabajo Fin de Máster estudia el uso de recuperación aumentada por generación, conocida como *Retrieval-Augmented Generation* (RAG), para diseñar y evaluar un asistente de programación para Nabsic, un lenguaje de dominio específico utilizado en el entorno Enablon. La motivación del proyecto es debida a que los modelos de lenguaje (LLM) generalistas como ChatGPT pueden generar respuestas plausibles pero incorrectas cuando la tarea depende de documentación local, APIs específicas, convenciones de programación o ejemplos que no están suficientemente representados en su conocimiento paramétrico.

El trabajo se aborda desde una perspectiva de Ciencia de Datos. Incluye la construcción y normalización de un corpus local de Nabsic, la generación de embeddings, la recuperación semántica mediante un índice vectorial FAISS, la construcción de prompts, la generación de respuestas y la preparación de salidas para evaluación automática y manual. El diseño experimental compara tres pipelines RAG con complejidad contextual creciente: un pipeline base sobre un corpus de texto plano, un pipeline adaptado a un corpus Nabsic semiestructurado que mantiene la misma estrategia base de prompting que el pipeline inicial, y un pipeline de Context Engineering que incorpora sesiones, memoria, priorización de fuentes, construcción explícita del contexto y prompting específico por tipo de tarea.

La evaluación final se define sobre tres ejecuciones principales, una por cada pipeline, usando el modelo de Hugging Face empleado en las prácticas de NLP 2025/2026 como generador común. Cada ejecución utiliza el mismo dataset de 24 prompts, distribuido entre generación de código, explicación de código y corrección o detección de errores. La evaluación combina la extracción de métricas automáticas con RAGAS y una rúbrica manual, debido a que la evaluación automática de sistemas RAG está orientada principalmente a respuestas en lenguaje natural y puede no capturar por completo la corrección de código en un lenguaje especializado.

El objetivo fundamental del proyecto es analizar si un mejor diseño del corpus y una construcción más cuidadosa del contexto pueden mejorar la calidad, funda-

mentación y utilidad de las respuestas generadas, incluso cuando el modelo generador utilizado se mantiene relativamente modesto.

Palabras clave: RAG, Ingeniería de Contexto, modelos de lenguaje de gran tamaño, recuperación semántica, embeddings, FAISS, RAGAS, Nabsic, Enablon, evaluación de sistemas inteligentes, Ciencia de Datos.

Abstract

This Master’s Thesis studies the use of Retrieval-Augmented Generation (RAG) to design and evaluate a programming assistant for Nabsic, a domain-specific language used in the Enablon environment. The motivation of the project is that general-purpose language models (LLM) may generate plausible but incorrect answers when the task depends on local documentation, specific APIs, coding conventions, or examples that are not sufficiently represented in their parametric knowledge.

The work is approached from a Data Science perspective. It includes the construction and normalization of a local Nabsic corpus, the generation of embeddings, semantic retrieval through a FAISS vector index, prompt construction, response generation and the preparation of outputs for automatic and manual evaluation. The experimental design compares three RAG pipelines with increasing context complexity: a baseline pipeline over a plain-text corpus, an adapted pipeline over a semi-structured Nabsic corpus that keeps the same strategy as the initial pipeline, and a Context Engineering pipeline that incorporates sessions, memory, source prioritization, explicit context construction and task-specific prompting.

The final evaluation is defined over three main executions, one for each pipeline, using the Hugging Face model from the NLP 2025/2026 practical sessions as the common generator model. Each execution uses the same dataset of 24 prompts, distributed across code generation, code explanation, and error correction or detection tasks. The evaluation combines RAGAS automatic metrics with a manual rubric, acknowledging that automatic RAG evaluation is mainly designed for natural-language answers and may not fully capture code correctness in a specialized language.

The goal of the project is to analyze whether better corpus design and more careful context construction can improve the quality, grounding and usefulness of generated answers, even when the generator model itself remains relatively modest.

Keywords: RAG, Context Engineering, large language models, semantic retrieval, embeddings, FAISS, RAGAS, Nabsic, Enablon, intelligent system evaluation, Data Science.

Índice general

Resumen	I
Abstract	III
1. Introducción	1
1.1. Motivación del proyecto	1
1.2. Aportación del trabajo	2
1.3. Objetivos	2
1.4. Preguntas de investigación	3
1.5. Estructura del documento	4
2. Estado del arte	5
2.1. Modelos de lenguaje y asistentes de programación	5
2.2. Embeddings y recuperación semántica	6
2.3. Retrieval-Augmented Generation	6
2.4. Ingeniería de contexto	7
2.5. Sesiones y memoria como estrategia de contexto	8
2.6. Evaluación de sistemas RAG - orientación a código	9
3. Caso de estudio y datos	10
3.1. Dominio Nabsic y Enablon	10
3.2. Corpus Nabsic	11
3.3. Dataset de evaluación	12
4. Diseño e implementación del pipeline	14

4.1. Arquitectura general	14
4.2. Funcionamiento detallado del RAG base y adaptado (P1 y P2)	15
4.2.1. Carga y normalización documental	16
4.2.2. Fragmentación del corpus	16
4.2.3. Representación vectorial	16
4.2.4. Indexación con FAISS	17
4.2.5. Recuperación de contexto	17
4.2.6. Construcción del prompt	18
4.2.7. Generación de la respuesta	18
4.2.8. Persistencia de resultados	19
4.2.9. Preparación para evaluación	19
4.2.10. Trazabilidad y reproducibilidad	20
4.2.11. Valor de la baseline en el proyecto	21
4.3. Pipeline P3: Context Engineering con sesiones y memoria	21
4.3.1. Sesiones	21
4.3.2. Memoria	22
4.3.3. Recuperación de memoria	22
4.3.4. Recuperación y priorización documental	23
4.3.5. Prompt específico por tipo de tarea	23
4.3.6. Trazabilidad del contexto	24
4.4. Componentes implementados	24
5. Metodología experimental	26
5.1. Diseño experimental	26
5.2. Dataset final	27
5.3. Evaluación automática con RAGAS	27
5.4. Evaluación manual	29
6. Resultados	31
6.1. Resultados de la evaluación manual	32
6.2. Resultados por tipo de tarea	33

6.3. Resultados de la evaluación automática con RAGAS	35
6.4. Análisis de RAGAS por tipo de tarea	37
6.5. Comparación entre P1 y P2: efecto del corpus semiestructurado	38
6.6. Comparación entre P2 y P3: efecto de sesiones, memoria y prompting específico	39
6.7. Resultados por caso	40
6.8. Análisis de tiempos de ejecución	41
6.9. Discusión y conclusiones del análisis experimental	42
7. Conclusiones y trabajo futuro	44
7.1. Conclusiones	44
7.2. Limitaciones	45
7.3. Trabajo futuro	46
A. Artefactos del proyecto	51
B. Rúbrica manual completa	52
C. Ejemplos del dataset de evaluación	56
D. Plantillas de prompt utilizadas	58
D.1. Prompt base de P1 y P2	58
D.2. Prompt de Context Engineering en P3	58
E. Memorias semilla del pipeline P3	61
F. Repositorio del proyecto	63

Índice de tablas

3.1. Fuentes documentales disponibles en el proyecto.	11
4.1. Pipelines RAG definidos para la experimentación final.	15
4.2. Salidas principales del pipeline RAG.	19
4.3. Etapas del pipeline RAG desde la perspectiva de Ciencia de Datos. . .	20
4.4. Componentes técnicos principales usados en los pipelines.	25
5.1. Distribución del dataset final de evaluación.	27
5.2. Métricas RAGAS utilizadas en la evaluación automática.	28
5.3. Resumen de la rúbrica manual aplicada.	30
6.1. Resultados globales de la evaluación manual por pipeline.	32
6.2. Puntuación manual media por tipo de tarea y pipeline.	33
6.3. Resultados globales de métricas RAGAS por pipeline.	36
6.4. Media RAGAS por tipo de tarea y pipeline.	37
6.5. Tiempo medio de ejecución por pipeline.	41
A.1. Artefactos principales del repositorio local.	51
B.1. Criterios de corrección técnica y utilidad para tareas de generación. . .	52
B.2. Criterios de corrección técnica y utilidad para tareas de explicación. . .	53
B.3. Criterios de corrección técnica y utilidad para tareas de corrección o detección de errores.	54
B.4. Criterios de uso del contexto recuperado.	54
B.5. Criterios de estilo Nabsic.	55
B.6. Interpretación práctica de la puntuación manual total.	55

E.1. Memorias semilla definidas en el pipeline P3.	61
--	----

Índice de figuras

4.1. Arquitectura conceptual de los pipelines RAG evaluados para Nabsic. . .	14
6.1. Comparación global de la puntuación manual media por pipeline. . . .	33
6.2. Comparación de la evaluación manual por tipo de tarea.	35
6.3. Comparación global de la media de métricas RAGAS por pipeline. . . .	36
6.4. Comparación de la media RAGAS por tipo de tarea.	38
6.5. Comparación de la puntuación manual por caso y pipeline.	41

Capítulo 1

Introducción

1.1. Motivación del proyecto

Los modelos de lenguaje de gran tamaño han mostrado una capacidad notable para asistir en tareas de programación, documentación técnica y resolución de dudas. Sin embargo, su utilidad disminuye cuando el dominio requiere conocimiento específico, poco frecuente o no representado de forma suficiente durante el entrenamiento del modelo.

Este es el caso de Nabsic, un lenguaje utilizado en la aplicación Enablon para reglas, automatizaciones, acceso a datos y lógica de negocio. Al tratarse de un lenguaje de dominio específico, con documentación limitada y ejemplos no siempre fáciles de encontrar, un modelo generalista puede no disponer de suficiente información fiable para responder con precisión.

En este contexto, un asistente genérico puede producir respuestas verosímiles pero no necesariamente correctas, inventar funciones inexistentes o mezclar convenciones de otros lenguajes. Esta es una limitación especialmente relevante en tareas de programación, donde una respuesta aparentemente razonable puede no ser ejecutable o puede no respetar las APIs reales del entorno. Para reducir ese riesgo, el proyecto explora una arquitectura RAG, en la que el modelo recibe contexto recuperado desde un corpus local antes de generar la respuesta.

Precisamente por estas características, Nabsic resulta un caso de estudio interesante para aplicar RAG y explorar diferentes estrategias de construcción del contexto. Dicho de forma sencilla, el proyecto no busca únicamente que el sistema “conteste”, sino entender qué efecto tiene cambiar la forma en que se prepara la información que llega al modelo.

La pregunta central del proyecto, por lo tanto, no es únicamente si el sistema puede responder, sino **cómo influyen las decisiones de ingeniería de contexto en la calidad de las respuestas**.

Desde el punto de vista de Ciencia de Datos, el proyecto combina varias etapas: preparación y normalización de fuentes, representación vectorial de documentos, recuperación semántica, generación condicionada, diseño de benchmarks, evaluación automática y análisis cualitativo. La memoria se orienta por tanto a describir un flujo experimental reproducible, no solo una aplicación final.

1.2. Aportación del trabajo

La aportación principal de este TFM es el diseño y evaluación de un asistente RAG para un lenguaje técnico especializado, comparando tres configuraciones progresivas. La primera actúa como punto de partida sobre un corpus textual sencillo; la segunda incorpora un corpus Nabsic semiestructurado, manteniendo una estrategia de prompting comparable; y la tercera introduce técnicas de Context Engineering mediante sesiones, memoria, priorización de fuentes y prompts específicos por tipo de tarea.

Esta comparación permite estudiar de forma práctica hasta qué punto la mejora del sistema depende del corpus, de la recuperación documental o de la forma en que el contexto se organiza antes de llamar al modelo. Además, el trabajo combina evaluación automática mediante RAGAS con una rúbrica manual adaptada al dominio, ya que la calidad de una respuesta de programación no siempre puede valorarse solo con métricas textuales.

1.3. Objetivos

El objetivo general del TFM es diseñar, implementar y evaluar un prototipo de asistente de programación basado en LLM para el lenguaje Nabsic, explorando técnicas de ingeniería de contexto y analizando su impacto en la calidad de las respuestas.

Los objetivos específicos son:

1. Construir un corpus local con documentación Nabsic/Enablon y ejemplos de código disponibles en los proyectos realizados en dicho lenguaje.
2. Adaptar/Implementar una arquitectura RAG mínima con recuperación semánti-

ca desde un índice vectorial local.

3. Adaptar el corpus local a una estructura semiestructurada con el objetivo de aportar un contexto mejor al modelo.
4. Definir tareas representativas para el asistente: generación de código, explicación de código y detección de errores.
5. Diseñar datasets de evaluación alineados con el dominio Nabsic a partir de estas tareas representativas del asistente.
6. Implementar una variante de Context Engineering basada en sesiones, memoria semilla y memoria de sesión, manteniéndola comparable con las variantes RAG base.
7. Comparar los tres pipelines RAG progresivos: RAG base sobre corpus `.txt`, RAG adaptado a corpus Nabsic semiestructurado manteniendo la misma estrategia base de prompting, y RAG con Context Engineering y prompting específico por tipo de tarea.
8. Integrar métricas automáticas de evaluación mediante RAGAS.
9. Complementar RAGAS con evaluación manual, especialmente por las limitaciones de las métricas automáticas al valorar código.
10. Evaluar cada pipeline con un mismo modelo generador local de Hugging Face, manteniendo constante el generador para centrar la comparación en el corpus y la construcción del contexto.
11. Analizar cómo afectan al resultado las decisiones de contexto: composición del corpus, selección de fragmentos, memoria, sesiones, priorización de información y, específicamente en P3, prompting diferenciado por tipo de tarea.
12. Documentar limitaciones, riesgos y conclusiones.

1.4. Preguntas de investigación

El trabajo se estructura alrededor de las siguientes preguntas:

- PI1. ¿Cómo influye la calidad del contexto recuperado en la precisión de un asistente de programación Nabsic?

