uc3m | Universidad Carlos III de Madrid

Grado en Ingeniería informática
2025-2026

Trabajo Fin de Grado

Exploiting Retrieval Latency Side-Channels in RAG Systems.

Sergio Verde Llorente
Tutor/es
José María de Fuentes
Lugar y fecha de presentación prevista

[Incluir en el caso del interés de su publicación en el archivo abierto]
Esta obra se encuentra sujeta a la licencia Creative Commons **Reconocimiento - No Comercial - Sin Obra Derivada**

AGRADECIMIENTOS

Quiero agradecer especialmente a mi tutor, José María de Fuentes, por todo el apoyo y la confianza que me ha dado durante la realización de este Trabajo de Fin de Grado.

Gracias por darme la oportunidad de trabajar en un tema tan interesante, por ayudarme a llevar esta investigación más allá de una simple memoria académica y por abrirme la posibilidad de participar en la preparación de un paper sobre el trabajo desarrollado. Este proyecto me ha permitido entrar en el mundo de la ciberseguridad desde una perspectiva que no conocía, mucho más cercana a la investigación y al análisis de vulnerabilidades. También quiero agradecerle la forma en la que ha guiado el trabajo, ya que me ha dejado libertad para investigar, probar ideas y equivocarme, pero siempre orientando el camino cuando era necesario. Esa combinación entre autonomía y guía ha sido clave para que el proyecto pudiera crecer y para que yo pudiera entender mejor qué significa investigar un problema técnico desde el principio.

Por último, gracias por el apoyo personal y académico durante todo este proceso, y por ayudarme a recuperar la motivación en un momento en el que la necesitaba. Este trabajo no solo me ha servido para cerrar una etapa universitaria, sino también para descubrir un área en la que quiero seguir aprendiendo y creciendo.

RESUMEN

Los sistemas de Generación Aumentada por Recuperación, conocidos como RAG, se utilizan cada vez más en entornos empresariales para mejorar la precisión de los modelos de lenguaje mediante el uso de documentos privados como contexto. Sin embargo, esta arquitectura introduce nuevos riesgos de seguridad, ya que el comportamiento interno del sistema puede revelar información aunque el contenido de los documentos no se muestre directamente al usuario. Este Trabajo de Fin de Grado estudia la existencia de un canal lateral temporal en sistemas RAG basado en la latencia de recuperación vectorial. La hipótesis principal es que las consultas relacionadas con información presente en la base de datos vectorial generan tiempos de respuesta distintos a las consultas asociadas a información ausente o poco relacionada con el corpus. Para analizar esta hipótesis, se ha realizado una experimentación progresiva que comienza con la recuperación vectorial aislada, continúa con la integración de un pipeline RAG completo con modelo de lenguaje y finaliza con pruebas sobre mecanismos de caché.

A partir de los resultados obtenidos, se ha desarrollado una herramienta orientada a la auditoría defensiva de sistemas RAG externos. La herramienta permite configurar targets, cargar datasets de consultas, ejecutar auditorías, medir el Time-to-First-Byte, analizar diferencias entre grupos de frecuencia y longitud, y realizar inferencias de términos a partir de auditorías previas.

Los resultados muestran que la señal temporal existe y puede medirse en distintos escenarios, aunque su claridad depende del ruido del entorno, del modelo utilizado, del valor de k, de la estabilidad del sistema auditado y de la presencia de cachés. El trabajo concluye que la seguridad de los sistemas RAG no debe limitarse al contenido generado por el modelo, sino que también debe considerar señales observables desde el exterior, como la latencia.

Palabras clave: RAG, recuperación vectorial, canal lateral temporal, Time-to-First-Byte, auditoría de caja negra, inferencia de términos, HNSW, caché semántica.

ABSTRACT

Retrieval-Augmented Generation systems, commonly known as RAG, are increasingly used in enterprise environments to improve the accuracy of language models by using private documents as contextual information. However, this architecture introduces new security risks, since the internal behaviour of the system may reveal information even when the content of the documents is not directly exposed to the user. This Bachelor's Thesis studies the existence of a timing side channel in RAG systems based on vector retrieval latency. The main hypothesis is that queries related to information present in the vector database produce different response times from queries associated with absent information or content weakly related to the corpus. To analyse this hypothesis, a progressive experimental process has been carried out, starting with isolated vector retrieval, continuing with a complete RAG pipeline including a language model, and finally evaluating the effect of cache mechanisms.

Based on the results obtained, a tool has been developed for defensive auditing of external RAG systems. The tool allows the auditor to configure targets, upload query datasets, execute audits, measure Time-to-First-Byte, analyse differences between frequency and length groups, and perform term inference using previous audits as calibration.

The results show that the timing signal exists and can be measured in different scenarios, although its clarity depends on environmental noise, the model used, the value of k, the stability of the audited system and the presence of caches. The work concludes that RAG security should not be limited to the text generated by the model, but should also consider externally observable signals such as latency.

Keywords: RAG, vector retrieval, timing side channel, Time-to-First-Byte, black-box auditing, term inference, HNSW, semantic cache.

ÍNDICE

ÍNDICE DE FIGURAS ..... **X**
ÍNDICE DE TABLAS ..... **XII**
LISTA DE ACRÓNIMOS Y TÉRMINOS ..... **XIII**
1. INTRODUCCIÓN **1**
1.1. CONTEXTO Y MOTIVACIÓN 1
1.2. PROBLEMA ABORDADO 3
1.3. OBJETIVOS DEL TRABAJO 4
1.4. ALCANCE DEL TRABAJO 5
1.5. ENFOQUE METODOLÓGICO SEGUIDO 6
1.6. ESTRUCTURA DE LA MEMORIA 7
2. ESTADO DEL ARTE **8**
2.1. PRIVACIDAD Y SEGURIDAD EN SISTEMAS RAG 8
2.1.1. Inferencia de Membresía en la Recuperación Vectorial 8
2.1.2. Explotación del Presupuesto de Tokens 10
2.2. ATAQUES DE CANAL LATERAL EN MODELOS DE LENGUAJE 10
2.2.1. Fugas de Información en la Inferencia Autoregresiva 11
2.2.2. Impacto Comercial y Estrategias de Mitigación 12
2.3. VULNERABILIDADES EN MECANISMOS DE CACHÉ 13
2.3.1. Jerarquía de Memoria: KV-Cache y Prompt Caching 13
2.3.2. Ataques en Cachés Semánticas 14
2.3.3. Fugas en KV-Cache Multi-inquilino 14
2.4. COMPARATIVA DE TRABAJOS RELACIONADOS Y BRECHA DE INVESTIGACIÓN 15
2.4.1. Taxonomía de las Metodologías de Exfiltración 15
2.4.2. Comparativa de Trabajos Relacionados 16
2.4.3. Lagunas en la Literatura Actual 17
2.4.4. Definición de la Brecha de Investigación 17
3. ANÁLISIS DE REQUISITOS **18**
3.1. DEFINICIÓN Y CLASIFICACIÓN DE REQUISITOS 19
3.2. MODELO DE CASOS DE USO 22
3.2.1. Actores 22
3.2.2. Diagrama de casos de uso 22
3.2.3. Descripción de alto nivel de los casos de uso 23
3.3. MATRIZ DE TRAZABILIDAD 24
4. DISEÑO Y METODOLOGÍA DEL SISTEMA **26**
4.1. OBJETIVO DE DISEÑO DE LA HERRAMIENTA 26
4.2. ARQUITECTURA GENERAL DE LA SOLUCIÓN 27

4.3. DISEÑO FUNCIONAL DE LA HERRAMIENTA 28
4.3.1. Ejecución de auditorías 30
4.3.2. Ejecución de inferencia de términos 31
4.4. DISEÑO DEL SISTEMA DE MEDICIÓN Y ANÁLISIS 32
4.4.1. Medición temporal y procesamiento estadístico 32
4.4.2. Análisis para inferencia de términos 33
4.5. DISEÑO DE PERSISTENCIA Y RESULTADOS 34
4.6. DISEÑO DE LA INTERFAZ DE USUARIO 37
4.7. DECISIONES DE DISEÑO Y LIMITACIONES 39
5. IMPLEMENTACIÓN DE LA HERRAMIENTA 40
5.1. ENTORNO TECNOLÓGICO UTILIZADO 40
5.2. ORGANIZACIÓN GENERAL DEL PROYECTO 42
5.3. IMPLEMENTACIÓN DEL BACKEND 44
5.3.1. API y servicios de gestión 45
5.3.2. Seguridad y autenticación 46
5.4. IMPLEMENTACIÓN DEL PROCESAMIENTO ASÍNCRONO 47
5.4.1. Encolado de trabajos 47
5.4.2. Flujo de estados y tratamiento de errores 48
5.5. IMPLEMENTACIÓN DEL MÓDULO DE MEDICIÓN TEMPORAL 50
5.5.1. Preparación de la petición HTTP 50
5.5.2. Medición del Time-to-First-Byte y del tiempo completo 51
5.5.3. Registro de mediciones y errores 52
5.6. IMPLEMENTACIÓN DEL ANÁLISIS ESTADÍSTICO 53
5.6.1. Análisis estadístico de auditorías 53
5.6.2. Análisis estadístico de términos inferidos 54
5.6.3. Exportación de resultados 56
5.7. IMPLEMENTACIÓN DE LA INTERFAZ DE USUARIO 56
5.8. EJECUCIÓN DE LA HERRAMIENTA 58
6. PRUEBAS Y EVALUACIÓN EXPERIMENTAL 60
6.1. DISEÑO EXPERIMENTAL PREVIO 60
6.1.1. Hipótesis y modelo de amenaza 60
6.1.2. Identificación y control de variables 62
6.1.3. Diseño del corpus y banco de consultas 63
6.1.4. Protocolo de ejecución y medición 65
6.2. EJECUCIÓN DE LA INVESTIGACIÓN 66
6.2.1. Validación local inicial con Project Gutenberg 66
6.2.2. Escalado del corpus en RunPod 67
6.2.3. Caracterización del corpus y construcción del banco de consultas 67

6.2.4. Ejecución del experimento de recuperación vectorial 68
6.2.5. Integración del pipeline RAG con modelo de lenguaje 68
6.2.6. Ejecución del experimento con caché 69
6.3. CRITERIOS DE ANÁLISIS DE RESULTADOS 70
6.4. RESULTADOS EXPERIMENTALES Y DISCUSIÓN 71
6.4.1. Resultados en recuperación vectorial aislada 71
6.4.2. Resultados en pipeline RAG con modelo de lenguaje 77
6.4.3. Resultados con mecanismos de caché 81
6.4.4. Comparación global entre escenarios 83
6.5. VALIDACIÓN DE LA HERRAMIENTA DESARROLLADA 84
6.5.1. Entorno RAG de validación en RunPod 85
6.5.2. Validación de los flujos principales de la herramienta 86
6.5.3. Evaluación de resultados de la herramienta 89
6.5.4. Validación del alcance definido por los requisitos 97
6.6. LIMITACIONES OBSERVADAS 98
7. ASPECTOS ECONÓMICOS Y SOCIALES 99
7.1. PRESUPUESTO DEL PROYECTO 99
7.2. ANÁLISIS DE IMPACTO SOCIAL Y ÉTICO 102
8. CONCLUSIONES Y TRABAJO FUTURO 103
8.1. CONCLUSIONES Y CONSECUCIÓN DE OBJETIVOS 104
8.2. LIMITACIONES DEL TRABAJO DESARROLLADO 106
8.3. LÍNEAS DE TRABAJO FUTURAS 107
9. BIBLIOGRAFÍA 110
ANEXO A. VISUALIZACIÓN DE LA INTERFAZ DE USUARIO 115
A.1. PANTALLA DE AUTENTICACIÓN 115
A.2. PANEL PRINCIPAL DE AUDITORÍAS 115
A.3. GESTIÓN DE TARGETS 116
A.4. CARGA Y VALIDACIÓN DE DATASETS 117
A.5. CREACIÓN DE UNA AUDITORÍA 118
A.6. RESULTADOS DE AUDITORÍA 118
A.7. PANEL DE INFERENCIA DE TÉRMINOS 120
A.8. CREACIÓN DE INFERENCIA DE TÉRMINOS 120
A.9. RESULTADOS DE INFERENCIA DE TÉRMINOS 122
A.10. CONFIRMACIÓN DE ACCIONES DESTRUCTIVAS 123
ANEXO B. MANUAL DE INSTALACIÓN Y EJECUCIÓN 125
B.1. REQUISITOS PREVIOS 125
B.2. OBTENCIÓN DEL CÓDIGO FUENTE 125
B.3. CONFIGURACIÓN DEL ENTORNO 125

B.4. ARRANQUE DE LA HERRAMIENTA 126
B.5. ACCESO A LA INTERFAZ WEB 126
B.6. PARADA Y REINICIO DEL ENTORNO 126
B.7. PROBLEMAS FRECUENTES 127
ANEXO C. FORMATOS DE ENTRADA Y SALIDA 128
C.1. CONFIGURACIÓN DE UN TARGET 128
C.2. FORMATO DEL DATASET DE AUDITORÍA 129
C.3. ENTRADA PARA INFERENCIA DE TÉRMINOS 131
C.4. SALIDA DE RESULTADOS DE AUDITORÍA 132
C.5. SALIDA DE RESULTADOS DE INFERENCIA DE TÉRMINOS 133
ANEXO D. DECLARACIÓN DE USO DE INTELIGENCIA ARTIFICIAL GENERATIVA 135

ÍNDICE DE FIGURAS

Fig. 1. Diagrama de casos de uso de la herramienta. 22
Fig. 2. Arquitectura general de la herramienta de auditoría. 28
Fig. 3. Flujo funcional de configuración de targets y datasets. 29
Fig. 4. Flujo funcional de ejecución de una auditoría. 31
Fig. 5. Flujo funcional de inferencia de términos. 32
Fig. 6. Modelo de datos de la herramienta de auditoría. 36
Fig. 7. Estructura de carpetas del desarrollo de la herramienta. 44
Fig. 8. Puntos de medición del Time-to-First-Byte y del tiempo completo. 52
Fig. 9. Comparación de latencias medias en recuperación vectorial aislada en ejecución local simplificada. 72
Fig. 10. Comparación de latencias medias en recuperación vectorial aislada en ejecución completa en RunPod. 73
Fig. 11. Mapas de calor de latencias medias en recuperación vectorial aislada con k=5, 50, 100 y 200. 75
Fig. 12. Diagramas de cajas de las latencias en recuperación vectorial aislada con k=5 y 200. 76
Fig. 13. Densidades superpuestas de latencia para alta, media y baja frecuencia en recuperación vectorial aislada con k=200. 77
Fig. 14. Comparación de latencias medias de la primera ejecución en pipeline RAG completo usando Qwen2.5-7B-Instruct. 78
Fig. 15. Comparación de latencias medias de la segunda ejecución en pipeline RAG completo usando Qwen2.5-7B-Instruct. 78
Fig. 16. Comparación de latencias medias en pipeline RAG completo usando SmolLM2-135M-Instruct. 79
Fig. 17. Mapa de calor de latencias medias en pipeline RAG con SmolLM2-135M-Instruct y k=50. 80
Fig. 18. Diagramas de cajas de latencias en pipeline RAG con SmolLM2-135M-Instruct y k=25 y 50. 80
Fig. 19. Comparación de latencias del pipeline RAG sin caché durante el experimento con mecanismos de caché. 81
Fig. 20. Comparación de latencias del pipeline RAG con caché exacta durante el experimento con mecanismos de caché. 82
Fig. 21. Comparación de latencias del pipeline RAG con caché semántica y umbral 0.85 durante el experimento con mecanismos de caché. 83
Fig. 22. Resultados gráficos de la primera auditoría ejecutada desde la herramienta. 90
Fig. 23. Comparaciones estadísticas por frecuencia en la primera auditoría. 90
Fig. 24. Resultados de la primera auditoría con eje vertical iniciado en cero. 91
Fig. 25. Petición fallida registrada durante la segunda auditoría. 91
Fig. 26. Comparación de latencias medias en la segunda auditoría. 92
Fig. 27. Mapa de calor de latencias medias en la segunda auditoría. 93
Fig. 28. Comparaciones estadísticas por frecuencia en la segunda auditoría. 93
Fig. 29. Comparación de latencias medias en la tercera auditoría. 94
Fig. 30. Comparaciones estadísticas por frecuencia en la tercera auditoría. 94
Fig. 31. Clasificación de términos en una ejecución estable de inferencia. 95
Fig. 32. Resultados tabulados de una ejecución estable de inferencia. 95
Fig. 33. Clasificación de términos en una ejecución de inferencia afectada por ruido. 96
Fig. 34. Resultados tabulados de una ejecución de inferencia afectada por ruido. 96
Fig. 35. Pantalla de autenticación 115

Fig. 36. Panel principal de auditorías. 116
Fig. 37. Pantalla de gestión y creación de targets. 116
Fig. 38. Formulario de edición de un target existente. 117
Fig. 39. Pantalla de carga de datasets. 117
Fig. 40. Vista previa y estadísticas de un dataset cargado. 118
Fig. 41. Formulario de creación de una auditoría. 118
Fig. 42. Vista general de resultados de una auditoría. 119
Fig. 43. Comparaciones estadísticas por frecuencia en una auditoría. 119
Fig. 44. Tabla de mediciones individuales de una auditoría. 120
Fig. 45. Panel principal de inferencias de términos. 120
Fig. 46. Formulario de creación de inferencia de términos con auditoría previa y términos individuales. 121
Fig. 47. Formulario de creación de inferencia de términos con calibración mediante CSV y modo de consultas. 122
Fig. 48. Clasificación de términos frente a las fronteras de calibración. 122
Fig. 49. Evolución de la clasificación por rondas durante una inferencia. 123
Fig. 50. Tabla de resultados de una inferencia de términos. 123
Fig. 51. Mediciones individuales registradas durante una inferencia de términos. 123
Fig. 52. Panel de confirmación para acciones de eliminación. 124

ÍNDICE DE TABLAS

Tabla I. COMPARATIVA DE TRABAJOS RELACIONADOS. 16
Tabla II. REQUISITOS FUNCIONALES Y NO FUNCIONALES DE LA HERRAMIENTA. 20
Tabla III. DESCRIPCIÓN DE ALTO NIVEL DE LOS CASOS DE USO. 23
Tabla IV. MATRIZ DE TRAZABILIDAD 24
Tabla V. ESTIMACIÓN DEL COSTE HUMANO DEL PROYECTO. 100
Tabla VI. ESTIMACIÓN DE COSTES DE INFRAESTRUCTURA Y OTROS RECURSOS. 101
Tabla VII. RESUMEN DEL PRESUPUESTO TOTAL DEL PROYECTO. 101
Tabla VIII. GRUPOS DE CAMPOS INCLUIDOS EN LA EXPORTACIÓN CSV DE AUDITORÍAS. 132
Tabla IX. GRUPOS DE CAMPOS INCLUIDOS EN EL CSV DE INFERENCIA DE TÉRMINOS. 133

LISTA DE ACRÓNIMOS Y TÉRMINOS

Término	Significado
RAG	Retrieval-Augmented Generation / Generación Aumentada por Recuperación.
LLM	Large Language Model / Modelo de Lenguaje de Gran Escala.
MIA	Membership Inference Attack / Ataque de Inferencia de Membresía.
ANN	Approximate Nearest Neighbors / búsqueda aproximada de vecinos más cercanos.
HNSW	Hierarchical Navigable Small World, algoritmo de búsqueda aproximada usado en bases de datos vectoriales.
TTFB	Time-to-First-Byte, tiempo hasta la recepción del primer byte de respuesta.
KV-Cache	Caché de claves y valores usada durante la inferencia de un modelo de lenguaje.
Prompt Caching	Técnica de reutilización de prompts o prefijos ya procesados para reducir latencia.
Caché semántica	Mecanismo de caché que reutiliza respuestas cuando dos consultas son suficientemente similares en el espacio vectorial.
Embedding	Representación vectorial de un texto.
Retriever	Componente encargado de recuperar documentos o fragmentos relevantes desde la base vectorial.
Pipeline RAG	Secuencia completa formada por consulta, recuperación de contexto y generación de respuesta.
Dataset	Conjunto de consultas o datos usado durante las pruebas.
Query	Consulta enviada al sistema RAG durante una auditoría o inferencia.
Target	Sistema RAG externo configurado como objetivo de una auditoría.
Worker	Proceso en segundo plano encargado de ejecutar auditorías e inferencias.
API	Application Programming Interface, interfaz usada para comunicar sistemas mediante llamadas definidas.
HTTP	Protocolo de comunicación usado para enviar las peticiones al sistema auditado.
JSON	Formato estructurado usado para configurar targets, datasets y cuerpos de petición.
CSV	Formato de archivo usado para exportar resultados y cargar calibraciones externas.
TLS	Protocolo de seguridad usado para proteger comunicaciones HTTPS.
p-value	Valor estadístico usado para analizar si una diferencia observada entre grupos puede deberse al ruido.
k	Número de vecinos recuperados por el sistema de recuperación vectorial.

1. INTRODUCCIÓN

Los sistemas basados en inteligencia artificial generativa han pasado en muy poco tiempo de ser herramientas experimentales a formar parte de procesos reales dentro de empresas y entornos técnicos [1]. Su capacidad para responder preguntas, resumir documentos, ayudar en tareas de análisis o asistir en procesos de soporte ha hecho que cada vez sea más habitual integrarlos con información privada de una organización. En este contexto aparecen las arquitecturas de Generación Aumentada por Recuperación, conocidas como RAG, que permiten combinar un modelo de lenguaje con una base documental externa [2], [3]. La idea principal de un sistema RAG es que, en lugar de pedir al modelo que responda únicamente con el conocimiento aprendido durante su entrenamiento, el sistema recupera documentos o fragmentos relevantes desde una base de datos y los incorpora como contexto para generar una respuesta más precisa [2], [3]. Esto resulta especialmente útil en entornos corporativos, donde se necesita ser muy preciso con la información buscada, que a la vez cambia con frecuencia y no siempre está disponible en los datos originales con los que fue entrenado el modelo. Gracias a este enfoque, una empresa puede construir asistentes internos capaces de responder sobre documentación técnica, procedimientos, contratos, políticas internas o bases de conocimiento propias.

Sin embargo, esta mejora funcional también introduce nuevos riesgos, ya que al conectar un modelo de lenguaje con documentos privados, la seguridad del sistema deja de depender únicamente del comportamiento del modelo y pasa a depender también del proceso de recuperación, de la base vectorial, de los mecanismos de caché, de la infraestructura y de las señales externas que el sistema puede emitir mientras responde. Aunque el contenido de los documentos no se muestre directamente al usuario, el comportamiento observable del sistema puede revelar información sobre lo que está almacenado en su interior [4], [5]. Este Trabajo de Fin de Grado se sitúa precisamente en ese punto. El objetivo es estudiar si la latencia de respuesta de un sistema RAG puede actuar como un canal lateral temporal capaz de filtrar información sobre la presencia o ausencia de determinados datos en una base de conocimiento [4], [5]. La hipótesis de partida es que no todas las consultas tienen por qué generar el mismo comportamiento temporal, ya que una consulta relacionada con información muy presente en el corpus puede recorrer el sistema de forma distinta a otra consulta asociada a información ausente o poco relacionada con los documentos almacenados.

A partir de esta idea, el trabajo combina dos partes. Por un lado, una investigación experimental orientada a comprobar si existe una diferencia temporal medible en distintos escenarios RAG. Por otro lado, el desarrollo de una herramienta software que permite automatizar auditorías de latencia contra sistemas RAG externos desde una perspectiva defensiva. El interés del trabajo reside no solo en demostrar una diferencia de tiempos en un entorno controlado, sino también en estudiar si esa diferencia puede convertirse en una metodología de auditoría. Por este motivo, la memoria, además de presentar resultados experimentales, también define requisitos, diseña una arquitectura software, implementa una herramienta y valida su funcionamiento mediante pruebas sobre un entorno RAG preparado para este fin.

1.1. CONTEXTO Y MOTIVACIÓN

El uso de modelos de lenguaje en entornos empresariales ha cambiado la forma en la que muchas organizaciones acceden a su información interna. Antes, consultar documentación técnica o buscar un procedimiento concreto solía depender de motores de búsqueda

tradicionales o de la navegación manual por carpetas, pero con la aparición de los sistemas RAG, esa interacción se vuelve mucho más directa, ya que el usuario puede hacer una pregunta en lenguaje natural y recibir una respuesta construida a partir de documentos internos [2], [3]. Esta capacidad tiene un valor empresarial evidente, ya que un sistema RAG puede reducir tiempos de búsqueda, mejorar la disponibilidad del conocimiento y ayudar a que información dispersa sea más fácil de consultar. También permite que los modelos de lenguaje trabajen con información actualizada sin tener que reentrenarlos cada vez que cambia un documento [2], [3], lo que ha hecho que este tipo de arquitectura se haya convertido en una de las formas más utilizadas para llevar la inteligencia artificial generativa a casos de uso reales dentro de las empresas.

En muchos casos, la base documental de un RAG puede contener datos internos, información estratégica, documentación financiera, procedimientos técnicos, contratos, datos de empleados o conocimiento que no debería ser accesible para todos los usuarios, haciendo que la arquitectura se conecte con información sensible. Aunque el sistema tenga controles para impedir que el modelo muestre información no autorizada, sigue existiendo la posibilidad de que otros elementos del funcionamiento interno filtren señales indirectas. En seguridad informática, este tipo de problemas se conocen como canales laterales. Un canal lateral es una señal medible producida por el sistema mientras funciona, por lo que los ataques que los explotan no necesitan encontrar un fallo directo en el código o un mecanismo criptográfico erróneo. Entre estas señales se encuentran el consumo de recursos, los tamaños de paquete, los tiempos de respuesta o las fugas electromagnéticas [4], [5]. Estas señales pueden parecer inofensivas, pero en determinados escenarios permiten inferir información que no debería estar disponible. En el caso de los sistemas RAG, la latencia es una señal especialmente interesante porque toda consulta tiene que atravesar varias fases antes de producir una respuesta. Primero se procesa la pregunta del usuario, después se transforma en una representación vectorial, posteriormente se recuperan documentos relacionados y finalmente se genera la respuesta del modelo [2], [3], [6]. Cada una de estas fases puede introducir variaciones temporales que, si dependen de la información almacenada en el sistema, pueden convertirse en una fuente de fuga de información.

Muchas investigaciones sobre privacidad en modelos de lenguaje se han centrado en analizar el texto generado, las probabilidades internas del modelo, los límites de tokens o el comportamiento de mecanismos de caché [7]-[15]. Sin embargo, el proceso de recuperación vectorial de un RAG también puede tener un comportamiento temporal propio [6]. Este componente es especialmente relevante porque es el encargado de buscar en la base documental privada y decidir qué fragmentos se entregan al modelo como contexto. Por tanto, la latencia de recuperación puede ser un problema de seguridad, ya que si un atacante autorizado a interactuar con un chatbot corporativo pudiera enviar consultas repetidas y medir sus tiempos de respuesta, podría llegar a inferir si ciertos términos o documentos presentes en la base vectorial. Esto tendría impacto en escenarios donde la simple existencia de un documento ya es información confidencial.

La motivación principal del proyecto es analizar este riesgo desde una perspectiva defensiva. La finalidad es estudiar una posible vulnerabilidad para que pueda ser entendida, medida y auditada en entornos controlados. En este sentido, el trabajo busca aportar una metodología práctica para evaluar si una arquitectura RAG presenta diferencias temporales explotables y, a partir de ahí, ayudar a mejorar su seguridad. Esta motivación también justifica el desarrollo de una herramienta software, ya que medir latencias de forma manual mediante scripts puede servir para una prueba inicial, pero no es suficiente para plantear un proceso de auditoría

repetible. Una herramienta permite configurar sistemas objetivo, lanzar pruebas, registrar resultados y visualizar diferencias estadísticas de forma más ordenada. Así, la investigación se traslada a un entorno más cercano al uso real por parte de un auditor técnico o de un equipo de seguridad.