- PI2. ¿Qué mejora aporta pasar de un corpus exclusivamente textual a un corpus semiestructurado con ejemplos, documentación Markdown y PDF?
- PI3. ¿Aporta valor una construcción de contexto más estructurada, con sesiones, memoria y prioridades, frente a un pipeline RAG basado solo en documentos recuperados?
- PI4. ¿Cómo se puede evaluar y qué limitaciones aparecen al evaluar automática y manualmente las respuestas de un asistente de programación en un lenguaje de dominio específico basado en RAG?

1.5. Estructura del documento

El Capítulo 2 revisa los conceptos necesarios: modelos de lenguaje, embeddings, RAG, ingeniería de contexto y evaluación. El Capítulo 3 describe el caso de estudio, el corpus y los datos disponibles. El Capítulo 4 presenta el diseño e implementación del pipeline. El Capítulo 5 define la metodología experimental. El Capítulo 6 resume los resultados experimentales finales. El Capítulo 7 recoge las conclusiones y líneas de trabajo futuro.

Capítulo 2

Estado del arte

Este capítulo revisa los conceptos sobre los que se apoya el trabajo: modelos de lenguaje aplicados a programación, recuperación semántica, arquitecturas RAG, ingeniería de contexto, uso de sesiones y memoria, y evaluación de sistemas RAG.

2.1. Modelos de lenguaje y asistentes de programación

Los LLM han cambiado la forma en que se abordan tareas de comprensión, generación y transformación de texto. En programación, pueden generar fragmentos de código, explicar funciones, sugerir correcciones y actuar como interfaz natural sobre documentación técnica [1], [2]. Esta capacidad es especialmente visible en lenguajes muy presentes en repositorios públicos y documentación abierta, donde el modelo ha visto suficientes patrones de uso durante su entrenamiento.

Sin embargo, en dominios especializados la situación es distinta. El conocimiento paramétrico del modelo puede no contener la información exacta del sistema local, o puede mezclarla con patrones de lenguajes más comunes. En tareas de programación esto es especialmente delicado: una respuesta puede ser lingüísticamente convincente y, aun así, usar una API inexistente, una sintaxis inválida o una convención ajena al entorno real. Esta limitación conecta con el problema general de las alucinaciones en generación de lenguaje natural y con la necesidad de fundamentar las respuestas en evidencia externa cuando la precisión técnica es importante [3], [4].

En generación de código, una línea de trabajo relevante consiste en recuperar documentación antes de generar la respuesta. DocPrompting muestra que recuperar fragmentos de documentación puede mejorar la generación de código cuando la tarea

depende de APIs o librerías específicas [1]. De forma complementaria, CodeRAG-Bench estudia hasta qué punto la recuperación aumenta la generación de código y muestra que el valor del RAG no depende solo de recuperar documentos, sino de que el modelo sea capaz de integrarlos correctamente en la respuesta [2].

2.2. Embeddings y recuperación semántica

Los embeddings representan fragmentos de texto en un espacio vectorial donde la cercanía geométrica aproxima la similitud semántica [5], [6]. En un sistema RAG, cada documento o fragmento del corpus se transforma en un vector y se almacena en un índice. Ante una consulta, el sistema calcula su embedding y recupera los fragmentos más próximos.

La recuperación densa se diferencia de la búsqueda léxica clásica en que puede encontrar documentos relacionados aunque no compartan exactamente las mismas palabras con la consulta. Esta propiedad es útil en documentación técnica, donde una misma idea puede expresarse con nombres de métodos, conceptos de negocio, ejemplos o descripciones textuales. Al mismo tiempo, la recuperación semántica no garantiza por sí sola que el fragmento recuperado sea suficiente, actual o directamente aplicable a la pregunta.

La evaluación de modelos de recuperación suele apoyarse en benchmarks de búsqueda y embeddings, como BEIR y MTEB, que permiten comparar modelos en tareas heterogéneas [7], [8]. Para almacenar y consultar vectores a escala se utilizan índices especializados como FAISS, diseñado para búsqueda eficiente de vecinos cercanos en espacios vectoriales de alta dimensión [9].

2.3. Retrieval-Augmented Generation

RAG combina recuperación de información y generación de lenguaje natural [10], [11]. En lugar de pedir al modelo que responda únicamente con su conocimiento paramétrico, se le proporciona un conjunto de fragmentos recuperados desde una base documental externa. Esta idea permite complementar la memoria interna del modelo con una memoria no paramétrica, normalmente implementada mediante un índice de documentos consultable en tiempo de inferencia [10].

La idea principal es separar dos funciones que en un LLM generalista aparecen mezcladas: por un lado, localizar información relevante para una consulta concreta;

por otro, redactar una respuesta útil a partir de esa información. En una arquitectura RAG, la fase de recuperación actúa como un mecanismo de búsqueda sobre un corpus externo, mientras que la fase generativa utiliza los fragmentos recuperados como contexto adicional dentro del prompt. Esta separación es coherente con la evolución de los modelos aumentados, donde los LLM se combinan con módulos externos de búsqueda, memoria o herramientas para superar algunas limitaciones de los modelos puramente paramétricos [12].

Este enfoque resulta especialmente útil cuando la respuesta depende de información específica, privada, reciente o poco representada en los datos de entrenamiento del modelo. En esos casos, el sistema no necesita modificar los pesos del LLM mediante fine-tuning, sino construir dinámicamente un contexto que permita fundamentar la respuesta en documentos disponibles. La literatura señala que RAG puede ayudar a reducir problemas asociados a conocimiento desactualizado, falta de trazabilidad y generación de información no fundamentada, aunque no elimina completamente estos riesgos [3], [11].

No obstante, RAG no elimina por completo los errores. La calidad final depende de varios factores: que el corpus tenga la información necesaria, que el sistema recupere los fragmentos adecuados, que el contexto no introduzca ruido y que el modelo generador utilice correctamente la evidencia proporcionada. Estudios recientes muestran que los sistemas RAG pueden seguir fallando cuando el contexto recuperado es irrelevante, contradictorio, incompleto o cuando el modelo no integra correctamente la información proporcionada [13]. Por ello, un sistema RAG debe evaluarse tanto desde la perspectiva de la recuperación como desde la calidad y fidelidad de la respuesta generada [14].

2.4. Ingeniería de contexto

La ingeniería de contexto estudia cómo seleccionar, estructurar y presentar la información que recibe el modelo.

Frente a una visión centrada solo en el prompt final (*prompt engineering*), este enfoque considera el contexto como un objeto construido: incluye instrucciones, documentos recuperados, ejemplos, historial conversacional, herramientas disponibles, memoria y restricciones de formato [12]. En sistemas RAG, no basta con recuperar documentos relevantes; también importa qué fragmentos entran finalmente en el prompt, en qué orden, con qué nivel de compactación y bajo qué instrucciones.

Una dificultad práctica es que el contexto disponible para un LLM es limitado. Aumentar la cantidad de información no mejora necesariamente la respuesta; puede

introducir ruido, desplazar información relevante o provocar que el modelo use peor los fragmentos situados en posiciones intermedias del prompt. Este fenómeno se relaciona con el problema conocido como *lost in the middle*, donde los modelos tienden a aprovechar de forma desigual la información según su posición en el contexto [15].

2.5. Sesiones y memoria como estrategia de contexto

Una línea especialmente relevante de ingeniería de contexto consiste en modelar el estado conversacional y el conocimiento reutilizable como elementos explícitos del sistema. En una interacción simple, el modelo recibe una pregunta y un conjunto de documentos. En una interacción más larga o en un asistente que trabaja sobre un dominio estable, también aparecen otros elementos: preferencias del usuario, decisiones previas, convenciones del proyecto, resúmenes de conversación, hechos persistentes y trazas de acciones anteriores.

MemGPT plantea esta cuestión mediante una analogía con los sistemas operativos: dado que el contexto del modelo es limitado, el sistema debe gestionar qué información permanece en el contexto inmediato y qué información se almacena externamente para recuperarse más adelante [16]. Desde esta perspectiva, la memoria no es simplemente “más historial”, sino un mecanismo para decidir qué conocimiento debe estar disponible en cada momento. Generative Agents explora una idea relacionada en agentes simulados, donde la memoria de experiencias se almacena, se recupera y se sintetiza para generar comportamientos coherentes a lo largo del tiempo [17].

La incorporación de memoria introduce ventajas y riesgos. Por un lado, puede reducir la necesidad de repetir contexto, mantener coherencia entre interacciones y priorizar información ya validada. Por otro lado, puede arrastrar errores, introducir conocimiento obsoleto o contaminar una evaluación si el sistema aprende de ejemplos que después forman parte del benchmark. Por este motivo, los sistemas con memoria deben definir con claridad qué se almacena, con qué alcance, cuándo se recupera y cómo se evita que información no validada influya en respuestas posteriores.

En este trabajo, estas ideas se utilizan principalmente como motivación conceptual para el diseño de P3. La aplicación concreta de sesiones, memoria y priorización del contexto se describe más adelante en la sección dedicada al pipeline de Context Engineering.

2.6. Evaluación de sistemas RAG - orientación a código

La evaluación de un sistema RAG debe considerar tanto la fase de recuperación como la fase de generación. Una respuesta puede ser incorrecta por distintos motivos: porque el sistema no recupera información suficiente, porque recupera fragmentos irrelevantes o ruidosos, o porque el modelo generador no utiliza correctamente el contexto proporcionado [14], [18]. Por ello, evaluar únicamente la respuesta final puede ocultar si el error procede del retrieval, del prompt o de la generación.

Herramientas como RAGAS proponen evaluar los sistemas RAG separando dimensiones como la calidad del contexto recuperado, la fidelidad de la respuesta respecto a ese contexto y la corrección factual de la salida generada [14]. Este enfoque resulta útil para comparar distintas configuraciones de forma sistemática y para identificar qué parte del pipeline contribuye más a los errores observados.

No obstante, estas métricas automáticas no sustituyen completamente una revisión experta cuando las respuestas incluyen código. En programación pueden existir varias soluciones correctas para una misma tarea, y pequeñas diferencias de sintaxis o estilo pueden ser decisivas aunque no se reflejen bien en una comparación textual. Además, una respuesta puede estar basada en el contexto recuperado y aun así no ser ejecutable, no respetar convenciones del dominio o no resolver de forma útil la tarea planteada.

La evaluación automática de código suele requerir criterios adicionales a la similitud textual. Benchmarks como HumanEval y MBPP evalúan modelos generadores mediante problemas de programación asociados a comportamientos esperados o tests funcionales, evitando depender únicamente del parecido superficial con una solución de referencia [19], [20]. De forma complementaria, suites como CodeXGLUE amplían la evaluación a tareas de comprensión, generación, reparación y transformación de código [21].

En este trabajo, dado que Nabsic es un lenguaje de dominio específico y no se dispone de un entorno de ejecución automatizado equivalente al de benchmarks como HumanEval o MBPP, la evaluación se adapta mediante una rúbrica manual. Esta rúbrica permite valorar la corrección técnica, el uso del contexto recuperado y la adecuación al estilo Nabsic. De este modo, la evaluación combina métricas automáticas propias de sistemas RAG con una revisión humana orientada a la utilidad práctica de las respuestas en el entorno Enablon.

Capítulo 3

Caso de estudio y datos

3.1. Dominio Nabsic y Enablon

Enablon es una plataforma de software empresarial de Wolters Kluwer orientada a la gestión integrada de riesgos, sostenibilidad, medio ambiente, salud y seguridad, calidad, cumplimiento normativo y excelencia operacional. La plataforma se utiliza en organizaciones que necesitan estructurar procesos complejos relacionados con EHSQ, ESG, gestión de riesgos operacionales, auditorías, incidentes, controles y datos corporativos distribuidos [22].

Dentro de este entorno, Nabsic es un lenguaje de dominio específico utilizado para expresar lógica de aplicación, reglas, automatizaciones, consultas a la base de datos, transformaciones de datos e integraciones. Su uso está ligado al funcionamiento interno y a la configuración avanzada de soluciones Enablon, por lo que combina elementos propios de programación con conceptos específicos de la plataforma.

El interés del caso de estudio reside en que Nabsic pertenece a un dominio técnico especializado. Su sintaxis presenta similitudes con Visual Basic, lenguaje de programación desarrollado por Microsoft del ecosistema .NET [23]. Sin embargo, Nabsic incorpora convenciones, objetos y funciones propias del entorno Enablon que no forman parte habitual del conocimiento general de los modelos de lenguaje. A diferencia de lenguajes ampliamente documentados como Python o JavaScript, la información disponible sobre Nabsic es mucho más limitada y depende en gran medida de manuales, ejemplos internos, documentación de objetos y patrones de uso específicos de Enablon.

Desde el punto de vista técnico, esto supone una dificultad importante a la hora de desarrollar código en este lenguaje, ya que a menudo es necesario recurrir al conocimiento experto o a documentación interna posiblemente fragmentada o desactualizada.

Esta combinación de lenguaje de programación, plataforma empresarial y documentación específica convierte a Nabsic-Enablon en un caso adecuado para estudiar el uso de RAG y técnicas de construcción de contexto sobre dicho entorno.

3.2. Corpus Nabsic

El corpus local recogido para este trabajo está compuesto por documentación oficial exportada, documentación convertida a Markdown, ejemplos de código y catálogos de módulos Enablon. La Tabla 3.1 resume las principales fuentes documentales consideradas en el proyecto.

Tabla 3.1: Fuentes documentales disponibles en el proyecto.

Fuente	Contenido	Estado
PDFs/	Manuales oficiales Nabsic: arrays, cadenas, sintaxis, operadores, variables predefinidas, variables de contexto y RecordSet.	Incluido
DocNabsicDetallada/	Documentación Markdown sobre variables de contexto, funciones de conversión, operadores y acceso a base de datos.	Incluido
Objects/	Documentación por objetos Nabsic y sus funciones: <code>Array</code> , <code>Dictionary</code> , <code>RecordSet</code> , <code>String</code> , <code>Date</code> , <code>SOAP_Client</code> , entre otros.	Incluido
Ejemplos/	Ejemplos internos y ejemplos de programación básicos añadidos.	Incluido
ModulosEnablon/	Catálogos compactos Markdown para campos y tablas de módulos Enablon HO (Shared Functions) e IMS (Health and Safety).	Incluido
*.xlsx	Ficheros Excel originales de módulos Enablon. Excluidos	Excluido

Para la experimentación final se definen dos configuraciones de corpus:

- C1. **Corpus TXT**: versión con los documentos directamente convertidos a `.txt`, alineada con el RAG base de las prácticas de NLP 2025/2026. Su objetivo es actuar como punto de partida sencillo y comparable.