1.2. PROBLEMA ABORDADO

El problema abordado en este Trabajo de Fin de Grado parte de una hipótesis que todavía no había sido comprobada en el contexto concreto de esta investigación: que el proceso de recuperación vectorial de un sistema RAG podría presentar diferencias temporales dependiendo de la relación entre la consulta enviada y el contenido almacenado en la base de conocimiento. En este tipo de arquitecturas, una consulta del usuario no se compara directamente con los documentos en formato textual, sino que primero se transforma en una representación vectorial y después se ejecuta un algoritmo de búsqueda encargado de localizar los fragmentos más cercanos dentro del espacio vectorial [2], [3], [6]. La cuestión que se plantea es si ese algoritmo puede comportarse de forma distinta cuando la consulta cae en una zona muy relacionada con información indexada frente a cuando se sitúa en una zona poco representada o ajena al corpus. Las bases de datos vectoriales suelen emplear algoritmos específicos de búsqueda aproximada de vecinos más cercanos, diseñados para encontrar resultados relevantes con baja latencia incluso cuando el corpus contiene una gran cantidad de documentos [6]. Estos algoritmos no recorren necesariamente el índice de la misma forma para todas las consultas. Su comportamiento puede variar dependiendo de la posición del vector consultado dentro del espacio, de la densidad de la zona en la que cae, de la similitud con los vectores ya indexados y del número de vecinos que se solicitan como resultado [6].

A partir de esta idea surge el problema central del trabajo: si el comportamiento temporal del algoritmo de recuperación cambia según la relación entre el contenido de la consulta y el contenido almacenado en la base vectorial, entonces un usuario podría llegar a inferir información sobre esa base de datos sin verla directamente. En ese escenario, la latencia de recuperación deja de ser solo una métrica de rendimiento y pasa a convertirse en una señal observable que podría revelar si una consulta está relacionada con información presente, frecuente o ausente en el sistema. En sistemas RAG corporativos, esta diferencia es especialmente relevante porque suelen estar construidos sobre documentación privada, como procedimientos internos, contratos, información técnica, datos financieros o documentos de recursos humanos. Si el proceso de recuperación se comporta de forma distinta cuando una consulta se aproxima a información existente en el índice y esa diferencia temporal se repite de forma consistente, podría aparecer un canal lateral capaz de filtrar información sobre la base documental.

Por tanto, el problema se encuentra en una fase anterior a la generación del modelo que es el momento en el que el sistema decide qué fragmentos recuperar. Si una consulta muy relacionada con documentos indexados produce un patrón temporal distinto al de una consulta alejada del corpus, el tiempo de respuesta puede actuar como una evidencia indirecta de presencia. Esta idea conecta el funcionamiento interno del algoritmo de recuperación con un posible riesgo de privacidad observable desde el exterior.

La dificultad de estudiar este problema está en que la latencia de un sistema RAG contiene ruido. En una respuesta completa intervienen varias fases, como la vectorización de la consulta, la búsqueda en la base vectorial, la construcción del contexto, la generación del modelo de lenguaje, la comunicación de red, la carga del servidor y los posibles mecanismos de caché.

Por ello, para medir esta diferencia es necesario repetir mediciones, comparar grupos, controlar variables y analizar si las diferencias observadas tienen suficiente consistencia estadística.

Así, el trabajo aborda el problema en dos niveles. Primero, desde una perspectiva experimental, se estudia si existe una diferencia temporal medible asociada al proceso de recuperación vectorial y si esa señal se mantiene cuando se añaden componentes más realistas del pipeline RAG. Después, desde una perspectiva de ingeniería, se desarrolla una herramienta que permite automatizar este análisis mediante auditorías de caja negra, enviando consultas a un sistema RAG externo y midiendo los tiempos observables desde el exterior.

1.3. OBJETIVOS DEL TRABAJO

El objetivo de este Trabajo de Fin de Grado es analizar si la latencia de respuesta de un sistema RAG puede actuar como un canal lateral temporal capaz de revelar información sobre la presencia de datos en su base de conocimiento, y desarrollar una herramienta software que permita evaluar este riesgo en entornos controlados. Para alcanzar este objetivo, el trabajo combina una fase de investigación experimental con una fase de desarrollo de software. La primera permite estudiar el comportamiento temporal de distintos escenarios RAG y la segunda permite convertir las conclusiones obtenidas en una herramienta de auditoría defensiva.

Este objetivo se concreta en el análisis de si las consultas relacionadas con información presente en una base vectorial producen un comportamiento temporal distinto al de consultas relacionadas con información ausente o poco representada. A partir de esa señal, se busca desarrollar una herramienta capaz de ejecutar auditorías de caja negra contra sistemas RAG externos mediante peticiones HTTP que registre los tiempos observados y presente resultados estadísticos que ayuden a valorar la existencia de una posible fuga de información.

Para cumplir el objetivo general, se definen los siguientes objetivos específicos:

1. Analizar el estado del arte relacionado con la privacidad en sistemas RAG, los ataques de inferencia de membresía, los canales laterales temporales en modelos de lenguaje y las vulnerabilidades asociadas a mecanismos de caché, con el fin de identificar la brecha de investigación que aborda este trabajo.

2. Diseñar y ejecutar una metodología experimental que permita estudiar la latencia en sistemas RAG de forma controlada. Este objetivo incluye la definición del modelo de amenaza, la identificación de variables relevantes, la construcción del corpus y del banco de consultas, la validación inicial en un entorno reducido, el escalado a un corpus de mayor tamaño, la evaluación de la recuperación vectorial aislada, el análisis de un pipeline RAG completo y el estudio del efecto de distintos mecanismos de caché.

3. Diseñar e implementar una herramienta web de auditoría que permita evaluar sistemas RAG externos desde una perspectiva de caja negra. Con este objetivo se realizará la definición de requisitos funcionales y no funcionales, el diseño de la arquitectura, la gestión de targets y datasets, la ejecución asíncrona de auditorías, la medición del Time-to-First-Byte, la persistencia de resultados, la calibración temporal, la clasificación de resultados como probablemente presentes, probablemente ausentes y la visualización de métricas estadísticas.

4. Construir un entorno RAG externo de validación que permita comprobar el funcionamiento real de la herramienta desarrollada. Para la realización de este objetivo será necesario preparar un sistema RAG accesible mediante un endpoint HTTP, cargar una base documental controlada, configurar el modelo de embeddings y el modelo de lenguaje, ejecutar auditorías e inferencias contra ese entorno y comprobar que los flujos principales de la herramienta funcionan correctamente en un escenario representativo.

5. Identificar las limitaciones del enfoque propuesto y analizar su impacto desde una perspectiva defensiva, ética y social. Este objetivo incluye el estudio de las limitaciones asociadas al ruido de red, la estabilidad del sistema auditado, la dependencia de la calibración, la interpretación estadística de los resultados y las posibles líneas de trabajo futuras.

Con estos objetivos, el trabajo busca cubrir todo el recorrido necesario para pasar de una hipótesis de seguridad a una herramienta funcional. Primero se estudia el contexto técnico y científico del problema, después se analiza experimentalmente si la señal temporal existe en distintos escenarios y, finalmente, se implementa y valida una solución que permite reproducir ese análisis de forma más ordenada.

1.4. ALCANCE DEL TRABAJO

El alcance de este Trabajo de Fin de Grado se centra en el estudio de un canal lateral temporal en sistemas RAG y en el desarrollo de una herramienta que permita auditar ese comportamiento en entornos controlados. El trabajo analiza una posible fuga de información desde una perspectiva defensiva, con el objetivo de que pueda ser interpretada y tenida en cuenta en el diseño de arquitecturas RAG más seguras.

Desde el punto de vista de la investigación, el trabajo estudia si existen diferencias de latencia entre consultas asociadas a información presente en una base vectorial y consultas asociadas a información ausente o poco relacionada con el corpus. Para ello, se parte de escenarios controlados donde se puede conocer qué información ha sido indexada y se realizan pruebas progresivas. Primero se analiza la recuperación vectorial aislada [6], después se incorpora un pipeline RAG completo con modelo de lenguaje [2], [3] y finalmente se estudia el efecto de distintos mecanismos de caché [12]-[15]. Esta progresión permite observar si la diferencia de la hipótesis aparece y cómo cambia esta cuando se añaden componentes que introducen ruido al sistema.

El análisis se limita a señales temporales observables desde el exterior, lo que significa que el enfoque propuesto no necesita acceder ni conocer ninguna configuración del sistema auditado. La herramienta desarrollada trabaja mediante peticiones HTTP contra un target configurado por el auditor y mide principalmente el Time-to-First-Byte. Por tanto, el modelo seguido es de caja negra, ya que solo se utiliza la información que un cliente externo puede observar durante la interacción con el sistema.

También forma parte del alcance el diseño e implementación de una herramienta web de auditoría, la cual permite registrar sistemas RAG externos, cargar datasets de consultas, ejecutar auditorías en segundo plano, guardar mediciones, visualizar resultados estadísticos y realizar inferencias de términos a partir de una auditoría previa.

Queda fuera del alcance del trabajo el acceso no autorizado a sistemas reales, la extracción directa de documentos, la evasión de controles de seguridad y la explotación sobre servicios de terceros sin permiso. Las pruebas planteadas se realizan sobre entornos propios o controlados, y la herramienta está pensada para ser usada por un auditor autorizado que quiera evaluar la exposición temporal de un sistema RAG bajo su responsabilidad. Tampoco se pretende demostrar que cualquier sistema RAG sea vulnerable en todos los escenarios. La latencia depende de muchos factores, como la carga del servidor, la red, la configuración del retriever, el modelo utilizado, el valor de k, la caché y la estabilidad del entorno. Por ello, los resultados obtenidos deben interpretarse como evidencia estadística dentro de unas condiciones concretas, no como una afirmación universal.

Por último, el trabajo no busca proponer una solución definitiva para eliminar este tipo de canal lateral. Se comentan limitaciones y posibles líneas de mitigación, pero el objetivo principal es identificar el riesgo, estudiarlo experimentalmente y proporcionar una forma práctica de auditarlo. La mitigación completa de fugas temporales en arquitecturas RAG queda abierta como una línea de trabajo futura, especialmente en entornos productivos con tráfico real, políticas de acceso complejas y cargas variables.

1.5. ENFOQUE METODOLÓGICO SEGUIDO

El desarrollo de este Trabajo de Fin de Grado se ha planteado siguiendo un enfoque progresivo, combinando investigación experimental e ingeniería del software. La primera fase del trabajo ha consistido en estudiar el contexto técnico y científico del problema. Para ello, se ha revisado literatura relacionada con el tema abordado, permitiendo identificar qué partes del problema ya habían sido estudiadas y dónde existía una brecha concreta relacionada con la latencia del proceso de recuperación vectorial. A partir de esa revisión, se definió una hipótesis de trabajo centrada en la existencia de diferencias temporales entre consultas asociadas a información presente y consultas asociadas a información ausente o poco representada dentro de una base vectorial. Para poder analizar esta hipótesis, se diseñó una experimentación controlada, separando las variables que podían influir en la medición.

Una vez diseñada la experimentación, esta se desarrolló de forma incremental, iniciando con el estudio de la recuperación vectorial de manera aislada, con el objetivo de observar si la señal temporal podía detectarse sin el ruido introducido por otros componentes del sistema. Después, el análisis se amplió a un escenario más completo, incorporando un pipeline RAG con modelo de lenguaje para comprobar si la señal seguía siendo observable cuando intervenían más fases de procesamiento y, finalmente, se analizaron escenarios con caché, ya que estos mecanismos son frecuentes en sistemas reales y pueden modificar de forma importante los tiempos de respuesta.

Los resultados obtenidos durante esta fase experimental sirvieron como base para el diseño de la herramienta, ya que las funcionalidades principales se definieron a partir de las necesidades observadas durante las pruebas: lanzar consultas de forma repetida, mezclar el orden de ejecución, medir tiempos, guardar resultados individuales, comparar grupos, calcular métricas estadísticas y visualizar diferencias temporales. La experiencia obtenida con los scripts experimentales también ayudó a trasladar la metodología de medición a una herramienta más ordenada, con una arquitectura preparada para trabajar en caja negra mediante endpoints HTTP. Por otro lado, el desarrollo de la herramienta comenzó con la definición de sus requisitos funcionales y no funcionales. A partir de ellos se diseñaron la arquitectura, los procesos principales y la separación entre interfaz, backend, base de datos, sistema de colas y

worker. Esta separación permitió mantener una forma de trabajo coherente con el objetivo del proyecto: configurar pruebas desde una interfaz web, ejecutar las mediciones en segundo plano y analizar los resultados sin interferir directamente en los tiempos observados.

Por último, la herramienta se validó mediante pruebas funcionales y experimentales sobre un entorno RAG preparado para este trabajo. En la construcción de este entorno también fue útil la experimentación previa, ya que los scripts iniciales ya habían permitido probar tecnologías necesarias para crear un pipeline RAG realista accesible mediante un endpoint. Esta validación permitió comprobar que los flujos principales funcionaban correctamente. Así, la metodología seguida cubre todo el recorrido del proyecto, desde la identificación de una posible vulnerabilidad hasta la construcción de una herramienta capaz de evaluarla de forma práctica.

1.6. ESTRUCTURA DE LA MEMORIA

La memoria se organiza en nueve capítulos principales y cuatro anexos. Esta estructura permite presentar primero el contexto y la base teórica del problema, después el diseño y desarrollo de la herramienta y, finalmente, la validación experimental y las conclusiones obtenidas.

El capítulo 1 introduce el trabajo, explicando el contexto en el que aparecen los sistemas RAG, la motivación del proyecto, el problema abordado, los objetivos definidos, el alcance de la investigación y el enfoque metodológico seguido.

El capítulo 2 recoge el estado del arte relacionado con la privacidad y seguridad en sistemas RAG, los ataques de inferencia de membresía, los canales laterales en modelos de lenguaje y las vulnerabilidades asociadas a mecanismos de caché. Este capítulo permite situar el trabajo dentro de la literatura existente e identificar la brecha de investigación que se aborda en la memoria.

El capítulo 3 presenta el análisis de requisitos de la herramienta desarrollada. En él se definen los requisitos funcionales y no funcionales, el modelo de casos de uso y la matriz de trazabilidad que relaciona los requisitos con las funcionalidades principales del sistema.

El capítulo 4 describe el diseño y la metodología del sistema. Se explica la arquitectura general de la solución, el diseño funcional de los flujos de auditoría e inferencia de términos, el sistema de medición temporal, el modelo de persistencia, la interfaz de usuario y las principales decisiones de diseño tomadas durante el proyecto.

El capítulo 5 desarrolla la implementación de la herramienta. En este capítulo se presentan las tecnologías utilizadas, la organización del proyecto, el backend, el procesamiento asíncrono, el módulo de medición temporal, el análisis estadístico, la interfaz de usuario y la forma de ejecución de la aplicación.

El capítulo 6 contiene las pruebas y la evaluación experimental. Primero se describe el diseño experimental previo y después se presentan las ejecuciones realizadas para analizar la recuperación vectorial aislada, el pipeline RAG con modelo de lenguaje, los escenarios con caché y la validación de la herramienta desarrollada.

El capítulo 7 recoge los aspectos económicos y sociales del proyecto. En él se incluye el presupuesto estimado del trabajo y se analiza el impacto social y ético de una herramienta orientada a auditar posibles fugas de información en sistemas RAG.

El capítulo 8 presenta las conclusiones del Trabajo de Fin de Grado. En este apartado se valora el grado de cumplimiento de los objetivos, se resumen las principales limitaciones encontradas y se plantean posibles líneas de trabajo futuras.

El capítulo 9 incluye la bibliografía utilizada durante la elaboración de la memoria, recogiendo los trabajos científicos, documentación técnica y fuentes consultadas para contextualizar y desarrollar el proyecto.

Por último, los anexos complementan la memoria principal con información práctica. El Anexo A muestra las principales pantallas de la interfaz de usuario. El Anexo B recoge el manual de instalación y ejecución de la herramienta. El Anexo C presenta los formatos de entrada y salida utilizados por la aplicación, incluyendo ejemplos de configuración de targets, datasets, entradas de inferencia y archivos exportados. El Anexo D recoge la declaración de uso de Inteligencia Artificial Generativa, indicando las herramientas utilizadas, los tipos de apoyo recibidos y una reflexión sobre su utilidad y sus limitaciones durante el desarrollo del trabajo.

2. ESTADO DEL ARTE

En este capítulo se realiza una revisión crítica de la literatura científica más reciente relacionada con la seguridad en arquitecturas de inteligencia artificial generativa. Concretamente, se analizarán los avances y vulnerabilidades descubiertas en los sistemas de Generación Aumentada por Recuperación (RAG) [2], [3], los ataques fundamentados en canales laterales (*side-channel attacks*) [4], [5] sobre Modelos de Lenguaje de Gran Escala (LLM) [1], y las brechas de privacidad emergentes en los mecanismos de memoria caché [12]-[15]. Este análisis permitirá contextualizar el presente Trabajo de Fin de Grado y justificar la relevancia de la brecha de investigación abordada.

2.1. PRIVACIDAD Y SEGURIDAD EN SISTEMAS RAG

Los LLM han revolucionado el procesamiento del lenguaje natural. Sin embargo, se ha visto con el paso del tiempo que tienen tendencia a generar información falsa por las alucinaciones. Por esto, muchas empresas están implementando en sus sistemas de *chatbots* de IA las arquitecturas RAG. Estas ayudan a mejorar ampliamente la precisión de los LLM al utilizar partes de documentos de una base vectorial como contexto para el LLM. El principal problema es que, como podemos ver en varias investigaciones recientes, introducen muchos problemas de seguridad y privacidad en los sistemas [8], [9], [14], [15].

2.1.1. INFERENCIA DE MEMBRESÍA EN LA RECUPERACIÓN VECTORIAL

El modelo de arquitectura RAG aparece como una solución a las limitaciones de los modelos de lenguaje pre-entrenados [2], [3]. Algunos de los problemas que estos sistemas solucionan son las alucinaciones y falta de veracidad, conocimiento obsoleto o desactualizado y la falta de precisión en dominios de datos privados. Al integrar un sistema de recuperación que extrae fragmentos de documentos desde una base de datos vectorial para contextualizar el *prompt* del modelo, se consigue una precisión muy superior. Sin embargo, este proceso crea una nueva ventana de ataque al unir directamente el contenido de la base de datos al modelo de lenguaje.

La principal amenaza sobre estas arquitecturas actualmente se materializa a través de los Ataques de Inferencia de Membresía (*Membership Inference Attacks* o MIA) [7]. Estos ataques tienen como objetivo descubrir si un documento específico está indexado en la base de datos de recuperación de una organización. Cualquier ataque que afirme que un documento confidencial se encuentra o no en una base de datos ya genera una violación grave de la privacidad. Anderson et al. [8] estudian este problema mediante ataques de inferencia de membresía sobre bases de recuperación RAG. Los autores demuestran cómo las probabilidades de predicción de los tokens (*log-probabilities*) emitidas por el LLM pueden revelar la presencia de archivos corporativos en una base de datos. Las *log-probabilities* en un LLM son las probabilidades que un modelo asigna a cada token posible al generar la siguiente unidad de texto. Técnicamente, representan la confianza del modelo en su predicción, indicando mayor confianza cuanto más cercana al 0 sea.

Esta investigación explica que el problema no está en el software del modelo, sino en el funcionamiento de la arquitectura interna del mismo. Cuando el sistema usa un documento real en su contexto, el LLM muestra mayor "facilidad" para predecir los siguientes tokens, lo que se traduce en unas *log-probabilities* más cercanas a 0. Para evaluar esto, los autores utilizaron dos escenarios distintos:

1. Ataques de Caja Negra: En este escenario, el atacante solo tiene acceso al texto final generado por el chat. A pesar del poco conocimiento sobre el sistema, el estudio revela que, mediante el análisis de la salida y su abundancia de detalles, el atacante puede entrenar clasificadores que detecten la aparición de un documento. Sin embargo, al aplicar este método encontraron el obstáculo de las alucinaciones, en las que el modelo generaba respuestas con seguridad alta sobre datos desconocidos o sin contexto, lo que elevaba la tasa de falsos positivos. Aun así, en pruebas con modelos como Mistral-7B, se alcanzaron valores de AUC-ROC cercanos a 0.8, lo que demuestra que, incluso sin ver los parámetros internos, el sistema filtra información [8].

2. Ataques de Caja Gris: Este escenario también es realista, ya que hay modelos que devuelven metadatos como las log-probabilities al usarlos a través de sus API. En este caso, el atacante tenía acceso a las *log-probabilities* y podía medir con precisión cuánta confianza tenía el modelo en cada palabra. Anderson et al. [8] diseñan reglas de decisión con esta métrica en la que, si la probabilidad acumulada superaba cierto umbral, el documento se consideraría presente en la base de datos. Los resultados aumentaron considerablemente frente al anterior escenario con una media de 0.9 en AUC-ROC, lo que muestra que el ataque obtiene un rendimiento muy alto en ese escenario.

Por todo esto, los autores concluyen que existe una correlación directa entre la presencia de un documento en la base de datos de recuperación y una disminución drástica en la entropía de la respuesta del modelo. Para aportar fuerza a sus conclusiones, los autores evaluaron el ataque sobre conjuntos de datos de alta sensibilidad. Se usó el dataset *Enron* (correos electrónicos corporativos reales) y *HealthCareMagic* (consultas médicas), que permitió observar que el ataque es especialmente efectivo en textos con terminología técnica. Esto sugiere que cuanto más "único" es el documento que una empresa intenta proteger en su RAG, más fácil es para un atacante detectar su presencia mediante este tipo de canales laterales [8].

2.1.2. EXPLOTACIÓN DEL PRESUPUESTO DE TOKENS

Además del análisis de confianza de una salida, los sistemas de LLM tienen muchas más métricas que se pueden estudiar, que también suponen un vector de ataque. Una de ellas es la limitación por tokens de los modelos. En entornos reales y plataformas comerciales, la generación de texto está estrictamente limitada por cuotas de tokens para garantizar la eficiencia y, sobre todo, el control de costes. Li et al. [9] utilizan esta restricción de diseño como base para proponer un ataque de inferencia de membresía basado en el presupuesto de generación.

La hipótesis de este trabajo trata sobre la diferencia de comportamiento de un Modelo de Lenguaje de Gran Escala cuando agota la limitación de tokens en la generación de una respuesta frente a cuando la respuesta es menor a la limitación de tokens. Al recibir una consulta sobre un tema específico, si en la arquitectura RAG se encuentra el documento pertinente y lo inyecta en el contexto, el LLM intentará formular una respuesta más larga y completa sobre dichos datos. Sin embargo, si el atacante puede modificar la limitación de tokens reduciéndola a unos pocos tokens, el modelo sufrirá una parada forzada en la generación o usará patrones semánticamente inusuales. Por el contrario, si el documento no se encuentra en la base de datos, el sistema generará una respuesta negando el conocimiento de los datos pedidos, como: "No dispongo de información suficiente", las cuales no suelen consumir los tokens limitantes por completo [9].

Para validar esta hipótesis, Li et al. [9] diseñan dos escenarios de amenaza distintos:

1. Conocimiento Parcial: En este escenario, se asume que el atacante posee información sobre la configuración general del RAG objetivo, como la estrategia de fragmentación. Utilizando arquitecturas iguales a la atacada, el atacante entrena un modelo clasificador capaz de identificar los patrones generados por una parada repentina en el texto de salida. Este clasificador aprende a discernir con gran agudeza los defectos que delatan la presencia de información inyectada.

2. Conocimiento Cero: Representa el entorno más realista, ya que el ataque se produce sin ninguna noción sobre la infraestructura interna del sistema objetivo. En este caso, el ataque prescinde del entrenamiento supervisado. En su lugar, emplea algoritmos de aprendizaje no supervisado basados en *clustering* aplicados sobre un dataset de respuestas obtenidas tras enviar consultas de manera sistemática. Las métricas muestran que las respuestas contextualizadas por la base de datos vectorial tienden a agruparse en clústeres radicalmente distintos a las respuestas "vacías".

Al realizar este ataque, los investigadores concluyeron que su hipótesis era cierta al observar resultados sorprendentemente altos. Se recogieron valores del 80% al 91% en precisión haciendo las pruebas con bancos de preguntas estandarizados como Natural Questions y MS-MARCO y utilizando datasets de alta privacidad como HealthCareMagic-100k [9].

2.2. ATAQUES DE CANAL LATERAL EN MODELOS DE LENGUAJE

Como se ha mencionado en la sección anterior, las estrategias para comprometer la privacidad de un sistema basado en LLM dependen de la interacción directa con los parámetros del modelo, analizando sus respuestas textuales o sus restricciones de tokens. Sin embargo, la implementación de Modelos de Lenguaje de Gran Escala introduce una superficie de

explotación que ocurre incluso usando las defensas criptográficas convencionales: los ataques de canal lateral (*side-channel attacks*). Estos ataques no se basan en la explotación de vulnerabilidades en el código o en la ruptura de algoritmos de cifrado, sino en la observación de variables físicas que proporcionan información sobre el estado interno del sistema y los datos procesados [4], [5]. Entre ellos encontramos algunos como el consumo energético o las latencias y el tráfico de red [10], [11].

2.2.1. FUGAS DE INFORMACIÓN EN LA INFERENCIA AUTOREGRESIVA

Para comprender la viabilidad de un ataque de canal lateral sobre un modelo de lenguaje, es necesario analizar la arquitectura de la fase de inferencia. La mayoría de los LLM modernos funcionan de forma autorregresiva [16], esto quiere decir que el modelo genera el texto secuencialmente, palabra por palabra, contextualizando cada nueva predicción con la secuencia previamente generada. Esta generación se divide en dos fases:

1. Fase de prellenado: En esta etapa, el sistema procesa el *prompt* inicial del usuario de manera paralela. Para ello se cargan los pesos del modelo en la GPU y se calculan las representaciones matemáticas de cada palabra [16]. El tiempo de esta fase es proporcional a la longitud de la consulta.

2. Fase de decodificación: Aquí es donde reside la principal vulnerabilidad de canal lateral. El modelo genera el texto token a token. Cada nuevo token requiere una pasada hacia delante por el sistema. El sistema accede constantemente a la VRAM para recuperar los pesos y el historial de la conversación, conocido como *KV-Cache* [12].

Es en la fase de decodificación donde la latencia inter-token (el tiempo transcurrido entre la emisión de una palabra y la siguiente) se convierte en una señal no constante, ya que tiene variaciones sutiles que dependen de la arquitectura [11]. Cosas como la dificultad semántica de la palabra a predecir o la complejidad del razonamiento requerido hacen que estas variaciones ocurran en la generación. Cuando estos modelos se exponen a través de internet mediante conexiones cifradas (TLS/HTTPS) [17], el texto plano queda protegido, pero los tiempos de llegada de los paquetes de red permiten reconstruir patrones semánticos complejos sin necesidad de descifrar el contenido.

Un trabajo especialmente relevante en este ámbito es el de McDonald y Bar Or [11], que estudia este tipo de canal lateral mediante el análisis de señales de red, en lugar de analizar el contenido textual generado por el modelo. En este estudio se demuestra que un atacante, situado en un nodo intermedio de la red, solo capaz de escuchar el tráfico de red entre el servidor y el usuario, es capaz de clasificar el contenido del que se habla en una conversación con una precisión muy alta. El ataque se centra en capturar los paquetes de red, quitando el tráfico de control para solo usar los paquetes de datos que transportan los tokens. Además, el ataque se centra exclusivamente en dos variables: el tamaño de los paquetes de red y el tiempo entre las llegadas de las respuestas de cada token o tiempo de inter-llegada.