C2. Corpus semiestructurado: versión enriquecida del corpus, formada por documentación en Markdown semiestructurado, ejemplos dedicados, documentación por objetos, catálogos compactos de módulos Enablon y PDFs técnicos.

El corpus semiestructurado se preparó progresivamente para facilitar la recuperación semántica y reducir el ruido documental. Para ello, la documentación y los ejemplos se convirtieron a Markdown, se añadieron resúmenes orientados a RAG en algunos documentos y se normalizó a inglés la documentación no procedente de manuales oficiales, con el objetivo de reducir la mezcla de idiomas dentro del índice.

También se compactaron los catálogos de módulos Enablon en registros más adecuados para recuperación, como `TABLE_RECORD` y `FIELD_RECORD`. En estos casos se utilizó una línea `rag_text` autocontenida por campo, de forma que cada fragmento incluyera información suficiente para ser interpretado sin depender excesivamente del contexto anterior o posterior.

Finalmente, los ficheros `.xlsx` se conservaron como fuentes originales, pero se excluyeron del índice RAG porque su estructura tabular introducía ruido y generaba fragmentos menos útiles para la recuperación.

La comparación entre ambos corpus permite medir si el enriquecimiento documental y la semiestructuración mejoran la recuperación y la respuesta generada usando un modelo generador relativamente simple.

3.3. Dataset de evaluación

Durante el desarrollo del proyecto se utilizaron varios conjuntos de prueba con objetivos diferentes. En las primeras fases se emplearon datasets reducidos para validar el flujo de evaluación, comprobar la compatibilidad con RAGAS y verificar que el sistema podía procesar preguntas, respuestas, contextos recuperados y referencias esperadas de forma consistente. Estos conjuntos preliminares tuvieron una función exploratoria y de depuración, por lo que no se utilizan para reportar los resultados finales.

En concreto, primero se generó una versión inicial de 15 casos generales que podría requerir un usuario del asistente. Después, se construyó un dataset específico con 10 casos, que incorporaba tareas de generación de código, explicación de fragmentos y detección o corrección de errores de Nabsic. Esta evolución permitió ajustar el diseño del benchmark y la rúbrica manual antes de fijar el conjunto definitivo.

La evaluación comparativa final se realiza únicamente sobre un dataset específico de Nabsic compuesto por 24 casos. Este conjunto constituye el benchmark empleado

para comparar los tres pipelines del sistema RAG y es el único dataset considerado en los resultados finales del trabajo.

La distribución del dataset final sigue una proporción 50 %-25 %-25 %. La mitad de los casos se dedica a generación de código, al ser el caso de uso principal del sistema. El resto se reparte entre explicación y corrección de errores, lo que permite evaluar también comprensión de código y utilidad en escenarios de depuración. Esta combinación se inspira en benchmarks de programación que no se limitan únicamente a generación, sino que incluyen tareas de comprensión, reparación o análisis de código, como HumanEval, MBPP y CodeXGLUE [19], [20], [21].

Cada caso del dataset final incluye el prompt de entrada, el tipo de tarea, la dificultad, una referencia esperada cuando aplica, contextos de referencia y notas de evaluación. Esta estructura permite aplicar una rúbrica manual común a las respuestas generadas por los distintos pipelines, manteniendo la comparación en condiciones homogéneas y separando claramente las pruebas de desarrollo de la evaluación experimental final.

Capítulo 4

Diseño e implementación del pipeline

4.1. Arquitectura general

La implementación final se organiza en tres pipelines RAG comparables, especificados en la Tabla 4.1. Todos comparten la misma lógica experimental general: cargar un dataset de prompts, construir o cargar un índice documental en base al corpus usado, recuperar contexto, generar una respuesta, guardar trazas y preparar resultados para evaluación automática y manual (ver Figura 4.1). La diferencia entre ellos reside en el corpus utilizado y en la estrategia de construcción del contexto.

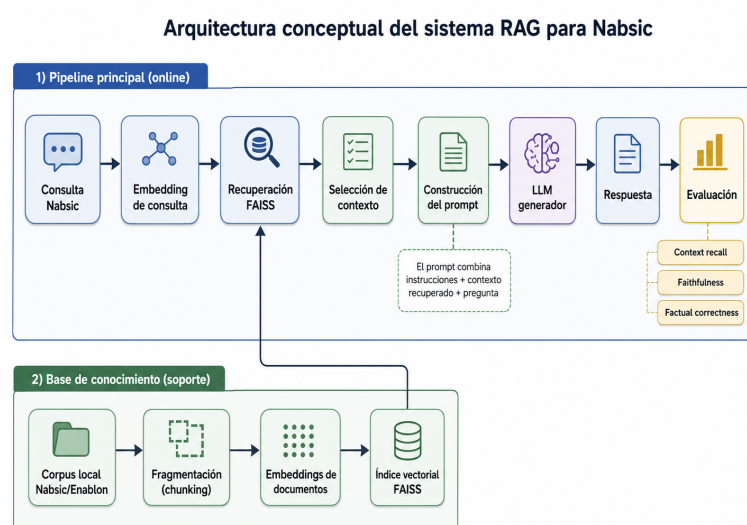


Figura 4.1: Arquitectura conceptual de los pipelines RAG evaluados para Nabsic.

Tabla 4.1: Pipelines RAG definidos para la experimentación final.

Pipeline	Notebook	Descripción
P1	RAG base NLP	Pipeline proporcionado en las prácticas de NLP 2025/2026, orientado a lectura de documentos <code>.txt</code> . Se usa como base mínima sobre el corpus convertido a <code>.txt</code> y emplea la estrategia base de prompting.
P2	RAG base multifor- mato	Adaptación del RAG base para corpus Nabsic semiestructurado, con Markdown, PDFs y ejemplos dedicados. Mantiene la misma estrategia base de prompting que P1, lo que permite aislar mejor el efecto del corpus.
P3	RAG Context Enginee- ring: Sesiones y Memo- ria	Pipeline que introduce técnicas de Context Engineering: sesiones, memoria, construcción trazable del contexto, priorización de fuentes, presupuestos aproximados de tokens y prompting específico por tipo de tarea.

4.2. Funcionamiento detallado del RAG base y adaptado (P1 y P2)

El Pipeline P1 tiene un gran valor dentro del proyecto, porque establece la base aportada en la asignatura de Procesamiento de Lenguaje Natural sobre la que ya se pueden realizar experimentos controlados. No se trata únicamente de un prototipo que responde preguntas, sino de un proceso reproducible que transforma un corpus documental en una fuente consultable, registra las decisiones de configuración y produce artefactos evaluables.

Partiendo de esta base, el pipeline P2 mantiene la misma estructura funcional general y la misma estrategia base de prompting que P1. Su diferencia principal reside en el corpus utilizado: mientras P1 trabaja sobre una versión reducida basada en documentos `.txt`, P2 incorpora un corpus Nabsic semiestructurado y multifor-
mato, compuesto por Markdown, PDFs y ejemplos dedicados.

Esta decisión tiene una función metodológica importante: permite comparar P1 y P2 aislando principalmente el efecto del corpus. Si se modificaran simultáneamente el corpus y la estrategia de prompting, sería más difícil atribuir las diferencias observadas a una causa concreta.

El pipeline P3 parte de la misma lógica general de recuperación y generación, pero introduce cambios más profundos en la construcción del contexto. En este caso sí se modifica la estrategia de prompting, incorporando instrucciones diferenciadas

por tipo de tarea junto con sesiones, memoria, priorización de fuentes y presupuestos aproximados de contexto.

4.2.1. Carga y normalización documental

La primera fase del pipeline consiste en cargar los documentos locales. Los PDF se procesan con `PyPDFLoader`; los Markdown se leen como texto plano conservando metadatos, títulos, secciones y ejemplos.

En un comienzo del proyecto, se consideró extraer los Excel con la información de los módulos Enablon básicos. Sin embargo, para mantenerlos alineados con el resto del corpus, no se indexan directamente: se sustituyen por versiones Markdown compactas, más adecuadas para recuperación semántica.

Cada documento cargado se representa internamente como un objeto con contenido textual y metadatos. Los metadatos conservan, cuando están disponibles, la ruta de origen, el tipo de fichero, la página en el caso de PDF y posiciones aproximadas dentro del documento. Esta información permite que los resultados no sean una caja negra: una respuesta puede trazarse hasta las fuentes recuperadas.

4.2.2. Fragmentación del corpus

Una vez cargados los documentos, el texto se divide en fragmentos mediante `RecursiveCharacterTextSplitter`. La configuración base usa fragmentos de 512 caracteres con solapamiento de 80 caracteres. El objetivo del solapamiento es reducir el riesgo de cortar una definición, una firma de función o un ejemplo de código justo en el límite entre dos fragmentos [11].

El chunking convierte documentos largos y heterogéneos en unidades comparables para recuperación. Esta decisión tiene implicaciones experimentales importantes. Fragmentos demasiado pequeños pueden perder contexto; fragmentos demasiado grandes pueden mezclar conceptos diferentes y consumir espacio del prompt. Por eso el tamaño de chunk se considera una variable central de ingeniería de contexto.

4.2.3. Representación vectorial

Cada fragmento se transforma en un embedding usando `thenlper/gte-small`. Formalmente, si el corpus fragmentado se representa como un conjunto de fragmentos $D = \{d_1, d_2, \dots, d_n\}$, el modelo de embeddings aplica una función:

$$e_i = f(d_i)$$

donde e_i es el vector asociado al fragmento d_i . La consulta del usuario se transforma del mismo modo:

$$q = f(x)$$

donde x es la pregunta o instrucción de entrada. La recuperación se basa en comparar el vector de la consulta con los vectores de los fragmentos del corpus.

Esta fase conecta el proyecto con algunas de las técnicas vistas durante el Máster relacionadas con Procesamiento del Lenguaje Natural: representación numérica de texto, búsqueda por similitud y construcción de índices para consulta eficiente [6], [7].

4.2.4. Indexación con FAISS

Los embeddings de los fragmentos se almacenan en un índice FAISS local. FAISS permite realizar búsqueda eficiente de vecinos cercanos en el espacio vectorial [9].

En la configuración base, ante una consulta se recuperan los 8 fragmentos más similares y se seleccionan 5 para construir la respuesta.

La separación entre “documentos recuperados” y “documentos usados” permite diagnosticar dos momentos distintos: primero, la amplitud de recuperación; segundo, la cantidad de evidencia que finalmente entra en el prompt. Esta distinción será útil para comparar configuraciones donde se recupere más información de la que se usa, o donde se aplique una fase futura de reranking.

4.2.5. Recuperación de contexto

Para cada fila del dataset, el pipeline toma `user_input`, calcula su embedding y consulta el índice FAISS. El resultado es una lista ordenada de fragmentos, cada uno acompañado de sus metadatos. El orden responde principalmente a similitud vectorial.

El contexto recuperado cumple dos funciones. Primero, proporciona evidencia técnica al modelo generador. Segundo, queda guardado como dato experimental para evaluar si el fallo, si lo hay, procede de la recuperación o de la generación. Por ejemplo, una respuesta incorrecta con contextos irrelevantes apunta a un problema de retrieval; una respuesta incorrecta con contextos correctos apunta más bien al prompt o al modelo generador.

4.2.6. Construcción del prompt

En P1 y P2 se utiliza la misma estrategia base de prompting. El prompt se compone de una instrucción de sistema, un bloque de contexto recuperado y la pregunta original del usuario. La instrucción general indica al modelo que responda usando la información contenida en el contexto, que la respuesta sea concisa y relevante, que cite el número del documento fuente cuando proceda, que no responda si la respuesta no puede deducirse del contexto y que evite inventar sintaxis o APIs de Nabsic. De forma esquemática, la plantilla común de P1 y P2 sigue la estructura indicada en el Apéndice D.1.

Por tanto, la diferencia entre P1 y P2 no está en el prompt, sino en el corpus y en la preparación documental previa. P1 recupera contexto desde un corpus `.txt`, mientras que P2 recupera contexto desde un corpus Nabsic semiestructurado y multi-formato. La diferenciación explícita por tipo de tarea se introduce únicamente en P3, ver Apéndice D.2.

4.2.7. Generación de la respuesta

La generación de la respuesta se realiza a partir del prompt construido por cada pipeline. Inicialmente, se planteó utilizar dos modelos generadores: *gemini-2.5-flash*, mediante API externa, y *HuggingFaceH4/zephyr-7b-beta*, ejecutado localmente en Colab mediante Hugging Face. Esta configuración habría permitido comparar los pipelines con dos generadores distintos.

Sin embargo, durante la extracción de los resultados finales, los límites de uso de los modelos gratuitos de Gemini resultaron ser más restrictivos de lo observado en las pruebas preliminares. Esto dificultaba completar la matriz experimental de forma homogénea para todos los pipelines y casos del dataset. Por este motivo, Gemini se descartó como generador en la evaluación principal.

La evaluación final se centra en el modo `local_hf`, usando el modelo *HuggingFaceH4/zephyr-7b-beta*. Esta decisión evita depender de cuotas de una API externa y mantiene constante el modelo generador en la comparación entre pipelines. Como contrapartida, la inferencia local queda condicionada por los recursos disponibles en Google Colab, especialmente por la disponibilidad de GPU, necesaria para obtener tiempos de generación razonables.

De este modo, la comparación final se concentra en los tres pipelines RAG definidos, usando el mismo dataset, el mismo modelo generador y la misma estrategia de evaluación. Así, las diferencias observadas pueden atribuirse principalmente a las

decisiones de corpus y construcción del contexto.

4.2.8. Persistencia de resultados

Cada ejecución genera outputs en `eval_outputs/`. En ellos se guardan respuestas, contextos recuperados, fuentes, parámetros de configuración, errores de generación y datasets preparados para RAGAS. Esta persistencia convierte cada ejecución en un experimento auditable. La Tabla 4.2 resume los principales campos producidos por el pipeline.