- Tamaños de paquete: Aunque los tokens se empaquetan en tramas cifradas de tamaño variable, la arquitectura del LLM y la configuración del servidor fuerzan agrupaciones de tokens que generan firmas de tamaño predecibles.

- Patrones de inter-llegada: Representa el tiempo con el que el motor de inferencia actúa. En este estudio se observó que temas con terminología más concreta y específica, como

la medicina, provocan una cadencia de generación distinta a la de una conversación casual. Por eso se utilizó como la métrica con mayor peso en el clasificador.

El entorno usado en este estudio fue orientado a detectar actividades de alta privacidad. En este caso se eligió el tema de lavado de dinero, ya que es un tema de valor muy alto para agencias gubernamentales y entidades financieras. Para hacer esto se usó el dataset *Quora Questions Pair,* que es una colección de datos diseñada para entrenar modelos de aprendizaje automático en la detección de preguntas duplicadas, que contiene más de 11.000 preguntas sobre diversos temas y se le inyectaron 100 consultas diseñadas para imitar escenarios de lavado de dinero.

Se utilizaron arquitecturas de redes neuronales convolucionales para realizar la clasificación. El clasificador fue capaz de distinguir si las latencias observadas en una traza de red de 30 segundos correspondían a un tema relacionado con el lavado de dinero. Los resultados obtenidos demostraron que, incluso ante la presencia de variaciones normales en el tráfico de internet, la señal emitida por el LLM era suficientemente robusta como para que el ataque mantuviese una tasa de acierto cercana al 95% [11].

Este hallazgo es muy importante, ya que demuestra que el aislamiento de los modelos no impide que la infraestructura filtre información. El procesamiento de cada consulta es único, pero predecible, dejando una huella que es consistente en diferentes plataformas, tanto de código abierto como privadas [11].

2.2.2. IMPACTO COMERCIAL Y ESTRATEGIAS DE MITIGACIÓN

Los ataques de canal lateral no son uniformes para todos los sistemas de LLM ya que dependen plenamente de la optimización del software y de la infraestructura. McDonald y Bar Or [11] analizan una batería de pruebas sobre distintos modelos comerciales, incluyendo Gemini, Grok y modelos de OpenAI. En todos ellos se reportaron valores muy altos de precisión y recall. Además, se analizó el ataque cuando se utilizaba por separado el tamaño de los paquetes y la latencia, y pudieron observar que también se conseguía comprometer la privacidad del usuario, aunque no tanto como con los dos datos juntos.

La importancia de estos hallazgos provocó la necesidad de comunicar a las diferentes empresas los problemas con sus modelos, llevando a los proveedores de estos a implementar parches de seguridad de urgencia. Las medidas utilizadas en estos casos se clasifican dentro de las técnicas de ofuscación de tráfico. En este caso se implementó una medida para cada dato utilizado:

1. Solución para tamaño de paquetes: La técnica utilizada fue el empaquetado de paquetes usando *padding* [11]. Esta técnica consiste en modificar la capa de transporte para que todos los paquetes de red tengan un tamaño uniforme, independientemente del número de tokens que transporten. Si bien esto elimina la "firma de tamaño", no soluciona el problema de la cadencia temporal.

2. Solución para tiempo inter-token: Se utilizó la inyección de retrasos o *jitter* [10], [11]. Esto consiste en introducir retrasos aleatorios artificiales en la fase de decodificación. El *jitter* debe ser lo suficientemente grande como para evadir la clasificación del atacante, pero lo suficientemente pequeño como para que no suponga un problema en la UX. Sin embargo, se ha demostrado que, para que el ruido sea efectivo, la latencia del sistema tendría que incrementarse de forma significativa [11].

El principal problema de estas soluciones es que perjudican directamente al usuario final. Añadir retrasos aleatorios o incrementar el tamaño de los paquetes empeora drásticamente la latencia del usuario final y aumenta el consumo del proveedor. Para una empresa que utiliza RAG para analizar miles de documentos, un incremento de la latencia es inaceptable. El tiempo de respuesta es una de las métricas más importantes en el mercado de los LLM. Por tanto, la mayoría de las mitigaciones actuales no son suficientemente efectivas y sugieren que el canal lateral temporal es una vulnerabilidad crítica de los sistemas autorregresivos que no puede eliminarse de forma competente sin un cambio de la arquitectura interna.

2.3. VULNERABILIDADES EN MECANISMOS DE CACHÉ

El rendimiento de los modelos de Inteligencia Artificial ha hecho que sea necesaria una gestión avanzada de la memoria. Como en los LLM la latencia es el principal problema en la experiencia de usuario, las empresas se han centrado en reducirla lo máximo posible. Para esto se han empezado a implementar en las arquitecturas los sistemas de cachés para almacenar algunos resultados precomputados y así evitar la repetición del proceso de inferencia en preguntas similares [12], **¡Error! No se encuentra el origen de la referencia.**. El principal problema de esta implementación es la introducción de una nueva vulnerabilidad en los sistemas, ya que la diferencia de tiempo entre acceder solo a la caché para recuperar una respuesta y realizar el proceso completo de inferencia para generar una nueva es muy grande, además de determinista [14], [15]. En esta sección se analizará la infraestructura implementada para reducir la latencia, así como cómo esta puede comprometer los datos privados de conversación de los usuarios.

2.3.1. JERARQUÍA DE MEMORIA: KV-CACHE Y PROMPT CACHING

Para entender cómo un sistema de caché puede generar una vulnerabilidad, es necesario entender más a fondo los procesos de una arquitectura de este estilo. En este caso nos centraremos en el coste de los procesos, ya que las cachés se aplican principalmente para reducir los más costosos. En este caso, el proceso más costoso dentro de la inferencia es el cálculo de relevancia [16], que ocurre tanto en la fase de prellenado (solo una vez para todo el prompt) como en la fase de decodificación (cada vez que se genera una nueva palabra). Este proceso requiere la computación de grandes matrices de Claves y Valores para cada token de la secuencia [16]. Es por esto que se utilizan mecanismos de caché para poder guardar el contexto computado y así no tener que hacer de nuevo los cálculos completos. Los mecanismos que intervienen en este proceso son:

1. KV-Cache (Caché de Claves y Valores): Esta caché es generada en el proceso de prellenado y guarda las matrices calculadas de Claves y Valores del prompt entregado al sistema [12]. Esta caché utiliza la VRAM de la GPU y se usa en el proceso de la decodificación. De este modo, el sistema, al generar un nuevo token, solo tendrá que calcular la relevancia para el nuevo token y concatenarla a la existente en la caché. Esto reduce considerablemente la necesidad de cómputo y, por lo tanto, el coste del proceso.

2. Prompt Caching y Prefix Caching: Este mecanismo de caché funciona a un nivel mayor, ya que se basa en el funcionamiento de cualquier sistema de caché. En este caso, este se implementa para que, si dos usuarios hacen consultas sobre el mismo contexto, por ejemplo, sobre el mismo documento en un RAG, se guarde el estado del prefijo, o sea,

de la primera parte del contexto. Por lo que el primer usuario sí que tendrá un coste temporal mayor, pero el resto que pregunten sobre el mismo documento no [15].

Estos dos mecanismos hacen que el funcionamiento de los sistemas con LLM sea mucho mejor al reducir drásticamente su latencia, pero son justo los mecanismos que hacen que el ataque sea posible. Ya que, si una consulta se procesa utilizando la caché, su Tiempo hasta el Primer Token (*Time-to-First-Token*) será mucho menor [14], [15]. Por lo que permite a los atacantes inferir si algo ha sido procesado con anterioridad o no, rompiendo de nuevo la privacidad del sistema.

2.3.2. ATAQUES EN CACHÉS SEMÁNTICAS

Zheng et al. [14] analizan esta amenaza y muestran cómo las cachés compartidas en sistemas de LLM pueden utilizarse para reconstruir *prompts* o inferir temas de consulta de otros usuarios. En concreto estudian las cachés de sistemas *multi-tenant* donde distintos usuarios acceden al mismo motor de inferencia.

El ataque propuesto por Zheng et al. [14] se basa en la medición del tiempo de respuesta utilizando consultas diseñadas específicamente para que contengan fragmentos de información sensible. Si el sistema devuelve la respuesta con una latencia extremadamente baja, el atacante puede afirmar que ha ocurrido un acierto en caché, lo que implica que otro usuario ha introducido previamente ese mismo fragmento o uno muy similar. El estudio se centra en observar estas mediciones y demostrar cómo la diferencia entre el acierto en caché y el fallo en caché es suficientemente amplia como para medirlas incluso a través del ruido de internet. Este ataque fue validado tanto con modelos *open source,* utilizando modelos de la familia LLaMA, como con modelos comerciales de OpenAI.

Para aportar robustez al trabajo, se usó un dataset de interacciones entre médicos y pacientes llamado *Chatdoctor* con más de 110.000 interacciones. Los investigadores demostraron que, mediante el envío de consultas específicas relacionadas con síntomas o enfermedades raras, un atacante puede determinar si un paciente en concreto ha consultado algo sobre una patología concreta observando únicamente la velocidad de respuesta del modelo [14].

Es relevante mencionar que Zheng et al. [14] concluyen que la inferencia del tema, ya sea genérico o específico, resulta más efectiva que la reconstrucción literal de un prompt palabra por palabra. Esto ocurre, ya que las cachés utilizadas hoy en día en este tipo de sistemas suelen ser cachés semánticas. Las cachés semánticas son cachés que están diseñadas para reconocer redundancias a nivel semántico, lo que quiere decir que trabajan con un umbral de similitud y, si ese umbral es superado, la caché genera un hit aunque lo que se compare no sea exactamente igual **¡Error! No se encuentra el origen de la referencia.**.

2.3.3. FUGAS EN KV-CACHE MULTI-INQUILINO

En el estudio de Wu et al. [15] se estudia una amenaza relacionada, centrada en la fuga de *prompts* mediante el uso compartido de la KV-Cache en entornos *multi-tenant*. A diferencia del trabajo anterior, este ataque no se dirige a la caché de prefijos, sino a la KV-Cache. De nuevo, lo que se usa es una arquitectura *multi-tenant*, ya que son las más utilizadas. Estas se usan para ahorrar recursos y hacer que varios usuarios compartan el mismo hardware y así poder maximizar el uso de la VRAM de cada GPU [12].

El ataque analizado en [15] se basa en el envío de múltiples candidatos de tokens y en la medición del *Time-to-First-Token*. Si detecta una latencia muy baja, confirma un acierto en caché, lo que le indica que ese token específico es el que la víctima ha utilizado en ese momento. Al repetir este proceso, el atacante consigue reconstruir el *prompt* privado palabra por palabra. En este caso, el ataque se lleva a cabo en distintos escenarios, pero incluso en el peor de todos, en el que el atacante no conoce nada del usuario ni de lo que está diciendo, se consigue una precisión del 95% en la reconstrucción del *prompt* del usuario [15]. Para aplicar robustez en este ataque, solo se asume que el atacante sabe cosas públicas del sistema, como por ejemplo la forma de tokenizar que tiene el modelo.

La integración de bases de datos vectoriales con mecanismos de caché es lo mejor para la eficiencia en las arquitecturas RAG actuales. Sin embargo, tras analizar los trabajos de Zheng et al. [14] y Wu et al. [15], es bastante claro que esta aceleración del proceso solo en situaciones específicas es a su vez un generador de vectores de ataque a la privacidad del usuario.

2.4. COMPARATIVA DE TRABAJOS RELACIONADOS Y BRECHA DE INVESTIGACIÓN

Después de haber explicado las vulnerabilidades de los sistemas RAG y los ataques principales que las envuelven, en este apartado se estudian las limitaciones existentes y se explica con precisión el punto en donde este trabajo resulta relevante.

2.4.1. TAXONOMÍA DE LAS METODOLOGÍAS DE EXFILTRACIÓN

La revisión del estado del arte permite agrupar las amenazas contra la privacidad en LLM en tres dimensiones metodológicas claramente diferenciadas:

1. Ataques de inferencia basados en análisis textual: Los trabajos centrados en ataques de inferencia de membresía (MIA) como los de Anderson et al. [8] y Li et al. [9] se basan en interactuar con el modelo y analizar el texto que este devuelve. Estos ataques estudian cómo responde la IA o analizan los valores de confianza matemática que el modelo asigna a cada palabra generada. Son métodos altamente efectivos para comprobar si un párrafo específico ha sido proporcionado al modelo como contexto. Sin embargo, su principal carencia es la escalabilidad. Para que funcionen bien, el auditor necesita hacerle muchísimas preguntas al modelo (interacciones largas) o tener acceso a los valores matemáticos internos que los modelos comerciales suelen ocultar por seguridad. Por tanto, aunque son excelentes para validar un documento suelto, resultan demasiado lentos e inviables para auditar una base de datos corporativa masiva.

2. Ataques de canal lateral en tráfico de red: Investigaciones como la de McDonald y Bar Or [11] cambian el enfoque, ya que en lugar de mirar qué dice el texto, analizan cómo viaja por internet. Su gran fortaleza es que son ataques pasivos. El atacante no necesita enviarle preguntas a la IA, solo observar el tráfico de red cifrado, midiendo el tamaño de los paquetes de datos y el tiempo entre ellos. Estos ataques son muy eficaces para interceptar conversaciones y deducir el tema general del que se está hablando. El problema es que las conexiones a internet tienen fluctuaciones y retrasos, lo que las hace útiles para inferir contextos amplios, pero no para tener precisión sobre si un documento confidencial se encuentra indexado en el sistema RAG de una empresa o no.

3. Ataques centrados en la arquitectura: Los estudios más recientes sobre el uso compartido de cachés como los de Zheng et al. [14] y Wu et al. [15] se centran en medir las diferencias exactas en los tiempos de respuesta del servidor. Como han demostrado, son herramientas tremendamente precisas que incluso permiten reconstruir palabra por palabra lo que otro usuario está escribiendo en ese instante. Su limitación radica en su objetivo y requisitos. Estos trabajos están orientados a espiar a otros usuarios (robar *prompts*) y suelen requerir que el atacante y la víctima compartan la misma tarjeta gráfica en el servidor.

2.4.2. COMPARATIVA DE TRABAJOS RELACIONADOS

A continuación, se presenta la Tabla I, que sintetiza los parámetros técnicos de las investigaciones analizadas. Esta tabla permite visualizar las diferencias en cuanto a objetivos, vectores de ataque y requisitos de hardware, sirviendo como base para la identificación de la brecha de investigación.

Tabla I. COMPARATIVA DE TRABAJOS RELACIONADOS.

Referencia	**Vector de Ataque**	**Objetivo Primario**	**Métrica de Fuga**	**Dependencia de Hardware**
Anderson et al. [8]	Interfaz Lógica	Inferencia de Membresía	*Log-probabilities*	Baja (Caja Negra/Gris)
Li et al. [9]	Restricciones API	Inferencia de Membresía	Presupuesto de Tokens	Baja (API)
McDonald y Bar Or [11]	Canal Lateral (Red)	Clasificación de Temas	Tiempo y Tamaño	Media (Sniffing)
Zheng et al. [14]	Canal Lateral (Caché)	Robo de Inputs/*Prompts*	Latencia de Respuesta	Alta (Multi-tenant)
Wu et al. [15]	Arquitectura	Reconstrucción de *Prompt*	KV-Cache / Latencia	Muy Alta (GPU/VRAM)
Propuesta TFG	Canal Lateral (Tiempo)	Detección de Documentos Indexados	Latencia Semántica	Media (Redis/RAG)

Este análisis comparativo revela que existe una progresión hacia el uso de métricas temporales de baja latencia debido a su alta fiabilidad. Sin embargo, se observa una desconexión entre los

ataques de microarquitectura (extremadamente complejos y dependientes de compartir una GPU físicamente) y los ataques de red (demasiado ruidosos).

2.4.3. LAGUNAS EN LA LITERATURA ACTUAL

A pesar de los avances significativos en la identificación de vulnerabilidades dentro del ecosistema de los *chatbots*, el análisis del estado del arte revela que todavía hay muchas partes de estos sistemas inexploradas que podrían generar nuevos problemas de seguridad y privacidad. En este trabajo se han aprovechado estos puntos no explorados para diseñar y ejecutar un nuevo vector de ataque.

La inmensa mayoría de los ataques enfocados a la arquitectura como los analizados por Wu et al. [15] y Zheng et al. [14], dirigen sus esfuerzos exclusivamente hacia la memoria del modelo de lenguaje, en concreto a la KV-Cache o al *prompt caching*. Sin embargo, se ha dado por sentado que el corazón de la arquitectura RAG, el motor de recuperación, es seguro y no se puede usar para realizar estos ataques. Las bases de datos vectoriales modernas (como ChromaDB) utilizan algoritmos de búsqueda aproximada por vecinos más cercanos (ANN), siendo HNSW (Hierarchical Navigable Small World) el estándar [6], [18]. Es aquí justo donde encontramos la principal laguna crítica, ya que hay una notable falta de pruebas sobre este algoritmo. HNSW optimiza las búsquedas navegando a través de grafos multicapa [6]. El tiempo de recorrido computacional varía dependiendo de si el vector consultado ya forma parte del índice (o es extremadamente similar a uno existente) o si, por el contrario, es información desconocida. En este TFG, se ha aprovechado esta diferencia temporal para convertir los tiempos de respuesta del algoritmo HNSW en un vector de ataque.

Además, para añadir robustez al trabajo, se ha estudiado si el ataque también se presenta en un sistema con todos los componentes aportando ruido. En los entornos de producción real, un sistema RAG es un pipeline secuencial orquestado por frameworks como LangChain o LlamaIndex (involucrando modelos de embedding, bases de datos y el LLM final). Todas estas partes del sistema aportan ruido al tener su propio tiempo de computación. En este caso se ha estudiado si la diferencia de tiempo de recuperación entre una query cuyo vector cae en una zona poblada del RAG y otra cuyo vector cae en una más vacía es suficientemente grande como para medirla con todo el ruido generado por los distintos componentes del sistema.

Finalmente, se observa una clara distinción respecto a cómo se aplican los ataques. Por un lado, las investigaciones que intentan comprometer la base de datos de un RAG, como las analizadas por Anderson et al. [8] y Li et al. [9], dependen exclusivamente del análisis de texto o del conteo de tokens, lo cual es ineficiente a gran escala. Por otro lado, las investigaciones que utilizan canales laterales de latencia, como las de Zheng et al. [14] y Wu et al. [15]Ref5, se han limitado principalmente a comprometer la privacidad del usuario, buscando reconstruir las preguntas dinámicas de los chats. Lo que se ha pasado por alto es cruzar ambos ataques. En lugar de utilizar la latencia para "espiar" a otros usuarios, este TFG propone explotar las variaciones de tiempo del motor de búsqueda para atacar componentes estáticos, confirmando la existencia de documentos confidenciales indexados en el *corpus* sin necesidad de analizar el texto generado.

2.4.4. DEFINICIÓN DE LA BRECHA DE INVESTIGACIÓN

El conjunto de estas lagunas permite identificar una brecha de investigación clara y de alta relevancia, que es la ausencia del estudio de un método para auditoría de membresía de

documentos en sistemas RAG mediante el análisis de latencia del algoritmo de búsqueda vectorial. La investigación propuesta para explotar esta brecha se define con los siguientes elementos:

1. El vector: La latencia generada específicamente por el motor de búsqueda de la base de datos vectorial al procesar la similitud de consultas contra su índice.
2. El objetivo: Descubrir y confirmar la existencia de archivos específicos dentro del índice vectorial de una arquitectura RAG.
3. El contexto: Un entorno de auditoría corporativa realista (que incluye capas de optimización estándar como cachés semánticas) donde se busca validar si el sistema delata la existencia de documentos que no deberían estar expuestos.

Esta investigación, a diferencia de otros ataques, se centra en una vulnerabilidad algorítmica que es la propia eficiencia de la indexación y recuperación vectorial, midiendo el comportamiento del *retriever* y definiendo qué diferencia de tiempo constituye una prueba de la presencia de un documento en el RAG. Además, este trabajo se justifica gracias a su gran innovación en varios pilares:

- Viabilidad y realismo: A diferencia del ataque analizado por Wu et al. [15], esta metodología no requiere acceso físico o lógico a la GPU. Se basa en una interacción de red estándar con el servicio RAG, lo que la hace aplicable a prácticamente cualquier despliegue en la nube (AWS, Azure, Google Cloud).
- Enfoque en la privacidad de los datos: Al orientar el análisis hacia la base de datos vectorial, este trabajo ataca la principal preocupación de las empresas que utilizan sus propios *chatbots*, el miedo a que datos corporativos confidenciales se filtren por algún problema de seguridad.
- Formalización de la auditoría: No solo se pretende demostrar una vulnerabilidad, sino realizar una herramienta que permita a las empresas auditar sus propios sistemas RAG. También se propondrán algunas soluciones para mitigar la brecha de seguridad.

3. ANÁLISIS DE REQUISITOS

El análisis de requisitos es uno de los pilares fundamentales de cualquier proyecto de software. Este trabajo no solo se centra en la investigación del side-channel attack, sino también en el desarrollo de un producto de software funcional que ayude a la realización del mismo en distintos entornos.

El objetivo de esta sección del proyecto es poder identificar y documentar de manera precisa las necesidades del sistema para que pueda cumplir todos los requisitos de la investigación planteada. Este análisis nos ayudará a transformar las ideas y objetivos del proyecto en especificaciones técnicas concretas para asegurar que las fases de diseño y desarrollo sean exitosas en un futuro.

Para poder asegurar la calidad del proyecto, se ha seguido una metodología basada en estándares de ingeniería del software utilizados en entornos profesionales [19]. Por lo tanto, cada requisito ha sido evaluado y revisado para asegurar que el resultado final sea un producto de software funcional, claro y útil para la comunidad de investigación de ciberseguridad e inteligencia artificial. Además, todos los requisitos también se han planteado buscando la escalabilidad y robustez del producto.

3.1. DEFINICIÓN Y CLASIFICACIÓN DE REQUISITOS

La identificación de los requisitos se ha realizado siguiendo la nomenclatura UG-nnn, donde la "U" indica un requisito del usuario y la "G" se refiere a un requisito general del sistema. A continuación, se presenta la Tabla II, que recoge los requisitos funcionales y no funcionales que regirán el desarrollo del proyecto.

Tabla II. REQUISITOS FUNCIONALES Y NO FUNCIONALES DE LA HERRAMIENTA.

ID	Tipo	Nombre	Prioridad	Fuente	Estabilidad	Descripción
UG-001	Funcional	Autenticación de usuario	Alta	Cliente	Alta	El sistema permitirá al auditor iniciar y cerrar sesión para acceder a las pantallas internas de la herramienta.
UG-002	Funcional	Gestión de targets	Alta	Cliente	Alta	El sistema permitirá crear, consultar, editar, eliminar y probar targets RAG externos, configurando endpoint, headers, timeout, verificación TLS y plantilla del body.
UG-003	Funcional	Gestión de datasets	Alta	Cliente	Alta	El sistema permitirá cargar datasets en formato JSON, validar su estructura y mostrar una previsualización con el número de consultas y su distribución por grupos.
UG-004	Funcional	Creación de auditorías	Alta	Cliente	Alta	El sistema permitirá crear auditorías seleccionando un target, un dataset y parámetros de ejecución como peticiones de calentamiento y semilla aleatoria.
UG-005	Funcional	Ejecución asíncrona	Alta	Analista	Alta	El sistema ejecutará auditorías en segundo plano mediante un worker, permitiendo consultar su estado, progreso, errores y solicitar su cancelación.
UG-006	Funcional	Medición temporal de peticiones	Alta	Analista	Alta	El sistema medirá cada petición HTTP enviada al target, registrando Time-to-First-Byte, tiempo completo, código HTTP, tamaño de respuesta y posibles errores.
UG-007	Funcional	Análisis y visualización de auditorías	Alta	Cliente	Alta	El sistema mostrará métricas, gráficas, tablas, distribución de mediciones y comparaciones estadísticas entre grupos de frecuencia y longitud.
UG-008	Funcional	Creación de inferencias de términos	Alta	Cliente	Media	El sistema permitirá crear inferencias seleccionando un target, una fuente de calibración y términos o consultas agrupadas para evaluar.

UG-009	Funcional	Clasificación y visualización de términos	Alta	Analista	Media	El sistema clasificará los términos como probablemente presentes, probablemente ausentes o no concluyentes, mostrando métricas temporales, fronteras de calibración y evolución por rondas.
UG-010	Funcional	Exportación de resultados	Media	Cliente	Alta	El sistema permitirá exportar en CSV los resultados de auditorías e inferencias, incluyendo mediciones individuales, resúmenes estadísticos o resultados agregados por término según el caso.
UG-011	No funcional	Funcionamiento de caja negra	Alta	Analista	Alta	La herramienta deberá poder auditar sistemas RAG externos mediante HTTP, sin necesitar acceso al índice vectorial, documentos internos, modelo de embeddings o configuración privada del sistema auditado.
UG-012	No funcional	Precisión temporal sin interferencias	Alta	Analista	Alta	La ejecución de pruebas deberá realizarse en un worker separado de la interfaz y del backend de gestión, evitando que acciones visuales o peticiones de usuario interfieran en las mediciones.
UG-013	No funcional	Seguridad y protección de acceso	Alta	Cliente	Alta	Las rutas internas deberán estar protegidas mediante autenticación, hash de contraseña, token de sesión y cookie HTTPOnly, evitando acceso no autorizado a targets, headers, datasets y resultados.
UG-014	No funcional	Persistencia, trazabilidad y reproducibilidad	Alta	Analista	Alta	El sistema deberá conservar configuraciones, sesiones, mediciones, errores, semillas y resultados para permitir revisar pruebas anteriores y repetir ejecuciones bajo condiciones equivalentes.
UG-015	No funcional	Usabilidad y ejecución local controlada	Media	Cliente	Alta	La herramienta deberá poder usarse desde una interfaz web sin ejecutar scripts manuales y deberá poder levantarse en local con todos sus servicios.

3.2. MODELO DE CASOS DE USO

Para definir el alcance completo de nuestra herramienta es muy importante la utilización del Modelo de Casos de Uso ya que nos ayuda a establecer una base sólida para la posterior implementación y validación de la plataforma [20]. Para esto se ha llevado a cabo un proceso de análisis sobre el modelo para poder asegurar su coherencia y completa alineación con los requisitos funcionales explicados en el apartado anterior.

3.2.1. ACTORES

Para el desarrollo de los casos de uso lo primero es identificar qué actores interactuarán con el producto. En este caso el único actor involucrado será:

- **Auditor:** Representa al perfil técnico o auditor de ciberseguridad que utiliza la herramienta para evaluar si cierta información confidencial o términos específicos residen en la base de datos de una empresa, utilizando para ello el ataque por canal lateral de latencias.

3.2.2. DIAGRAMA DE CASOS DE USO

Para mostrar con mayor claridad las interacciones del actor con el sistema utilizamos el diagrama de casos de uso que podemos observar en la Fig. 1.

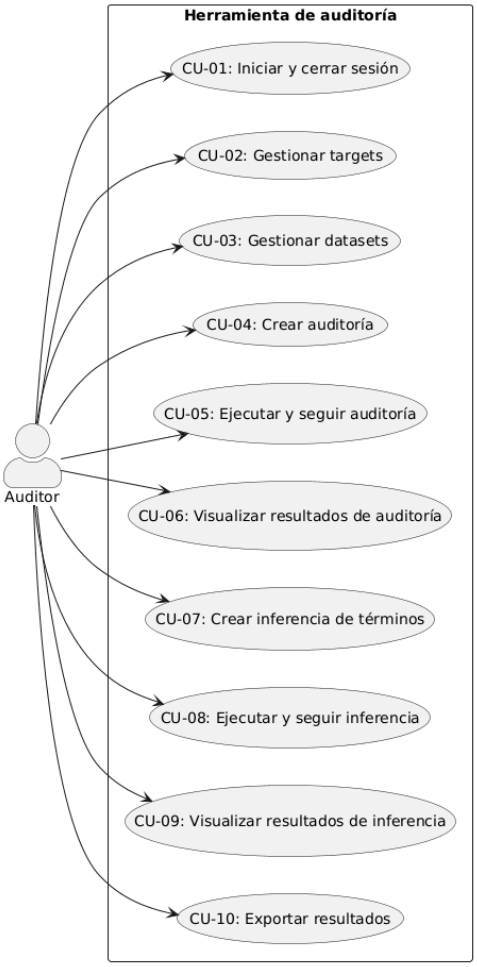


Fig. 1. Diagrama de casos de uso de la herramienta.