Tabla 4.2: Salidas principales del pipeline RAG.

Campo	Valor experimental
<code>id</code>	Identifica el caso evaluado.
<code>task_type</code>	Permite agrupar resultados por generación, explicación o detección de errores.
<code>response</code>	Contiene la respuesta generada por el sistema.
<code>retrieved_contexts</code>	Conserva los fragmentos usados como evidencia.
<code>retrieved_sources</code>	Permite auditar de qué documentos proceden los fragmentos.
<code>rag_error</code>	Registra fallos de generación, cuota o proveedor.
<code>chunk_size</code>	Documenta el tamaño de fragmentación.
<code>chunk_overlap</code>	Solapamiento entre fragmentos.
<code>num_docs_for_answer</code>	Número de documentos incluidos finalmente en el prompt.
<code>generator_mode</code>	Modo de generación utilizado.
<code>generator_model</code>	Modelo generador empleado.

4.2.9. Preparación para evaluación

Tras generar las respuestas, el notebook prepara un dataset compatible con RAGAS. Cada muestra combina pregunta, respuesta, contextos recuperados y referencia. Esta estructura permite calcular métricas como `context_recall`, `faithfulness` y `factual_correctness`.

El valor de esta fase no reside solo en obtener una puntuación, sino en separar dimensiones del problema. Una métrica baja de recuperación puede indicar que el corpus, el chunking o el índice no están aportando información suficiente. Una métrica

baja de fidelidad puede indicar que el modelo se aparta del contexto. Una baja corrección factual puede indicar que, aunque el contexto exista, la respuesta no satisface la referencia esperada.

Sin embargo, cabe recalcar que RAGAS está más enfocado a evaluar sistemas RAG orientados principalmente a texto, no a generación de código. Por lo tanto, se han definido métricas complementarias que serán extraídas por evaluación humana.

4.2.10. Trazabilidad y reproducibilidad

En la ejecución de los pipelines se registran los parámetros principales de cada ejecución. Esto es esencial para comparar experimentos de forma justa: una diferencia en resultados solo es interpretable si se conocen las diferencias de configuración. En este sentido, el pipeline actúa como un experimento reproducible de Ciencia de Datos, con entradas, transformaciones, parámetros y salidas. La Tabla 4.3 resume el flujo completo.

Tabla 4.3: Etapas del pipeline RAG desde la perspectiva de Ciencia de Datos.

Etapas	Transformación	Artefacto resultante
Ingesta	Lectura de PDF y Mark-down.	Documentos con texto y metadatos.
Fragmentación	División con tamaño y solapamiento configurables.	Chunks documentales.
Vectorización	Aplicación del modelo de embeddings.	Vectores semánticos.
Indexación	Almacenamiento en FAISS.	Índice vectorial local.
Consulta	Embedding de <code>user_input</code> .	Vector de consulta.
Retrieval	Búsqueda de vecinos cercanos.	Contextos y fuentes recuperadas.
Prompting	Composición de instrucciones, contexto y pregunta.	Prompt final.
Generación	Llamada al generador.	Respuesta y error si existe.
Evaluación	Conversión a formato RAGAS y cálculo de métricas.	Scores automáticos y CSV de resultados.

4.2.11. Valor de la baseline en el proyecto

La baseline es valiosa porque fija un punto de partida medible. Sin ella, cualquier mejora posterior sería difícil de atribuir a una decisión concreta. Con ella, se puede estudiar de forma controlada si un cambio en el tamaño de chunk, en el número de documentos recuperados, en la composición del corpus o en el prompt mejora realmente la calidad de las respuestas.

Además, la baseline permite analizar errores con granularidad. El sistema no solo devuelve una respuesta, sino que deja trazas de qué datos fueron recuperados, qué fuentes influyeron en la generación, qué parámetros se usaron y qué fallos aparecieron. Esta trazabilidad es especialmente importante en un dominio técnico como Nabsic, donde la confianza en la respuesta depende de que pueda justificarse contra documentación concreta.

4.3. Pipeline P3: Context Engineering con sesiones y memoria

El pipeline P3 mantiene la base RAG de P2, es decir, utiliza el corpus Nabsic semiestructurado, embeddings e índice FAISS para recuperar evidencia documental. La diferencia principal está en que no construye el prompt únicamente concatenando los documentos recuperados, sino que introduce una capa explícita de *Context Engineering* [24]. Esta capa organiza la información disponible antes de llamar al modelo generador, combinando sesión, memoria, documentos recuperados, priorización de fuentes y una instrucción adaptada al tipo de tarea.

El objetivo de esta variante, como se ha mencionado anteriormente, no es cambiar el modelo generador ni el dataset, sino estudiar si una construcción más cuidada del contexto mejora las respuestas en un dominio técnico especializado como Nabsic-Enablon.

4.3.1. Sesiones

En P3, cada caso del dataset se encapsula en una sesión independiente. La sesión actúa como contenedor del estado inmediato de una interacción: identifica el caso evaluado, el tipo de tarea, los eventos de usuario y asistente, y un resumen compacto del turno. En una aplicación conversacional real, una sesión podría acumular varios turnos y mantener continuidad entre preguntas relacionadas.

Sin embargo, es relevante mencionar que en la evaluación de este trabajo cada fila del benchmark se trata como una sesión separada. Esto permite introducir el concepto de sesión sin que una pregunta del dataset contamine las siguientes. De este modo, P3 puede utilizar estructura conversacional dentro de cada caso, pero no aprende información procedente de otros casos de evaluación.

4.3.2. Memoria

Además de la evidencia documental recuperada mediante FAISS, P3 incorpora una memoria ligera formada por registros compactos. Estas memorias no sustituyen a los documentos del corpus, sino que actúan como reglas o recordatorios generales para orientar el comportamiento del sistema. En la implementación, las memorias incluyen información como el tipo de memoria, su contenido, etiquetas, importancia, fuente y procedencia.

El pipeline inicializa memorias semilla de tipo procedimental y declarativo (Apéndice E). Estas memorias recogen criterios generales del diseño del TFM, por ejemplo: preferir APIs documentadas, reconocer explícitamente cuándo el contexto no es suficiente, explicar código siguiendo el orden de ejecución o priorizar manuales oficiales y documentación de objetos frente a catálogos amplios de tablas Enablon.

Una decisión relevante es que estas memorias semilla no se extraen de las referencias esperadas del dataset. Es decir, no contienen respuestas del benchmark ni soluciones específicas a los casos evaluados.

Se define un modo de escritura de memoria, que por defecto es de sesión, de forma que las memorias generadas tras una respuesta no se promocionan a memoria persistente para casos posteriores. Esto evita que el sistema aprenda de una muestra y utilice esa información en las siguientes, lo que mantendría la comparación experimental en condiciones más controladas. Sin embargo, sería un modo interesante a contemplar para trabajos futuros en el que se compare con la escritura de memoria activa.

4.3.3. Recuperación de memoria

Para cada pregunta, P3 recupera memorias candidatas combinando varios criterios: similitud semántica con la consulta, importancia asignada a la memoria, recencia y coincidencia de etiquetas con el tipo de tarea. Esta recuperación se restringe a memorias de alcance general de aplicación y a memorias de la sesión activa. Así, el sistema puede recuperar reglas útiles para la tarea sin mezclar información de sesiones anteriores del benchmark.

La memoria cumple una función distinta a la recuperación documental. Mientras que FAISS recupera fragmentos del corpus Nabsic-Enablon que sirven como evidencia técnica, la memoria aporta criterios de comportamiento y organización del contexto. Esta separación permite analizar si la respuesta se apoya principalmente en documentos, en memoria o en una combinación de ambos.

4.3.4. Recuperación y priorización documental

La recuperación documental de P3 sigue utilizando FAISS, igual que P1 y P2. No obstante, P3 añade una fase posterior de ordenación y selección. En lugar de quedarse únicamente con el orden de similitud vectorial, el pipeline ajusta la prioridad de los documentos según su procedencia y su utilidad esperada para la tarea.

En general, se favorecen fuentes más directamente relacionadas con la documentación técnica de Nabsic, como manuales, documentación detallada y documentación de objetos. Los ejemplos se consideran especialmente útiles en tareas de generación cuando coinciden con la consulta. En cambio, los catálogos amplios de módulos Enablon se tratan con más cautela, ya que pueden aportar información útil sobre tablas y campos, pero también introducir ruido si desplazan documentación más específica sobre sintaxis, objetos o funciones.

Esta priorización busca que el prompt final no contenga simplemente los fragmentos más similares, sino los fragmentos más adecuados para responder a la tarea. Además, P3 utiliza presupuestos aproximados de tokens para repartir el espacio disponible entre memoria y evidencia documental. Esta decisión está relacionada con la limitación práctica de la ventana de contexto: no toda la información recuperada puede incluirse en el prompt final, por lo que es necesario seleccionar y ordenar.

4.3.5. Prompt específico por tipo de tarea

Otra diferencia importante de P3 es que el prompt se adapta al tipo de tarea. P1 y P2 utilizan una instrucción base común, mientras que P3 introduce instrucciones diferenciadas para generación, explicación y detección o corrección de errores.

En tareas de generación, el prompt orienta al modelo a producir un fragmento Nabsic breve y basado en APIs o patrones presentes en el contexto. En tareas de explicación, se prioriza describir el comportamiento del código paso a paso y calcular el resultado observable cuando la evidencia lo permite. En tareas de corrección o detección de errores, la instrucción pide identificar el problema principal, explicar por qué ocurre y proponer una corrección concreta. Esta diferenciación permite adaptar el

comportamiento del modelo a objetivos de programación distintos [1].

De forma esquemática, el prompt final de P3 integra los siguientes bloques:

$\text{Prompt}_{P3} = \text{instrucciones globales} + \text{tipo de tarea} + \text{resumen de sesión} + \text{memoria relevante} + \text{evidencia documental} + \text{pregunta}$

Esta estructura hace que P3 sea más complejo que P1 y P2, pero también más expresivo. La hipótesis experimental es que, en un dominio como Nabsic, una respuesta puede mejorar no solo por recuperar documentos relevantes, sino por organizar mejor qué información entra al prompt y cómo se presenta al modelo generador.

4.3.6. Trazabilidad del contexto

P3 conserva salidas adicionales respecto a los pipelines base. Además de la respuesta, los contextos y las fuentes recuperadas, registra información específica sobre la construcción del contexto: memorias seleccionadas, documentos seleccionados, fuentes priorizadas, estimación de tokens de memoria, estimación de tokens documentales y traza del contexto final.

Estas columnas no modifican la evaluación final, que se realiza con las mismas métricas automáticas y la misma rúbrica manual que en P1 y P2. Sin embargo, resultan útiles para interpretar los resultados. Por ejemplo, si P3 falla en una pregunta, puede revisarse si el problema procede de una mala recuperación documental, de una memoria poco útil, de una priorización incorrecta o de una generación deficiente pese a disponer de buen contexto.

En resumen, P3 representa la variante más rica del sistema: mantiene la recuperación documental de P2, pero añade sesiones, memoria, priorización de fuentes, presupuestos de contexto, prompting específico por tarea y trazabilidad detallada. Esto permite comparar un RAG semiestructurado convencional frente a una estrategia donde el contexto se construye de forma más explícita antes de generar la respuesta.

4.4. Componentes implementados

La tabla a continuación (Tabla 4.4) resume los principales componentes definidos en el proyecto.

Tabla 4.4: Componentes técnicos principales usados en los pipelines.

Componente	Configuración actual
Orquestación	LangChain
Carga PDF	PyPDFLoader
Datos tabulares	Pandas para datasets y outputs
Chunking	RecursiveCharacterTextSplitter
Embeddings	HuggingFaceEmbeddings
Modelo de embeddings	thenlper/gte-small
Índice vectorial	FAISS local
Chunk size base	512 caracteres
Chunk overlap base	80 caracteres
Documentos recuperados	8
Documentos usados para responder	5
Generador principal	<i>HuggingFaceH4/zephyr-7b-beta</i>

Capítulo 5

Metodología experimental

5.1. Diseño experimental

La metodología final consiste en ejecutar los tres pipelines descritos en el Capítulo 4 sobre un dataset fijo de 24 prompts y comparar su comportamiento manteniendo constante el modelo generador. La hipótesis de trabajo es que, con un mismo modelo generador, las diferencias en corpus y construcción del contexto pueden traducirse en respuestas más correctas, útiles y fundamentadas [10], [25], [26].

El diseño evita una matriz excesivamente amplia de variantes de chunking, top-k, modelos o prompts. Estas dimensiones se documentan como decisiones técnicas de implementación, pero la comparación final se concentra en tres niveles de complejidad contextual de cada uno de los tres pipelines ya descritos.

Inicialmente, se estimó hacer uso de dos modelos, Gemini = *gemini-2.5-flash* y Huggingface = *HuggingFaceH4/zephyr-7b-beta*. Sin embargo, durante la extracción de resultados finales, los límites de los modelos gratuitos de Gemini resultaron ser mucho más limitantes de lo inicialmente visto y estimado.

Por lo tanto, en la definición final del experimento, los tres pipelines se ejecutan con el mismo dataset, el mismo modelo generador y el mismo procedimiento de evaluación. De esta forma, el experimento centra la comparación en dos factores principales: la calidad del corpus y la estrategia de construcción del contexto. En particular, la comparación P1–P2 mantiene constante la estrategia de prompting y se centra en el efecto del corpus semiestructurado. La comparación P2–P3 concentra los cambios asociados a Context Engineering.

5.2. Dataset final

El dataset final usado en el experimento contiene 24 prompts orientados a programación Nabsic, ya descrito en la sección 3.3. La proporción se fija para priorizar la generación de código, sin perder tareas de comprensión y depuración:

Tabla 5.1: Distribución del dataset final de evaluación.

Tipo de tarea	Casos	Objetivo
Generación de código	12	Medir capacidad de producir fragmentos Nabsic útiles y sintácticamente plausibles.
Explicación de código	6	Medir comprensión de fragmentos, conceptos y comportamiento esperado.
Corrección de código	6	Medir detección de errores, riesgos y propuesta de corrección.