3.2.3. DESCRIPCIÓN DE ALTO NIVEL DE LOS CASOS DE USO

A continuación, se presenta la Tabla III, que recoge la especificación de alto nivel para cada uno de los casos de uso identificados en el diagrama del sistema. Esta descripción permite comprender de manera rápida la interacción principal entre el actor y la plataforma, así como el propósito fundamental de cada acción.

Tabla III. DESCRIPCIÓN DE ALTO NIVEL DE LOS CASOS DE USO.

Caso de Uso	Actor	Tipo	Descripción
CU-01	Auditor	Primario / Esencial	El auditor accede a la herramienta introduciendo sus credenciales. Si son correctas, el sistema crea una sesión y permite el acceso a las pantallas internas. Cuando termina, el auditor puede cerrar sesión y el sistema elimina la cookie de autenticación.
CU-02	Auditor	Primario / Esencial	El auditor crea y mantiene los sistemas RAG externos que quiere evaluar. Para ello introduce los datos necesarios. También puede probar la conexión para comprobar si el target responde antes de usarlo en una auditoría.
CU-03	Auditor	Primario / Esencial	El auditor sube datasets en formato JSON con consultas clasificadas por frecuencia y longitud. El sistema valida la estructura del archivo, lo guarda si es correcto y muestra una previsualización con el número de consultas y su distribución por grupos.
CU-04	Auditor	Primario / Esencial	El auditor crea una nueva auditoría seleccionando un target, un dataset y los parámetros de ejecución. El sistema registra la sesión y la deja preparada para su ejecución en segundo plano.
CU-05	Auditor	Primario / Esencial	El sistema ejecuta la auditoría mediante el worker, enviando las consultas al target y guardando las mediciones temporales obtenidas. Mientras la prueba avanza, el auditor puede solicitar la cancelación de la auditoría si es necesario.
CU-06	Auditor	Primario / Esencial	Mientras avanza la auditoría y una vez terminada, el auditor revisa los resultados calculados por la herramienta. El sistema muestra métricas, gráficas, tablas, distribución de mediciones y comparaciones estadísticas entre grupos de frecuencia y longitud.
CU-07	Auditor	Primario / Esencial	El auditor crea una prueba de inferencia seleccionando un target, una fuente de calibración y los términos o consultas que quiere evaluar. El sistema guarda la sesión y prepara la información necesaria para comparar esos términos con el perfil temporal de referencia.

CU-08	Auditor	Primario / Esencial	El sistema ejecuta la inferencia por rondas mediante el worker, enviando consultas asociadas a cada término y guardando sus mediciones. Durante la ejecución, el auditor puede cancelar la prueba si fuera necesario.
CU-09	Auditor	Primario / Esencial	Mientras se ejecuta una inferencia y cuando esta termina, el auditor revisa la clasificación de cada término. El sistema muestra si cada uno se considera probablemente presente, probablemente ausente o no concluyente, junto con métricas temporales, fronteras de calibración y evolución por rondas.
CU-10	Auditor	Secundario / Opcional	El auditor solicita la descarga de los resultados generados por la herramienta. El sistema crea archivos CSV con los datos de auditorías o inferencias, permitiendo conservar las mediciones, resúmenes o clasificaciones fuera de la aplicación.

3.3. MATRIZ DE TRAZABILIDAD

A partir del modelo de casos de uso, se ha construido la matriz de trazabilidad mostrada en la Tabla IV. Esta matriz permite comprobar que todos los requisitos definidos quedan cubiertos por al menos un caso de uso del modelo final, dejando el alcance del proyecto correctamente definido y validado. Una «X» en la intersección indica que el caso de uso de la columna satisface, implementa o se ve afectado por el requisito de la fila correspondiente.

Tabla IV. MATRIZ DE TRAZABILIDAD

Requisitos / Casos de Uso	**CU-01**	**CU-02**	**CU-03**	**CU-04**	**CU-05**	**CU-06**	**CU-07**	**CU-08**	**CU-09**	**CU-10**
UG-001 Autenticación de usuario	X									
UG-002 Gestión de targets RAG		X								
UG-003 Gestión de datasets			X							
UG-004 Creación de auditorías				X						

UG-005 Ejecución asíncrona					X					
UG-006 Medición temporal de peticiones					X			X		
UG-007 Análisis y visualización de auditorías						X				
UG-008 Creación de inferencias de términos							X			
UG-009 Clasificación y visualización de términos								X	X	
UG-010 Exportación de resultados										X
UG-011 Funcionamiento de caja negra		X			X			X		
UG-012 Precisión temporal sin interferencias					X			X		
UG-013 Seguridad y protección de acceso	X	X	X	X	X	X	X	X	X	X
UG-014 Persistencia, trazabilidad y reproducibilidad		X	X	X	X	X	X	X	X	X
UG-015 Usabilidad y ejecución local controlada	X	X	X	X	X	X	X	X	X	X

4. DISEÑO Y METODOLOGÍA DEL SISTEMA

Una vez definido el problema de seguridad y los requisitos del sistema, este apartado describe el diseño de la herramienta desarrollada para automatizar la auditoría de latencias en sistemas RAG. La herramienta se plantea como una plataforma de análisis que permite ejecutar consultas de forma controlada contra un sistema externo y estudiar si los tiempos de respuesta muestran diferencias estadísticamente significativas. Por tanto, el sistema no necesita acceder ni conocer ninguna configuración interna del entorno atacado, como su modelo de embeddings. Por esto tiene un funcionamiento de caja negra en el que el usuario define un sistema RAG objetivo, carga un conjunto de consultas y la herramienta se encarga de lanzar peticiones, medir tiempos, procesar resultados y mostrar gráficas con datos estadísticos, orientando al usuario sobre la efectividad de su ataque.

4.1. OBJETIVO DE DISEÑO DE LA HERRAMIENTA

El objetivo principal de la herramienta es permitir que un auditor pueda evaluar un sistema RAG desde el exterior, lanzando una batería de consultas y midiendo si existen diferencias de latencia asociadas a su contenido. A partir de esas mediciones, la herramienta también permite realizar una inferencia posterior de términos, comparando nuevas consultas contra el comportamiento temporal aprendido durante una auditoría previa. Además, esta herramienta surge como una profesionalización de unas pruebas explicadas en el apartado de diseño experimental previo en las que estas consultas se lanzaban de forma local con scripts automatizados a un sistema local. La herramienta convierte ese proceso experimental en un flujo reproducible. Con esto se busca que la explotación del canal lateral no dependa de ejecuciones manuales difíciles de repetir, sino de un procedimiento definido y medible.

Por otro lado, la herramienta introduce una de las variables evitadas en la fase experimental, que es el ruido de red, ya que en este caso, para realizar una auditoría a un sistema realista, es obligatorio tenerlo presente. Para ello, el sistema permite configurar el endpoint objetivo, definir el formato de las peticiones y ejecutar un dataset de consultas preparado previamente. Esta decisión permite que la herramienta sea independiente de la tecnología usada por el sistema auditado, siempre que pueda recibir consultas HTTP [21].

La entrada principal de una auditoría es un dataset de consultas clasificadas por frecuencia y longitud. Esta clasificación permite mantener la relación con las variables definidas en el diseño experimental. La frecuencia representa si la consulta está asociada a términos muy presentes, medianamente presentes o poco presentes en el corpus, y la longitud es el tamaño de la consulta, dividiéndose entre corto, medio y largo.

La salida de la herramienta está formada por métricas y resultados estadísticos útiles para interpretar la efectividad del ataque. Cada consulta lanzada contra el sistema genera una medición temporal que después se agrupa por frecuencia y longitud para calcular medias, medianas, percentiles y el p-value asociado a la comparación entre grupos [22]. El p-value es especialmente importante porque permite valorar si las diferencias observadas entre distribuciones pueden considerarse estadísticamente significativas.

Una vez realizada una auditoría, sus resultados pueden usarse como base de calibración para la inferencia de términos. En este segundo flujo, el usuario introduce nuevos términos o consultas que quiere evaluar contra el mismo sistema objetivo. La herramienta ejecuta nuevas

mediciones y compara sus tiempos con las distribuciones ya obtenidas en la auditoría anterior. Con esto se puede estimar si el nuevo término presenta un comportamiento más cercano a las consultas de alta presencia o baja presencia. La inferencia de términos se plantea como una segunda funcionalidad principal de la herramienta, posterior a la auditoría inicial. Su funcionamiento depende de que exista una auditoría previa sobre el mismo target, ya que necesita una referencia temporal para poder comparar las nuevas mediciones. Además, es prioritario que este target sea estable, ya que si no, este módulo generará mediciones distintas y la clasificación de los términos podría ser errónea.

El alcance de la herramienta se limita a sistemas sobre los que el auditor tenga autorización. Su diseño está orientado a pruebas de seguridad defensivas, investigación y validación de arquitecturas RAG propias.

4.2. ARQUITECTURA GENERAL DE LA SOLUCIÓN

Para la realización de una herramienta de este estilo es muy importante no interferir en los datos medidos, ya que es la única manera de poder saber si el ataque está siendo exitoso. Por esto la herramienta se ha separado en varias partes, para que el uso del sistema no interfiera en la ejecución de una prueba contra un entorno externo. Es muy necesario evitar cualquier tipo de latencia producida por llamadas de nuestro propio sistema, como podrían ser llamadas de la interfaz de usuario contra el backend. Con todo esto, la arquitectura definida se divide en cinco componentes principales: la interfaz de usuario, el backend de gestión, el sistema de colas, el sistema de ejecución de auditorías y la base de datos de la aplicación. Con esta arquitectura, la aplicación realiza los dos flujos principales de la herramienta: la ejecución de auditorías y la inferencia posterior de términos. Ambos se crean desde la interfaz, son coordinados por el backend, pasan por el sistema de colas y se ejecutan en segundo plano para no bloquear el funcionamiento normal de la aplicación.

La interfaz de usuario es donde el auditor configura el sistema que quiere evaluar, carga el dataset de consultas, lanza la auditoría y consulta los resultados obtenidos. Además, se puede lanzar una inferencia de términos seleccionando una auditoría previa e introduciendo nuevas consultas que se compararán contra el comportamiento temporal ya medido en ese target. Su función principal es guiar al usuario y presentar la información de forma entendible, evitando que acciones visuales, refrescos de pantalla o procesos del navegador alteren los tiempos recogidos en las pruebas. La interfaz se comunica con el backend de gestión para cualquier acción realizada.

El backend de gestión actúa como punto central de la aplicación y recibe las acciones realizadas desde la interfaz. Se encarga de registrar los sistemas objetivo, validar los datasets cargados, crear las sesiones de auditoría, registrar las solicitudes de inferencia de términos y consultar los resultados almacenados. Este no se usa para ejecutar las pruebas, ya que estas pueden tardar mucho tiempo y bloquear el funcionamiento normal del sistema, por lo que su papel en la arquitectura se basaría en hacer todo menos la ejecución de las mismas, coordinando el sistema al completo. Por tanto, cuando desde la interfaz se crea una auditoría o una inferencia de términos, el backend de gestión guarda la configuración inicial y envía el trabajo al sistema de colas. Además, este actuará como la capa para consultar los resultados de las pruebas que estarán guardados en la base de datos.

El sistema de colas actúa como intermediario entre el backend de gestión y el sistema de ejecución de auditorías. Su función es almacenar las auditorías y las inferencias de términos

pendientes, entregándolas al *worker* cuando este esté disponible para ejecutarlas. Cuando el backend registra la configuración de una auditoría y la inserta en la cola, el *worker* la recupera y comienza su ejecución. De esta manera, la cola sirve como punto de separación entre ambos módulos y permite que las auditorías se ejecuten de forma ordenada sin bloquear el resto de funcionalidades de la herramienta.

El sistema de ejecución de auditorías trabaja en segundo plano y es el encargado de lanzar las peticiones HTTP contra el RAG objetivo. Aquí se miden los tiempos de respuesta de cada consulta. Por esto se ha separado del backend principal. Si la medición se hiciera dentro del mismo proceso que atiende a la interfaz, una ejecución larga podría afectar al resto de acciones del usuario y también introducir ruido innecesario. Este sistema, también llamado *worker*, recupera las auditorías pendientes en la cola y las ejecuta, mandando las consultas al sistema RAG objetivo y midiendo los tiempos de respuesta.

La base de datos de la aplicación guarda los sistemas objetivo configurados, los datasets de consultas, las auditorías lanzadas y las mediciones obtenidas. Esta persistencia es necesaria porque una auditoría puede generar miles de resultados individuales y no tendría sentido mantenerlos solo en memoria. También permite comparar ejecuciones, revisar resultados antiguos y exportar datos para analizarlos fuera de la herramienta.

El sistema RAG al que se le realiza la auditoría no es una parte de la herramienta, pero es importante aclarar que con este sistema la única comunicación que se establece es por medio de un endpoint HTTP en el que se le envían las consultas del dataset utilizado y nada más.

Por todo esto, la arquitectura propuesta permite separar tres momentos distintos: la configuración de la auditoría, la ejecución de las mediciones y el análisis de los resultados. Esta separación reduce interferencias, facilita repetir pruebas y hace que el sistema pueda adaptarse a distintos entornos RAG sin modificar su lógica principal. En la Fig. 2 se muestra la arquitectura general de la herramienta y la relación entre sus módulos principales, incluyendo la comunicación con el sistema RAG externo auditado.