El esquema del dataset final de pruebas puede observarse en el Apéndice: C

5.3. Evaluación automática con RAGAS

La evaluación automática se realiza con RAGAS (*Retrieval-Augmented Generation Assessment*), un marco orientado a evaluar sistemas RAG separando distintas dimensiones del proceso: la recuperación del contexto, la fidelidad de la respuesta respecto a la evidencia recuperada y la corrección factual de la salida generada [14]. Esta separación resulta útil en este trabajo, ya que una respuesta incorrecta puede deberse a problemas distintos: recuperación insuficiente, contexto poco relevante, mal uso de la evidencia o generación de una respuesta que no coincide con el comportamiento esperado.

En los notebooks del proyecto, RAGAS se aplica después de ejecutar cada pipeline sobre el dataset final. La generación de respuestas y la evaluación automática se mantienen como fases independientes. Primero, cada notebook ejecuta `run_eval_dataset(max_rows=24)`, guarda las respuestas completas y crea un archivo `*_ragas_dataset.jsonl`. Este archivo contiene, para cada caso, la consulta del usuario, la respuesta generada, los contextos recuperados por el pipeline y la referencia esperada del dataset. Después, la función `evaluate_latest_ragas()` carga ese JSONL y llama a RAGAS mediante `evaluate()`, usando el mismo conjunto de métricas para los tres pipelines.

Un aspecto importante de la configuración final es la separación entre modelo generador y modelo evaluador. El modelo generador comparado en los tres pipelines es *HuggingFaceH4/zephyr-7b-beta*. En cambio, RAGAS necesita un modelo juez independiente para valorar las respuestas.

Inicialmente se intentó utilizar Gemini también como modelo evaluador, pero los límites de uso de la modalidad gratuita impidieron completar la evaluación automática de forma estable y homogénea. Por este motivo, la evaluación RAGAS final se realizó con el modelo de OpenAI *gpt-5.4* facilitado por la empresa (All4). Este modelo se emplea únicamente como evaluador automático y se mantiene constante para todas las ejecuciones, por lo que no afecta a la comparación del modelo generador entre pipelines.

Las métricas automáticas utilizadas son las siguientes:

Tabla 5.2: Métricas RAGAS utilizadas en la evaluación automática.

Métrica	Qué evalúa	Interpretación en el proyecto
<code>context_recall</code>	Estima si los contextos recuperados contienen la información necesaria para cubrir la referencia esperada.	Permite analizar la calidad de la recuperación y detectar si el corpus, el chunking o FAISS dejan fuera información relevante.
<code>faithfulness</code>	Estima si las afirmaciones de la respuesta están apoyadas por los contextos recuperados.	Permite detectar respuestas plausibles pero poco fundamentadas en la evidencia documental.
<code>factual_correctness</code>	Compara la respuesta generada con la referencia esperada mediante solapamiento factual; en este trabajo se utiliza el modo F1.	Proporciona una aproximación automática a la coincidencia entre la respuesta del sistema y la solución esperada.

Los resultados se guardan como archivos CSV y se consolidan para obtener medias globales y análisis por tipo de tarea, presentados en el Capítulo 6.

Las puntuaciones RAGAS se interpretan como evidencia complementaria, no como veredicto único. Aunque RAGAS permite comparar de forma homogénea recuperación, fidelidad y corrección factual, el benchmark de este TFM incluye generación,

explicación y corrección de código Nabsic. En este tipo de tareas pueden existir varias respuestas técnicamente válidas que no coincidan literalmente con la referencia. Además, la utilidad de una respuesta depende de aspectos difíciles de capturar mediante métricas automáticas, como el uso correcto de objetos del dominio, la sintaxis Nabsic, la adecuación a convenciones internas y la capacidad de detectar errores sutiles. Por este motivo, la evaluación automática se complementa con una rúbrica manual específica.

Esta decisión está alineada con trabajos recientes sobre RAG aplicado a código, donde la utilidad del sistema depende tanto de recuperar contextos de calidad como de integrarlos correctamente en una solución programática [2]. Aunque CodeRAG-Bench no evalúa Nabsic, sí refuerza la necesidad de combinar métricas automáticas con revisión humana cuando el objetivo final es valorar respuestas de programación.

5.4. Evaluación manual

Dado que Nabsic es un dominio especializado y la tarea incluye generación, explicación y corrección de código, la evaluación automática se complementa con una revisión manual basada en rúbrica [2]. Esta revisión se diseñó para valorar aspectos que RAGAS no captura de forma fiable: corrección sintáctica, uso adecuado de objetos y funciones Nabsic, utilidad práctica de la respuesta, mantenimiento de la intención original del prompt y adecuación al estilo del lenguaje.

La evaluación manual se aplicó a las 72 respuestas de la matriz final: 24 casos del dataset evaluados en los tres pipelines P1, P2 y P3. Para cada respuesta se tuvieron en cuenta seis elementos: el prompt original, el tipo de tarea, el comportamiento esperado, los criterios funcionales definidos en el dataset, la respuesta generada y los contextos recuperados por el pipeline. La puntuación se registró junto con una decisión práctica y comentarios cuando la diferencia entre pipelines requería justificación.

La rúbrica se organiza en tres dimensiones principales, con un máximo de 10 puntos por respuesta. La dimensión de corrección técnica y utilidad se adapta al tipo de tarea: en generación se valora si el código resuelve el objetivo pedido, usa estructuras Nabsic adecuadas y evita errores funcionales; en explicación se valora si la respuesta identifica el propósito, describe el flujo y resulta útil para un desarrollador; en corrección o detección de errores se valora si localiza el error principal, lo justifica y propone una modificación concreta. La rúbrica completa se incluye en el Apéndice B.

Tabla 5.3: Resumen de la rúbrica manual aplicada.

Criterio	Descripción	Rango
Corrección técnica y utilidad	Evalúa si la respuesta cumple el objetivo de la tarea. Se puntúa con criterios específicos para generación, explicación y corrección/detección de errores.	0–6
Uso del contexto	Evalúa si la respuesta se apoya en fragmentos recuperados relevantes, evita contradicciones con la documentación y no introduce APIs no respaldadas.	0–3
Estilo Nabsic	Evalúa si el código o la explicación usan convenciones plausibles del dominio y no parecen trasladados de otro lenguaje generalista.	0–1

Además de la puntuación numérica, cada respuesta se clasificó con una decisión práctica: *Deficiente* para puntuaciones entre 0 y 3, *Insuficiente/parcial* para 4 y 5, *Aceptable* para 6 y 7, *Buena* para 8 y 9, y *Excelente* para 10. Esta clasificación facilita la interpretación de los resultados agregados, especialmente el porcentaje de respuestas con puntuación igual o superior a 7,5 utilizado en el análisis final.

Capítulo 6

Resultados

En este capítulo se presentan los resultados obtenidos tras la evaluación de los tres pipelines desarrollados para el sistema RAG aplicado a consultas sobre Nabsic. El objetivo principal de esta evaluación es comparar de forma sistemática el rendimiento de las distintas configuraciones propuestas, tanto desde el punto de vista automático, mediante métricas RAGAS, como desde el punto de vista cualitativo, mediante una rúbrica manual específica para tareas de generación, explicación y corrección de código.

La evaluación se llevó a cabo sobre un conjunto final de 24 prompts. Estos prompts se distribuyen en tres tipos de tarea: 12 casos de generación de código, 6 casos de explicación de código y 6 casos de detección o corrección de errores. Cada caso fue ejecutado sobre los tres pipelines, dando lugar a un total de 72 respuestas evaluadas. Esta configuración permite analizar no solo el rendimiento global de cada pipeline, sino también su comportamiento específico en función del tipo de tarea.

Los pipelines comparados fueron los siguientes:

- **Pipeline 1 (P1):** configuración base del sistema RAG sobre corpus de texto plano.
- **Pipeline 2 (P2):** configuración con incorporación de corpus Nabsic semiestructurado, manteniendo la estrategia base de prompting.
- **Pipeline 3 (P3):** configuración avanzada, que incorpora sesiones, memoria, priorización del contexto y prompting específico por tipo de tarea.

6.1. Resultados de la evaluación manual

La evaluación manual se realizó mediante una rúbrica específica para el dominio Nabsic. La puntuación total de cada respuesta se calculó sobre 10 puntos, considerando tres dimensiones principales: corrección técnica y utilidad, uso del contexto recuperado y adecuación al estilo del lenguaje Nabsic.

La Tabla 6.1 resume los resultados medios obtenidos por cada pipeline.

Tabla 6.1: Resultados globales de la evaluación manual por pipeline.

Pipeline	Total	Corrección	Contexto	Estilo	% $\geq 7,5$
P1	4,89	2,55	1,79	0,55	25,0 %
P2	5,13	2,93	1,67	0,53	33,3 %
P3	5,89	3,29	1,83	0,78	41,7 %

Los resultados muestran una mejora progresiva en la puntuación media total desde P1 hasta P3. El pipeline base obtiene una puntuación media de 4,89 sobre 10, mientras que P2 alcanza 5,13 y P3 obtiene 5,89. Por tanto, P2 mejora ligeramente respecto a P1, con una diferencia de 0,24 puntos, mientras que P3 mejora en 0,77 puntos respecto a P2 y en 1,00 punto respecto a P1.

Esta evolución indica que la simple incorporación del corpus semiestructurado en P2 produce una mejora moderada, especialmente en la dimensión de corrección técnica. Sin embargo, la mejora más clara se observa al pasar de P2 a P3. Esto sugiere que el rendimiento del sistema no depende únicamente de disponer de más documentación, sino también de cómo se selecciona, prioriza y utiliza el contexto dentro del prompt final.

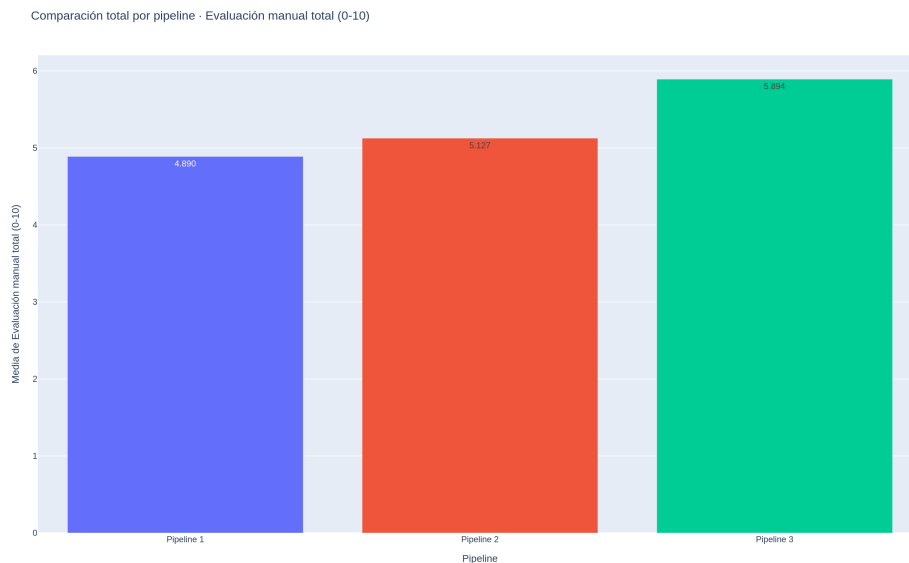


Figura 6.1: Comparación global de la puntuación manual media por pipeline.

En cuanto al porcentaje de respuestas consideradas buenas o excelentes, utilizando como umbral una puntuación igual o superior a 7,5, también se observa una mejora progresiva. P1 alcanza este umbral en el 25,0 % de los casos, P2 en el 33,3 % y P3 en el 41,7 %. Esto refuerza la idea de que P3 no solo mejora la media global, sino que también aumenta la proporción de respuestas de alta calidad.

6.2. Resultados por tipo de tarea

El análisis por tipo de tarea permite identificar diferencias importantes entre los pipelines. La Tabla 6.2 muestra la puntuación manual media para generación, explicación y corrección de errores.

Tabla 6.2: Puntuación manual media por tipo de tarea y pipeline.

Pipeline	Generación	Explicación	Corrección
P1	4,16	6,65	4,58
P2	4,45	7,10	4,52
P3	6,86	5,88	3,97

Los resultados muestran que P3 obtiene el mejor rendimiento en las tareas de generación de código, con una puntuación media de 6,86 frente a 4,16 en P1 y 4,45 en P2. Esta diferencia es relevante, ya que la generación de código es el tipo de tarea

más complejo y también el más representado en el dataset. La mejora de P3 en este tipo de tarea puede atribuirse a la combinación de una estrategia de contexto más rica (especialmente en este caso existen varias memorias semilla para la generación de código), que ayuda a la priorización de información relevante y el uso de instrucciones específicas adaptadas a la generación de código Nabsic.

Por el contrario, las tareas de explicación de código presentan un comportamiento distinto. En este caso, P2 obtiene la mejor puntuación media, con 7,10 puntos, seguido de P1 con 6,65 y P3 con 5,88. Esto indica que, para tareas explicativas, la estrategia más avanzada de P3 no necesariamente produce mejores respuestas. Una posible explicación es que las respuestas de explicación requieren menos generación estructurada y dependen más de interpretar correctamente el fragmento de código proporcionado. En estos casos, una configuración más simple y centrada en el contexto documental puede ser suficiente e incluso más estable.

En las tareas de corrección o detección de errores, los tres pipelines presentan resultados más bajos. P1 obtiene una media de 4,58, P2 alcanza 4,52 y P3 desciende a 3,97. Este resultado sugiere que la detección de errores en código Nabsic sigue siendo una tarea especialmente difícil para el sistema. A diferencia de la generación o explicación, la corrección requiere identificar con precisión la causa del fallo, distinguir errores reales de posibles variaciones válidas del lenguaje y proponer una modificación mínima. Los resultados indican que ninguno de los pipelines resuelve esta tarea de forma suficientemente robusta. Estas puntuaciones pueden estar relacionadas también con la ausencia de ejemplos específicos de resolución de errores en el corpus preparado.

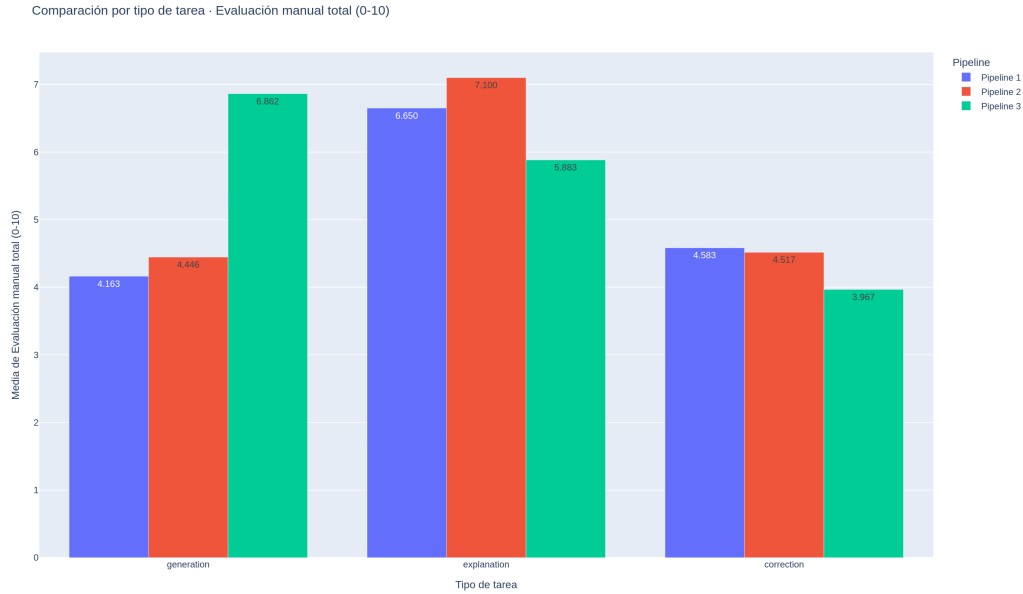


Figura 6.2: Comparación de la evaluación manual por tipo de tarea.

En conjunto, el análisis por tipo de tarea muestra que P3 es la mejor configuración para generación de código, P2 es la más adecuada para explicación y ninguno de los tres pipelines obtiene un rendimiento claramente satisfactorio en corrección de errores. Esto evidencia que el diseño óptimo de un sistema RAG puede depender del tipo de tarea, y no únicamente de una configuración global única.

6.3. Resultados de la evaluación automática con RAGAS

Además de la evaluación manual, se calcularon métricas automáticas mediante RAGAS. Las métricas consideradas fueron *context recall*, *faithfulness* y *factual correctness* en modo F1. Estas métricas permiten analizar, respectivamente, la capacidad del sistema para recuperar contexto relevante, la fidelidad de la respuesta respecto al contexto utilizado y la coincidencia factual con la referencia esperada.

La Tabla 6.3 presenta los resultados globales medios por pipeline.

Tabla 6.3: Resultados globales de métricas RAGAS por pipeline.

Pipeline	Context Recall	Faithfulness	Factual Correctness	Media RAGAS
P1	0,417	0,245	0,257	0,306
P2	0,319	0,239	0,262	0,273
P3	0,465	0,346	0,206	0,339

De acuerdo con las métricas RAGAS, P3 obtiene la mejor media global, con un valor de 0,339. P1 alcanza 0,306 y P2 obtiene 0,273. La diferencia entre P3 y P1 es menor que la observada en la evaluación manual, pero mantiene la misma tendencia general: P3 es el pipeline con mejor rendimiento global.

El resultado más destacado de P3 se observa en *context recall* y *faithfulness*. P3 alcanza un *context recall* de 0,465, superior a P1 y P2, y una *faithfulness* de 0,346, también la más alta de los tres pipelines. Esto indica que la configuración avanzada recupera y utiliza mejor la información contextual en términos globales.

Sin embargo, P3 obtiene el valor más bajo en *factual correctness*, con 0,206 frente a 0,257 en P1 y 0,262 en P2. Este resultado debe interpretarse con cautela. En tareas de programación, especialmente cuando existen varias soluciones válidas, las métricas automáticas basadas en comparación con una referencia pueden penalizar respuestas técnicamente correctas pero formuladas de forma distinta a la solución esperada. Por este motivo, la evaluación RAGAS se utiliza como complemento de la evaluación manual, pero no como único criterio para determinar la calidad final de las respuestas.



Figura 6.3: Comparación global de la media de métricas RAGAS por pipeline.

6.4. Análisis de RAGAS por tipo de tarea

La Tabla 6.4 resume la media de las tres métricas RAGAS por tipo de tarea.

Tabla 6.4: Media RAGAS por tipo de tarea y pipeline.

Pipeline	Generación	Explicación	Corrección
P1	0,284	0,439	0,216
P2	0,198	0,475	0,222
P3	0,326	0,419	0,284

En las tareas de generación, P3 obtiene la mejor media RAGAS, con 0,326. Este resultado es coherente con la evaluación manual, donde P3 también era claramente superior para este tipo de tarea. Por tanto, tanto la evaluación automática como la manual apuntan a que la configuración avanzada resulta especialmente beneficiosa para la generación de código.

En las tareas de explicación, P2 obtiene la mejor media RAGAS, con 0,475, seguida de P1 con 0,439 y P3 con 0,419. Este resultado también coincide con la evaluación manual, donde P2 era el pipeline con mejor rendimiento en explicaciones. La consistencia entre ambas evaluaciones refuerza la conclusión de que P2 es una configuración adecuada para tareas explicativas.

En las tareas de corrección, P3 obtiene la mejor media RAGAS, con 0,284, mientras que P1 y P2 obtienen 0,216 y 0,222 respectivamente. No obstante, esta conclusión difiere parcialmente de la evaluación manual, donde P3 era el pipeline con menor puntuación en corrección. Esta diferencia muestra una limitación importante de las métricas automáticas en tareas de detección de errores: una respuesta puede coincidir parcialmente con el contexto o con la referencia, pero no identificar de forma suficientemente útil el problema real desde el punto de vista del evaluador humano.

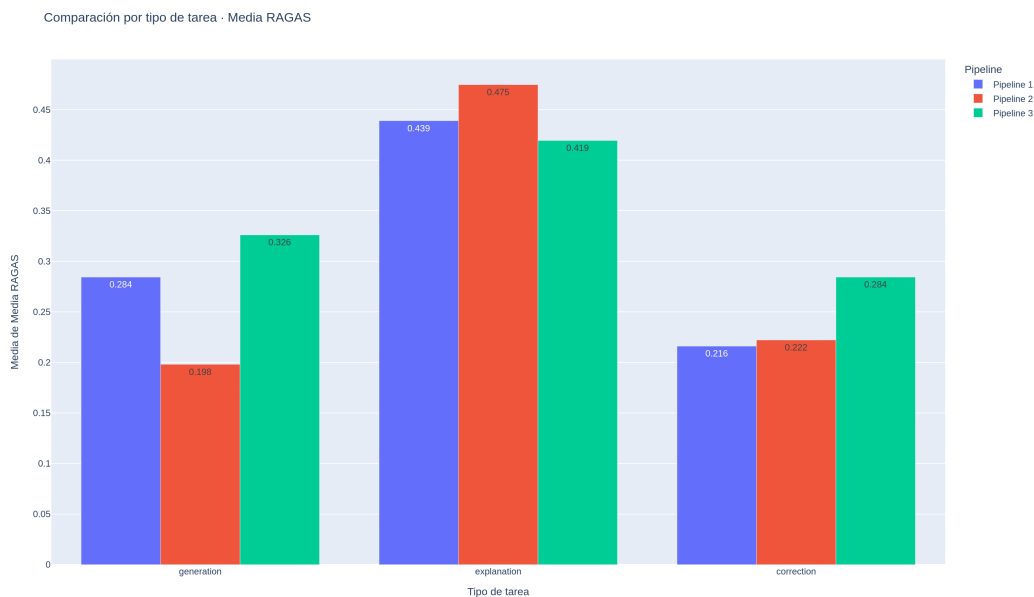


Figura 6.4: Comparación de la media RAGAS por tipo de tarea.

6.5. Comparación entre P1 y P2: efecto del corpus semiestructurado

La comparación entre P1 y P2 permite aislar el efecto de incorporar un corpus semiestructurado. En términos de evaluación manual, P2 mejora ligeramente respecto a P1, pasando de 4,89 a 5,13 puntos de media. La mejora se observa principalmente en la dimensión de corrección técnica y utilidad, que aumenta de 2,55 a 2,93.

Este resultado sugiere que el corpus semiestructurado aporta información útil para formular respuestas más completas y técnicamente más adecuadas. Sin embargo, la mejora no es homogénea en todas las dimensiones. El uso del contexto baja ligeramente de 1,79 en P1 a 1,67 en P2, y la adecuación al estilo Nabsic también disminuye de 0,55 a 0,53. Por tanto, aunque P2 introduce más información estructurada, esto no siempre se traduce en un mejor aprovechamiento del contexto dentro de la respuesta final.

El análisis por tipo de tarea confirma esta interpretación. En generación, P2 mejora ligeramente respecto a P1, pasando de 4,16 a 4,45. En explicación, la mejora es más clara, pasando de 6,65 a 7,10. En cambio, en corrección, P2 no mejora a P1 y obtiene una puntuación muy similar, 4,52 frente a 4,58.

Las métricas RAGAS muestran una lectura algo diferente. La media global RAGAS de P2 es inferior a la de P1, con 0,273 frente a 0,306. Esta diferencia se debe

especialmente a la caída en *context recall*, que pasa de 0,417 en P1 a 0,319 en P2. Por tanto, aunque P2 mejora ligeramente en evaluación manual, no necesariamente recupera mejor la información esperada por las métricas automáticas. Esto puede deberse a que el nuevo corpus semiestructurado introduce información adicional útil, pero también puede aumentar la dispersión del contexto recuperado.

En resumen, P2 aporta una mejora moderada respecto a P1, especialmente en explicación de código y corrección técnica. No obstante, el efecto del corpus semiestructurado por sí solo es limitado y no garantiza una mejora sistemática en recuperación ni en fidelidad automática.

6.6. Comparación entre P2 y P3: efecto de sesiones, memoria y prompting específico

La comparación entre P2 y P3 permite analizar el efecto conjunto de las mejoras incorporadas en el pipeline avanzado. Estas mejoras incluyen el uso de sesiones, memoria, priorización del contexto y prompting específico en función del tipo de tarea.

Desde el punto de vista de la evaluación manual, P3 mejora de forma clara respecto a P2. La puntuación media total aumenta de 5,13 a 5,89, lo que supone una mejora de 0,77 puntos. También mejoran las tres dimensiones de la rúbrica: corrección técnica, uso del contexto y estilo Nabsic. En particular, la puntuación de estilo aumenta de 0,53 a 0,78, lo que indica que P3 produce respuestas más alineadas con las convenciones del lenguaje y con el formato esperado.

La mejora de P3 es especialmente significativa en generación de código. En este tipo de tarea, la puntuación manual pasa de 4,45 en P2 a 6,86 en P3. Este incremento sugiere que la combinación de contexto priorizado y prompting específico ayuda al modelo a producir código más útil, más estructurado y más cercano al estilo esperado.

Sin embargo, P3 no mejora en todos los tipos de tarea. En explicación, P3 obtiene 5,88 frente a 7,10 en P2, y en corrección obtiene 3,97 frente a 4,52. Esto indica que la estrategia avanzada está especialmente optimizada para generación, pero puede introducir complejidad innecesaria o respuestas menos directas en tareas donde se requiere una explicación precisa o una corrección puntual.

Las métricas RAGAS también muestran una mejora global de P3 respecto a P2. La media RAGAS pasa de 0,273 en P2 a 0,339 en P3. Esta mejora se debe principalmente al aumento en *context recall*, que pasa de 0,319 a 0,465, y en *faithfulness*, que pasa de 0,239 a 0,346. Por tanto, P3 no solo obtiene mejores puntuaciones manua-

les, sino que también muestra una mayor capacidad para recuperar y utilizar contexto relevante.

No obstante, P3 presenta una menor *factual correctness* automática que P2. Este resultado confirma que las métricas automáticas deben interpretarse de forma complementaria a la revisión manual, especialmente en tareas de código. La respuesta generada puede ser técnicamente válida aunque no coincida exactamente con la referencia utilizada por RAGAS.

6.7. Resultados por caso

Además del análisis agregado, se analizaron los resultados a nivel de caso individual. Este análisis permite identificar si la mejora de un pipeline se debe a un comportamiento estable o a un número reducido de casos con puntuaciones muy altas.

En la evaluación manual, P1 y P3 empatan como mejores pipelines en número de casos, con 9 casos cada uno en los que obtienen la mejor puntuación. P2 es el mejor en 6 casos. Aunque P3 presenta la mejor media global, este resultado muestra que su superioridad no es absoluta en todos los prompts. Existen casos concretos en los que P1 o P2 resultan más adecuados, especialmente en tareas de explicación o en consultas donde un contexto más simple es suficiente.

En las métricas RAGAS, P3 es el pipeline con mejor media por caso en 11 de los 24 prompts, seguido de P1 en 9 casos y P2 en 4 casos. Esto refuerza la conclusión de que P3 es el pipeline más competitivo en términos globales, aunque no domina de forma uniforme todos los escenarios evaluados.



Figura 6.5: Comparación de la puntuación manual por caso y pipeline.

El análisis por caso también muestra que la variabilidad entre prompts es elevada. Esto es esperable en un benchmark de programación, ya que cada prompt puede requerir diferentes capacidades: conocimiento de sintaxis, recuperación de documentación, adaptación al estilo del lenguaje, explicación de comportamiento o identificación de errores. Por tanto, la evaluación agregada debe interpretarse junto con el análisis por tipo de tarea.

6.8. Análisis de tiempos de ejecución

Además de la calidad de las respuestas, se analizaron los tiempos medios de ejecución. La Tabla 6.5 muestra el tiempo total medio por pipeline.

Tabla 6.5: Tiempo medio de ejecución por pipeline.

Pipeline	Tiempo total medio (s)	Tiempo generación medio (s)
P1	45,09	45,00
P2	42,94	42,86
P3	101,47	101,24

Los resultados muestran que P3 es considerablemente más costoso en tiempo

de ejecución. Mientras que P1 y P2 se sitúan en torno a 45 y 43 segundos de media respectivamente, P3 alcanza 101 segundos. Este incremento se debe principalmente al aumento del tiempo de generación y a la construcción de un prompt más complejo, con mayor número de tokens aproximados y respuestas de mayor longitud.

En promedio, P3 utiliza 2796 tokens aproximados en el prompt, frente a 2085 en P1 y 2079 en P2. También genera respuestas más largas, con una media de 1863 caracteres frente a 1287 en P1 y 1323 en P2. Estos datos indican que la mejora de calidad de P3 tiene un coste computacional relevante. Por tanto, la elección entre P2 y P3 depende del escenario de uso: P3 es más adecuado cuando se prioriza la calidad, especialmente en generación de código, mientras que P2 puede resultar más eficiente para tareas explicativas o escenarios donde el tiempo de respuesta sea un factor importante.

6.9. Discusión y conclusiones del análisis experimental

Los resultados obtenidos muestran que el pipeline P3 es la configuración más sólida en términos globales. Este pipeline alcanza la mejor puntuación manual media, la mejor media RAGAS y el mayor porcentaje de respuestas de alta calidad. Su principal fortaleza aparece en las tareas de generación de código, donde supera claramente a P1 y P2. Este resultado sugiere que la incorporación de sesiones, memoria, priorización del contexto y prompting específico por tipo de tarea puede mejorar la capacidad del sistema para producir respuestas más completas y útiles en un dominio técnico especializado como Nabsic-Enablon.

Sin embargo, los resultados también indican que una mayor complejidad contextual no siempre implica una mejora uniforme. En otras tareas, P3 no ofrece una mejora sobre P1 y P2 en cuanto a rendimiento. Además, P3 presenta un mayor coste temporal y computacional, por lo que su uso debe valorarse en función del tipo de tarea y de las necesidades concretas del escenario de aplicación.

Esto indica que el enfoque tiene margen de mejora, especialmente en tareas que requieren razonamiento más fino sobre código, validación sintáctica o identificación de errores sutiles. Futuras versiones podrían incorporar ejemplos contrastivos, reglas específicas de validación, comprobaciones automáticas o mecanismos de verificación del código generado.

También se observa que la evaluación automática y la evaluación manual no

siempre coinciden. RAGAS resulta útil para analizar tendencias generales de recuperación, fidelidad y corrección factual, pero presenta limitaciones en tareas de programación. En este contexto, pueden existir varias respuestas técnicamente válidas que no coincidan exactamente con la referencia esperada. Por ello, la rúbrica manual resulta necesaria para valorar aspectos como la utilidad práctica, la corrección técnica, el uso adecuado de objetos Nabsic y la adecuación al estilo del dominio.

En conjunto, los resultados confirman que mejorar un sistema RAG para un dominio técnico no depende únicamente de ampliar el corpus documental. La calidad de la respuesta también está condicionada por la selección del contexto, la forma de presentarlo al modelo, la adaptación del prompt al tipo de tarea y la evaluación con criterios específicos del dominio. P3 representa el enfoque más prometedor del trabajo, pero no puede considerarse una solución óptima para todos los escenarios. Su rendimiento global es superior, aunque su coste y sus debilidades en explicación y corrección muestran que todavía existe margen para optimizar la estrategia de Context Engineering y adaptarla mejor a cada tipo de tarea.

Capítulo 7

Conclusiones y trabajo futuro

7.1. Conclusiones

Este trabajo ha estudiado la aplicación de técnicas de *Retrieval-Augmented Generation* y *Context Engineering* a un dominio técnico especializado: el desarrollo, explicación y corrección de código Nabsic en el entorno Enablon. Para ello, se ha construido un corpus local a partir de documentación oficial, documentación semiestructurada, ejemplos de código y catálogos de módulos; se han implementado tres pipelines RAG comparables; y se ha definido un dataset final de 24 prompts orientados a generación, explicación y corrección de código.

La evaluación final combina métricas automáticas RAGAS con una rúbrica manual específica para el dominio. Esta combinación ha sido necesaria porque RAGAS permite analizar recuperación, fidelidad y corrección factual, pero no captura por completo aspectos propios de la programación en Nabsic, como la plausibilidad sintáctica, el uso correcto de objetos y funciones, la utilidad práctica de una corrección o la adecuación al estilo del lenguaje.

Los resultados muestran que P3 es la configuración con mejor rendimiento global. Este pipeline, basado en Context Engineering con sesiones, memoria, priorización de fuentes y prompting específico por tipo de tarea, obtiene la mejor puntuación manual media y la mejor media RAGAS. Su principal fortaleza aparece en las tareas de generación de código, donde supera claramente a P1 y P2. Esto sugiere que, en un dominio especializado, no basta con recuperar documentos relevantes: también es importante seleccionar, ordenar y presentar el contexto de forma adecuada al modelo generador.

P2 obtiene un comportamiento especialmente competitivo en tareas de explicación, lo que indica que una configuración más simple puede ser suficiente para ciertos

escenarios. En cambio, las tareas de corrección y detección de errores siguen siendo el punto más débil de los tres pipelines. En conjunto, el trabajo confirma que la mejora de un sistema RAG para un dominio técnico no depende únicamente de ampliar el corpus, sino también de cómo se construye y evalúa el contexto.

7.2. Limitaciones

El trabajo presenta varias limitaciones. En primer lugar, el dataset final está compuesto por 24 casos, lo que permite una comparación controlada entre pipelines, pero no conclusiones estadísticas amplias. Además, aunque los casos cubren generación, explicación y corrección de código, no representan toda la variedad de situaciones reales que pueden aparecer en un entorno Enablon.

La evaluación manual también introduce un componente subjetivo. La rúbrica permite valorar aspectos que RAGAS no captura, como la utilidad técnica o la adecuación al estilo Nabsic, pero la puntuación depende del criterio del evaluador y del conocimiento experto disponible. Una evaluación con varios revisores permitiría medir el acuerdo entre evaluadores y reducir esta subjetividad.

Otra limitación procede de las métricas automáticas. RAGAS es útil para comparar recuperación, fidelidad y corrección factual, pero está orientado principalmente a respuestas textuales. En tareas de programación, una respuesta puede ser válida aunque no coincida literalmente con la referencia, o puede parecer cercana a ella y contener errores de sintaxis o de uso de API. Por ello, sus resultados se han interpretado como evidencia complementaria.

El uso de modelos también condiciona los resultados. La generación final se realizó con *HuggingFaceH4/zephyr-7b-beta*, un modelo relativamente limitado frente a modelos comerciales más recientes. No fue posible completar una comparación homogénea con Gemini debido a las cuotas de uso de la modalidad gratuita. Además, la evaluación RAGAS final requirió un modelo de OpenAI, lo que introduce dependencia de un servicio externo y puede dificultar la reproducción exacta del experimento.

El corpus documental puede contener información incompleta, fragmentada o parcialmente desactualizada. Aunque se realizó un proceso de normalización y conversión a Markdown, algunas fuentes pueden introducir ruido o no cubrir todos los objetos, funciones y patrones de uso de Nabsic. Asimismo, la exclusión de ficheros Excel originales reduce ruido, pero puede dejar fuera información tabular potencialmente útil.

Por último, P3 se evaluó con una configuración controlada de memoria para

evitar contaminación entre muestras del benchmark. Esto protege la validez experimental, pero no permite estudiar todo el potencial de una memoria persistente en un uso conversacional real. Tampoco se dispone de un entorno de ejecución automatizado para validar funcionalmente el código generado, por lo que la corrección se ha evaluado principalmente mediante revisión manual.

7.3. Trabajo futuro

Como trabajo futuro, la primera línea natural consiste en ampliar el dataset de evaluación, incorporando más casos, más objetos Nabsic, más módulos Enablon y ejemplos cercanos a incidencias reales. También sería conveniente incluir varios evaluadores humanos para reducir la subjetividad de la rúbrica manual.

Otra línea importante es mejorar la validación del código generado. Si se dispusiera de un entorno de ejecución o validación parcial para Nabsic, podrían añadirse comprobaciones sintácticas, validadores de uso de API o pruebas funcionales. Esto sería especialmente útil para las tareas de generación y corrección de errores.

En relación con el corpus, futuras versiones podrían mejorar la conversión de Excel a Markdown, añadir metadatos más ricos, detectar fuentes obsoletas e incorporar mecanismos de reranking o filtros por tipo documental. Estas mejoras ayudarían a reducir ruido y priorizar documentación técnica más precisa.

También sería interesante estudiar nuevas configuraciones de P3: sin memorias semilla, con solo memoria de sesión, con memoria persistente de aplicación o con políticas de actualización y olvido. Además, se podrían comparar distintos tamaños de chunk, valores de *top-k*, estrategias de priorización y prompts específicos por tarea.

Finalmente, sería útil comparar los pipelines con modelos generadores más recientes y con distintos modelos evaluadores para RAGAS. A medio plazo, el prototipo podría evolucionar desde notebooks experimentales hacia una herramienta más reproducible, como un script, una CLI o una interfaz de consulta integrada con documentación interna de Nabsic-Enablon.

Bibliografía

- [1] S. Zhou, U. Alon, F. F. Xu, Z. Wang, Z. Jiang y G. Neubig, «DocPrompting: Generating Code by Retrieving the Docs,» en *International Conference on Learning Representations*, 2023. DOI: 10.48550/arXiv.2207.05987. dirección: <https://arxiv.org/abs/2207.05987>.
- [2] Z. Z. Wang et al., «CodeRAG-Bench: Can Retrieval Augment Code Generation?» En *Findings of the Association for Computational Linguistics: NAACL 2025*, Albuquerque, New Mexico: Association for Computational Linguistics, 2025, págs. 3199-3214. DOI: 10.18653/v1/2025.findings-naacl.176. dirección: <https://aclanthology.org/2025.findings-naacl.176/>.
- [3] Z. Ji et al., «Survey of Hallucination in Natural Language Generation,» *ACM Computing Surveys*, vol. 55, n.º 12, págs. 1-38, 2023. DOI: 10.1145/3571730. dirección: <https://arxiv.org/abs/2202.03629>.
- [4] S. Lin, J. Hilton y O. Evans, «TruthfulQA: Measuring How Models Mimic Human Falsehoods,» en *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics*, Association for Computational Linguistics, 2022, págs. 3214-3252. DOI: 10.18653/v1/2022.acl-long.229. dirección: <https://aclanthology.org/2022.acl-long.229/>.
- [5] N. Reimers e I. Gurevych, «Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks,» en *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing*, Hong Kong, China: Association for Computational Linguistics, 2019, págs. 3982-3992. DOI: 10.18653/v1/D19-1410. dirección: <https://aclanthology.org/D19-1410/>.
- [6] V. Karpukhin et al., «Dense Passage Retrieval for Open-Domain Question Answering,» en *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing*, Association for Computational Linguistics, 2020, págs. 6769-6781. DOI: 10.48550/arXiv.2004.04906. dirección: <https://arxiv.org/abs/2004.04906>.

- [7] N. Thakur, N. Reimers, A. Rüclé, A. Srivastava e I. Gurevych, «BEIR: A Heterogeneous Benchmark for Zero-Shot Evaluation of Information Retrieval Models,» en *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, 2021. DOI: 10.48550/arXiv.2104.08663. dirección: <https://arxiv.org/abs/2104.08663>.
- [8] N. Muennighoff, N. Tazi, L. Magne y N. Reimers, «MTEB: Massive Text Embedding Benchmark,» en *Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics*, Association for Computational Linguistics, 2023, págs. 2014-2037. DOI: 10.48550/arXiv.2210.07316. dirección: <https://arxiv.org/abs/2210.07316>.
- [9] J. Johnson, M. Douze y H. Jégou, «Billion-Scale Similarity Search with GPUs,» *IEEE Transactions on Big Data*, vol. 7, n.º 3, págs. 535-547, 2019. DOI: 10.1109/TBDATA.2019.2921572. dirección: <https://arxiv.org/abs/1702.08734>.
- [10] P. Lewis et al., «Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks,» en *Advances in Neural Information Processing Systems*, vol. 33, 2020, págs. 9459-9474. DOI: 10.48550/arXiv.2005.11401. dirección: <https://arxiv.org/abs/2005.11401>.
- [11] Y. Gao et al., «Retrieval-Augmented Generation for Large Language Models: A Survey,» *arXiv preprint arXiv:2312.10997*, 2023. DOI: 10.48550/arXiv.2312.10997. dirección: <https://arxiv.org/abs/2312.10997>.
- [12] G. Mialon et al., «Augmented Language Models: A Survey,» *arXiv preprint arXiv:2302.07842*, 2023. DOI: 10.48550/arXiv.2302.07842. dirección: <https://arxiv.org/abs/2302.07842>.
- [13] J. Chen, H. Lin, X. Han y L. Sun, «Benchmarking Large Language Models in Retrieval-Augmented Generation,» *arXiv preprint arXiv:2309.01431*, 2023. DOI: 10.48550/arXiv.2309.01431. dirección: <https://arxiv.org/abs/2309.01431>.
- [14] S. Es, J. James, L. Espinosa-Anke y S. Schockaert, «RAGAS: Automated Evaluation of Retrieval Augmented Generation,» en *Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics: System Demonstrations*, Association for Computational Linguistics, 2024, págs. 150-158. DOI: 10.18653/v1/2024.eacl-demo.16. dirección: <https://aclanthology.org/2024.eacl-demo.16/>.

- [15] N. F. Liu et al., «Lost in the Middle: How Language Models Use Long Contexts,» *Transactions of the Association for Computational Linguistics*, vol. 12, págs. 157-173, 2024. DOI: 10.1162/tac1_a_00638. dirección: <https://arxiv.org/abs/2307.03172>.
- [16] C. Packer et al., «MemGPT: Towards LLMs as Operating Systems,» *arXiv preprint arXiv:2310.08560*, 2023. DOI: 10.48550/arXiv.2310.08560. dirección: <https://arxiv.org/abs/2310.08560>.
- [17] J. S. Park, J. C. O'Brien, C. J. Cai, M. R. Morris, P. Liang y M. S. Bernstein, «Generative Agents: Interactive Simulacra of Human Behavior,» en *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*, Association for Computing Machinery, 2023. DOI: 10.1145/3586183.3606763. dirección: <https://arxiv.org/abs/2304.03442>.
- [18] J. Saad-Falcon, O. Khattab, C. Potts y M. Zaharia, «ARES: An Automated Evaluation Framework for Retrieval-Augmented Generation Systems,» *arXiv preprint arXiv:2311.09476*, 2023. DOI: 10.48550/arXiv.2311.09476. dirección: <https://arxiv.org/abs/2311.09476>.
- [19] M. Chen et al., «Evaluating Large Language Models Trained on Code,» *arXiv preprint arXiv:2107.03374*, 2021. DOI: 10.48550/arXiv.2107.03374. dirección: <https://arxiv.org/abs/2107.03374>.
- [20] J. Austin et al., «Program Synthesis with Large Language Models,» *arXiv preprint arXiv:2108.07732*, 2021. DOI: 10.48550/arXiv.2108.07732. dirección: <https://arxiv.org/abs/2108.07732>.
- [21] S. Lu et al., «CodeXGLUE: A Machine Learning Benchmark Dataset for Code Understanding and Generation,» *arXiv preprint arXiv:2102.04664*, 2021. DOI: 10.48550/arXiv.2102.04664. dirección: <https://arxiv.org/abs/2102.04664>.
- [22] Wolters Kluwer, *Enablon Environmental, Health and Safety Software*, 2026. dirección: <https://www.wolterskluwer.com/en/solutions/enablon>.
- [23] Microsoft, *Visual Basic documentation*, 2026. dirección: <https://learn.microsoft.com/en-us/dotnet/visual-basic/>.
- [24] K. Milam y A. Gulli, *Context Engineering: Sessions, Memory*, Whitepaper, Google/Kaggle whitepaper, 2025.

- [25] A. Asai, Z. Wu, Y. Wang, A. Sil y H. Hajishirzi, «Self-RAG: Learning to Retrieve, Generate, and Critique through Self-Reflection,» en *International Conference on Learning Representations*, 2024. DOI: 10.48550/arXiv.2310.11511. dirección: <https://arxiv.org/abs/2310.11511>.
- [26] S.-Q. Yan, J.-C. Gu, Y. Zhu y Z.-H. Ling, «Corrective Retrieval Augmented Generation,» *arXiv preprint arXiv:2401.15884*, 2024. DOI: 10.48550/arXiv.2401.15884. dirección: <https://arxiv.org/abs/2401.15884>.

Apéndice A

Artefactos del proyecto

Tabla A.1: Artefactos principales del repositorio local.

Artefacto	Descripción
nlp2526_rag_tfm.ipynb	P1 - Notebook exploratorio inicial.
nlp2526-rag-multiformato.ipynb	P2 - Notebook de RAG multiformato para el corpus semiestructurado
nlp2526_rag_context_engineering_sessions_memory.ipynb	P3 - Pipeline RAG que aplica los conceptos de Context Engineering como sesiones, memoria, presupuestos de contexto y trazas.
vectorstores/faiss_nabsic_multiformato/	Índice FAISS local persistido.
eval_outputs/	Outputs de experimentos preliminares.

Apéndice B

Rúbrica manual completa

La evaluación manual se registró en el archivo `nabsic_rag_manual_rubric_3_pipelines.xlsx`. Cada respuesta de cada pipeline se puntuó de forma independiente usando la misma evidencia disponible: `user_input`, `expected_behavior`, `functional_checks`, `reference`, `retrieved_contexts` y `response`. La puntuación total se calcula como:

$$\text{Total} = \text{Corrección técnica y utilidad} + \text{Uso del contexto} + \text{Estilo Nabsic}$$

Corrección técnica y utilidad: generación de código

Tabla B.1: Criterios de corrección técnica y utilidad para tareas de generación.

Subcriterio	Puntos	Qué valorar
Comprende la tarea	1	Resuelve el objetivo pedido, no otra tarea parecida.
Usa estructuras Nabsic adecuadas	1	Emplea objetos, funciones o sentencias apropiadas, como <code>Array</code> , <code>Dictionary</code> , <code>NewRecordSet</code> , <code>If</code> o <code>For</code> .
Sintaxis Nabsic plausible	1	Usa declaraciones, asignaciones, llamadas, separadores <code>:</code> , tipos, <code>Dim</code> , <code>:=</code> y variables con <code>@</code> de forma razonable.

Subcriterio	Puntos	Qué valorar
Corrección funcional principal	2	El código produciría el comportamiento esperado: cálculo, consulta, inserción, validación o salida correcta.
Manejo de casos o riesgos relevantes	1	Evita errores obvios: claves inexistentes, nulos, índices incorrectos, filtros mal formados o confusión de tipos.

Corrección técnica y utilidad: explicación

Tabla B.2: Criterios de corrección técnica y utilidad para tareas de explicación.

Subcriterio	Puntos	Qué valorar
Identifica el propósito general	1	Explica qué hace el fragmento.
Explica elementos Nabsic relevantes	1	Reconoce funciones, objetos, variables, tipos o flujo de ejecución.
Describe el comportamiento paso a paso	2	Explica entradas, proceso y salida esperada.
Detecta implicaciones o riesgos	1	Menciona nulos, índices, tipos, filtros, errores o dependencias si aplica.
Utilidad para un desarrollador	1	La explicación ayuda a modificar, depurar o reutilizar el código.

Corrección técnica y utilidad: corrección o detección de errores

Tabla B.3: Criterios de corrección técnica y utilidad para tareas de corrección o detección de errores.

Subcriterio	Puntos	Qué valorar
Identifica el error principal	2	Localiza el fallo relevante del fragmento.
Explica por qué es un error	1	Justifica el problema con comportamiento Nabsic o documentación recuperada.
Propone una corrección concreta	2	Da código o cambios aplicables.
Mantiene la intención original	1	La solución conserva el objetivo del código inicial.

Uso del contexto

Tabla B.4: Criterios de uso del contexto recuperado.

Puntos	Criterio
0	No usa el contexto recuperado, contradice la documentación o inventa APIs no respaldadas.
1	Usa alguna información contextual, pero de forma débil, incompleta o con errores.
2	Se apoya en fragmentos relevantes y mantiene coherencia general con ellos.
3	Usa claramente los fragmentos recuperados para fundamentar funciones, sintaxis o comportamiento.

Estilo Nabsic

Tabla B.5: Criterios de estilo Nabsic.

Puntos	Criterio
0	La respuesta parece código genérico de otro lenguaje, usa convenciones ajenas o introduce APIs inventadas.
1	La respuesta usa convenciones plausibles del dominio, como variables con @, tipos declarados, Dim, :=, :, Write, DebugPrint, If, For, objetos y métodos Nabsic.

Decisión práctica

Tabla B.6: Interpretación práctica de la puntuación manual total.

Decisión	Puntos	Interpretación
Deficiente	0–3	Respuesta poco útil o incorrecta.
Insuficiente/parcial	4–5	Hay elementos aprovechables, pero falla algo importante.
Aceptable	6–7	Útil con correcciones menores o validación adicional.
Buena	8–9	Correcta, contextualizada y cercana al dominio.
Excelente	10	Correcta, completa, fundamentada y con estilo Nabsic claro.

Apéndice C

Ejemplos del dataset de evaluación

El dataset final de 24 prompts mantendrá el mismo esquema para facilitar la reutilización de los notebooks y la conversión a formato RAGAS. A continuación se muestran dos ejemplos simplificados de casos incluidos en el dataset final. En ambos casos, los campos de salida como `response`, `retrieved_contexts` y `retrieved_sources` aparecen vacíos en el dataset inicial y se completan durante la ejecución de cada pipeline.

Ejemplo 1: generación con Array ■ id: `gen_nabsic_array`

- `task_type`: `generation`
- `benchmark_profile`: `HumanEval/text-to-code`
- `evaluation_scope`: `short_program`
- `oracle_type`: `manual_functional`
- `user_input`: escribir un fragmento Nabsic relacionado con el uso de un `Array`.
- `reference`: la respuesta esperada debe resolver la tarea usando estructuras plausibles de Nabsic.
- `reference_contexts`: documentación de `Objects/Array.txt` y ejemplos relacionados con `Array`.
- `rubric`: debe usar `Array` de forma adecuada, respetar la sintaxis esperada y evitar APIs no documentadas.
- `difficulty`: `low`
- `notes`: caso de generación básica orientado a comprobar el uso de estructuras simples del lenguaje.

Ejemplo 2: generación con Dictionary ■ id: gen_nabsic_dictionary

- task_type: generation
- benchmark_profile: HumanEval/text-to-code
- evaluation_scope: short_program
- oracle_type: manual_functional
- user_input: escribir un fragmento Nabsic relacionado con el uso de un Dictionary.
- reference: la respuesta esperada debe resolver la tarea usando estructuras plausibles de Nabsic.
- reference_contexts: documentación de Objects/Dictionary.txt y ejemplos relacionados con Dictionary.
- rubric: debe usar Dictionary de forma adecuada, respetar la sintaxis esperada y evitar APIs no documentadas.
- difficulty: medium
- notes: caso de generación orientado a comprobar el uso de estructuras clave-valor.

Apéndice D

Plantillas de prompt utilizadas

D.1. Prompt base de P1 y P2

```
SYSTEM:
Using the information contained in the context,
give a comprehensive answer to the question.
Respond only to the question asked, response should be concise and relevant.
If the answer cannot be deduced from the context, do not give an answer.
Answer in Spanish when the user question is in Spanish and do not invent
  Nabsic syntax or APIs.

USER:
Context:
{context}
---
Question: {question}
```

Listing D.1: Estructura simplificada del prompt base usado en P1 y P2.

D.2. Prompt de Context Engineering en P3

```
TASK_INSTRUCTIONS = {
  "generation": (
    "Generate a small Nabsic fragment when possible. Use only APIs,
    operators and patterns supported by the retrieved documents or memory. "
    "Then explain briefly why the code satisfies the request."
  ),
```

```

    "explanation": (
        "Explain the code or concept step by step. Compute the expected
        result if the provided evidence is sufficient."
    ),
    "correction": (
        "Identify the main error or risk, explain why it happens and propose
        a concrete Nabsic correction."
    ),
    "error_detection": (
        "Identify the main error or risk, explain why it happens and propose
        a concrete Nabsic correction."
    ),
    "general": (
        "Answer concisely using only evidence available in the provided
        context."
    ),
}

CE_PROMPT_TEMPLATE = """You are a technical assistant specialized in Nabsic
    and Enablon.

Global rules:
- Answer in Spanish.
- Use only the provided session, memory and document context.
- If the context is insufficient, say so explicitly.
- Prefer official manuals, detailed Nabsic docs and object docs over broad
    Enablon table catalogs.
- If you generate code, keep it small and explain the relevant calls.
- Do not mention internal scoring unless the user asks for experiment traces
    .

Task type: {task_type}
Task instruction: {task_instruction}

Session summary:
{session_summary}

Relevant memory:
{memory_context}

```

```
Retrieved document evidence:
```

```
{document_context}
```

```
User question:
```

```
{question}
```

```
Answer: ""
```

Listing D.2: Estructura de la estrategia de prompting usada en P3

Apéndice E

Memorias semilla del pipeline P3

El pipeline P3 inicializa un conjunto reducido de memorias semilla con alcance de aplicación. Estas memorias no se generan a partir de las referencias esperadas del dataset, sino que recogen reglas generales de comportamiento, criterios de uso del contexto y decisiones de diseño del TFM. Su objetivo es orientar la construcción del prompt sin introducir conocimiento específico de los casos de evaluación.

Cada memoria se representa como un registro con identificador, alcance, tipo, contenido, etiquetas, importancia, fuente y procedencia. En la configuración utilizada, todas las memorias semilla tienen alcance `application` y una procedencia marcada como `uses_dataset_reference=false`. La Tabla E.1 resume las memorias semilla definidas.

Tabla E.1: Memorias semilla definidas en el pipeline P3.

ID	Tipo	Imp	Contenido	Fuente
seed_001	Procedural	0.95	Para tareas de generación de código Nabsic, priorizar APIs y operadores que aparezcan en el contexto recuperado. Si falta evidencia, indicar la limitación en lugar de inventar funciones.	Regla de diseño del TFM
seed_002	Procedural	0.85	Para tareas de explicación de código Nabsic, explicar las operaciones en orden de ejecución y calcular el resultado observable final cuando el contexto recuperado lo permita.	Diseño de tarea del TFM

ID	Tipo	Imp.	Contenido	Fuente
seed_ 003	Procedural	0.90	Para explicación de código Nabsic, no inventar relaciones causales, efectos secundarios ni comportamiento interno de funciones personalizadas que no estén documentadas en el contexto. Explicar solo lo observable desde el código proporcionado.	Diseño de tarea del TFM
seed_ 004	Procedural	0.85	Para detección de errores en Nabsic, identificar el error o riesgo concreto, explicar por qué ocurre y proponer un fragmento corregido pequeño.	Diseño de tarea del TFM
seed_ 005	Declarative	0.80	Las respuestas RAG sobre Nabsic deben priorizar manuales oficiales y documentación de objetos frente a catálogos amplios de tablas Enablon cuando ambos tipos de fuente sean recuperados.	Decisión de corpus del TFM
seed_ 006	Procedural	0.90	Para tareas de generación, priorizar ejemplos didácticos recuperados de <code>Ejemplos/nabsic_basico_*.md</code> cuando coincidan con la API, tabla o patrón solicitado. Usar catálogos amplios de módulos Enablon principalmente para confirmar rutas de tablas, nombres XML de campos, tipos, semántica de enlaces y valores CCL, sin dejar que desplacen ejemplos validados.	Decisión de corpus tras validación manual

Apéndice F

Repositorio del proyecto

El código fuente, los notebooks de los tres pipelines, los resultados experimentales, las evaluaciones y las figuras generadas durante el proyecto se encuentran disponibles en el siguiente repositorio:

<https://github.com/JulioCrbe/TFM-Final>