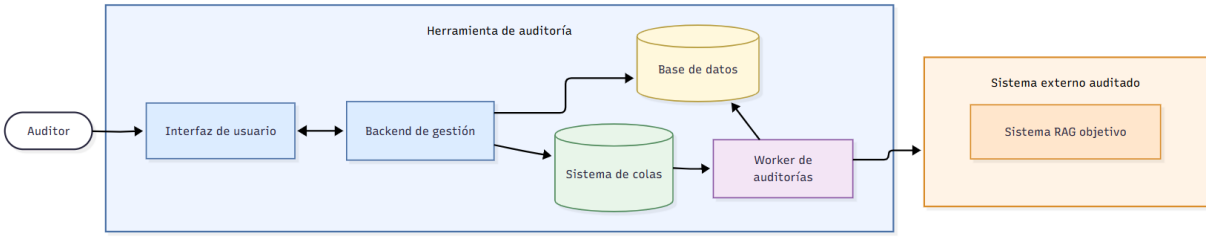


Fig. 2. Arquitectura general de la herramienta de auditoría.

4.3. DISEÑO FUNCIONAL DE LA HERRAMIENTA

El diseño funcional de la herramienta se basa en dos flujos principales: la ejecución de auditorías y la inferencia de términos. Ambos necesitan comunicarse con un sistema RAG externo, por lo que antes de lanzar cualquier prueba es necesario tener un *target* configurado. En el caso de las auditorías, también es necesario disponer de un dataset de consultas validado. El *target* es el sistema externo que se quiere auditar y que se podrá probar en distintas pruebas con diferentes datasets. El dataset contiene las consultas que se enviarán durante la auditoría y que también podrán usarse contra varios *targets*.

Para configurar el *target,* simplemente será necesario configurar la URL del endpoint del chatbot, los headers del mismo y el formato del *body* en el que se debe incluir la variable de las consultas. Para facilitar este proceso, se le entrega al usuario un cuadro de texto donde debe introducir en formato JSON los distintos datos [23]. Con esto conseguimos que, si un sistema espera recibir la pregunta en el campo question, query o message, la herramienta pueda adaptarse perfectamente.

Por otro lado, para la configuración del dataset se requiere un JSON con el formato especificado, ofreciendo al usuario uno de ejemplo para que sepa cómo construirlo. El formato usa las variables consideradas importantes en la experimentación explicada en el apartado de pruebas. Por esto, las consultas del JSON deben venir ordenadas por frecuencia y longitud. De esta forma, la herramienta conserva las etiquetas necesarias para comparar grupos de consultas cuando se analicen los resultados.

La Fig. 3 muestra el flujo funcional de configuración de targets y datasets. En ambos casos, el usuario introduce los datos desde la interfaz, el backend valida el formato y la información queda almacenada para ser usada en auditorías posteriores.

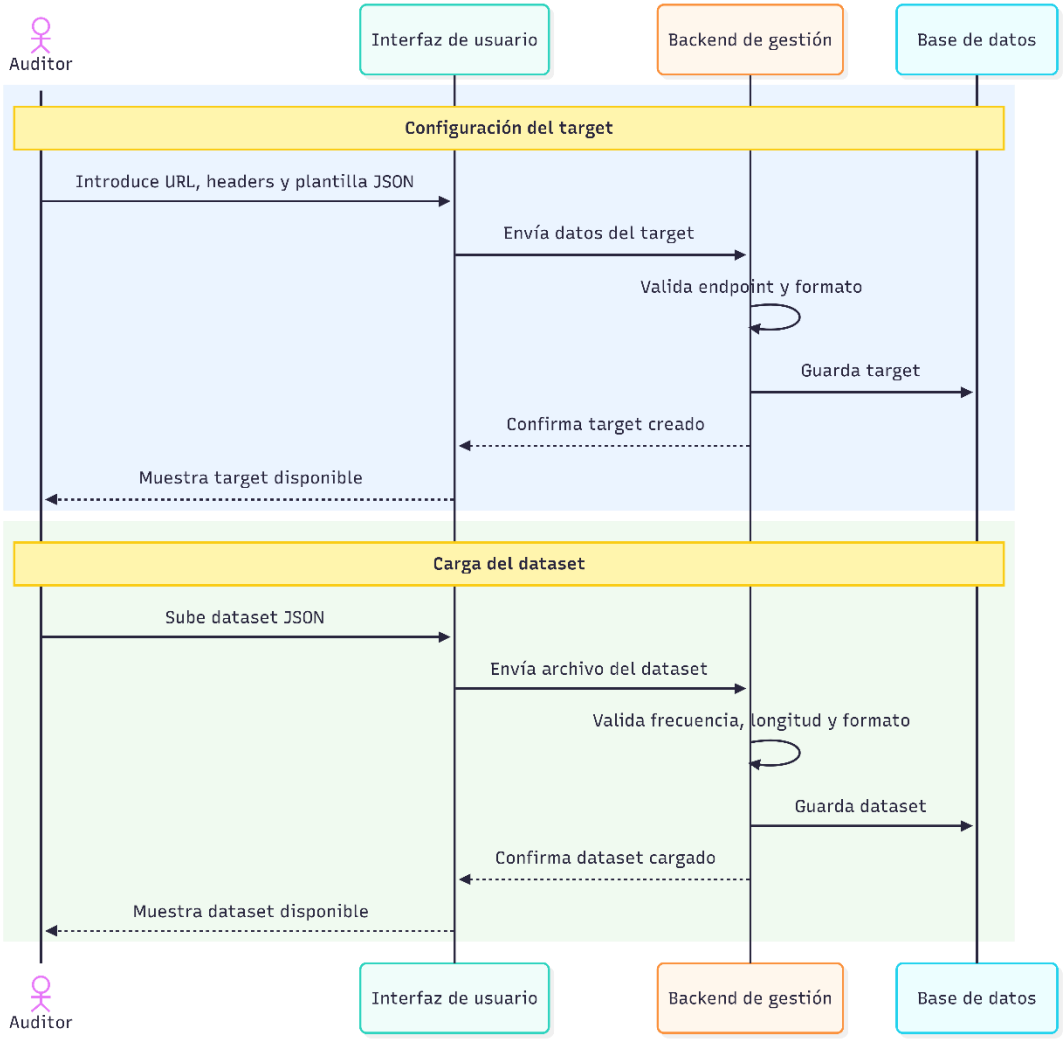


Fig. 3. Flujo funcional de configuración de targets y datasets.

4.3.1. EJECUCIÓN DE AUDITORÍAS

Una vez configurados el *target* y el dataset, el auditor puede lanzar una nueva auditoría seleccionándolos y escogiendo los parámetros de ejecución de la prueba, que son el número de peticiones de calentamiento y la semilla aleatoria (opcional). Esto permite que el usuario pueda repetir una prueba con las mismas condiciones o modificar ciertos valores si necesita comparar el comportamiento del sistema. En ese momento, como se ha explicado en la arquitectura, la herramienta creará una sesión de auditoría con la configuración seleccionada y la dejará registrada en el sistema. Además, esta auditoría será encolada a la espera del worker para su ejecución. Cuando el worker se encuentre liberado, antes de comenzar la ejecución, realizará una pequeña fase de calentamiento explicada más adelante en la experimentación y usada para evitar el ruido generado por la inicialización de algunos procesos. Su objetivo es estabilizar la conexión con el sistema objetivo y evitar que la primera petición de la auditoría altere las mediciones del resto. Después del calentamiento, el worker ejecutará las consultas del dataset. Para evitar que las consultas de una misma categoría se lancen todas juntas, el sistema las baraja mediante una semilla aleatoria, decisión también tomada en la experimentación para evitar problemas con las memorias de los sistemas. Además, la semilla podrá ser elegida por el usuario, asegurándonos de poder mantener la posibilidad de repetir una auditoría con el mismo orden si se necesita revisar el experimento.

Durante la ejecución, cada petición genera un resultado individual, este representa el tiempo desde que se envía la consulta hasta que se recibe el primer byte de respuesta, por lo tanto, incluye los tiempos de envío, vectorización, recuperación de documentos asociados y generación de la respuesta. Cada tiempo medido es guardado por el worker en la base de datos con todos los datos que lo engloban, como la propia consulta, la frecuencia, la longitud, etc. Si una petición falla, el sistema no interrumpe automáticamente toda la auditoría, sino que registra el error y continúa mientras el número de fallos no indique que el target está caído o mal configurado. Esto evita que un error puntual invalide una auditoría completa, pero permite detectar cuándo el entorno objetivo no está respondiendo correctamente.

En la interfaz de usuario se puede ver cómo el estado de la auditoría se va actualizando durante todo el proceso, ya que al no estar bloqueado el backend de gestión, se puede consultar si la auditoría está pendiente, en ejecución, completada, abortada o fallida, además de poder consultar todos los datos en tiempo real y poder ver cómo avanzan los resultados de la propia prueba. Cuando termina la ejecución, la auditoría queda disponible de forma persistente para su análisis, calculando los resultados a partir de las mediciones guardadas en la base de datos.

La Fig. 4 muestra el flujo funcional de una auditoría. En este flujo se parte de un *target* y un dataset ya configurados. A partir de ellos, el usuario crea la auditoría, el backend la guarda, la envía al sistema de colas y el *worker* ejecuta las peticiones contra el sistema externo.

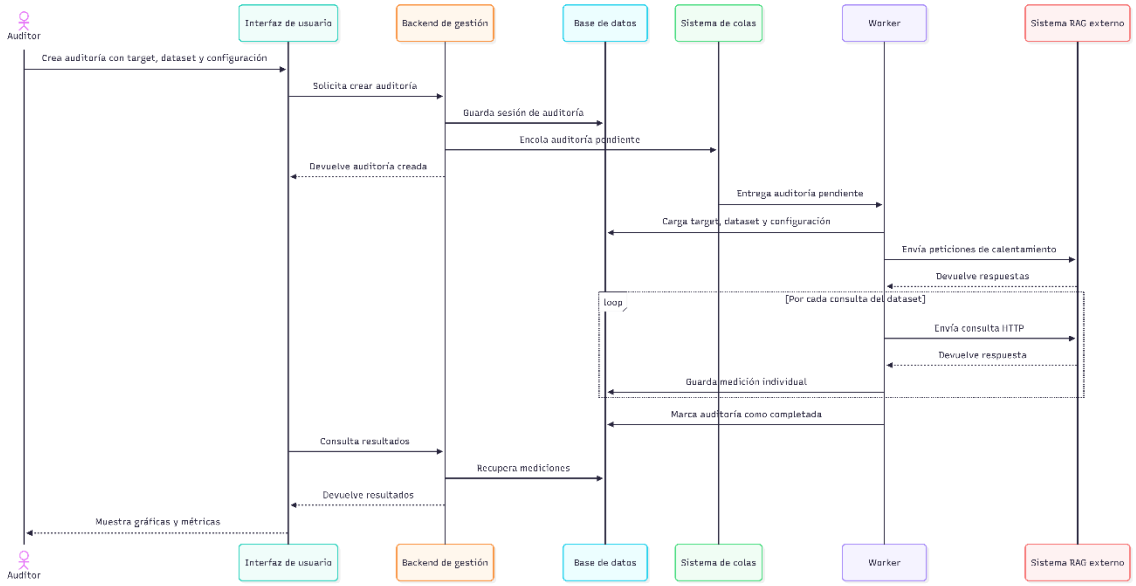


Fig. 4. Flujo funcional de ejecución de una auditoría.

4.3.2. EJECUCIÓN DE INFERENCIA DE TÉRMINOS

La inferencia de términos es el segundo flujo principal de la herramienta. Esta usa una auditoría previa como referencia temporal para comparar consultas centradas en términos y estimar si su comportamiento se parece más al de términos de alta o baja presencia. Por lo tanto, para crear una inferencia de términos, el usuario debe seleccionar el *target* que quiere evaluar y una auditoría previa asociada a ese mismo sistema. En este caso, al usuario se le dan dos opciones, pudiendo elegir una auditoría ya guardada en el sistema o subir un archivo CSV que contenga los valores de una auditoría en las columnas pedidas [24].

Después, el usuario debe seleccionar de qué términos se quiere estudiar la presencia. En este caso se le da al usuario dos opciones. La primera de ellas es simplemente elegir términos a estudiar escribiéndolos en formato JSON. Estos términos se meterán dentro de unas consultas genéricas para poder enviarlas al sistema externo. La otra opción es usada para mayor personalización de la prueba, en la que el usuario escribe directamente en el sistema las consultas ordenadas por términos que quiere probar contra el sistema. En este caso, el usuario debe escribir un número mayor de consultas, ya que serán las únicas que se envíen al sistema, y si usa demasiado pocas, puede generar problemas con la caché. De esta manera, el usuario puede realizar una prueba más específica o más genérica contra el sistema, siempre centrado en algunos términos específicos que quiera probar.

Una vez seleccionadas las consultas a enviar o los términos específicos que se quieren probar, el usuario deberá elegir cuántas pruebas se deben enviar en cada ronda y cuántas rondas se deben hacer como máximo si el sistema no consigue clasificar el término antes. Por último, ya que también se necesita mezclar las consultas de manera aleatoria, el usuario podrá o no elegir la semilla usada para esto para asegurar la reproducibilidad del experimento.

Cuando el usuario crea la inferencia, el backend guarda la solicitud y la envía al sistema de colas. El worker recupera el trabajo pendiente, carga la auditoría usada como calibración y empieza a lanzar las nuevas consultas contra el mismo sistema RAG objetivo. Cada medición queda guardada en la base de datos y asociada al término correspondiente. Una vez recogidos los tiempos, la herramienta compara las nuevas mediciones con las fronteras temporales

obtenidas en la auditoría previa. El objetivo es estimar si el nuevo término se comporta de forma más cercana a las consultas de alta presencia o baja presencia. Este resultado no debe interpretarse como algo definitivo, ya que se basa en la señal de latencia observada y en la estabilidad del sistema auditado.

La Fig. 5 muestra el flujo funcional de la inferencia de términos. En este caso, el proceso depende de una auditoría anterior, ya que sin esa referencia no existe una base temporal con la que comparar los nuevos términos.

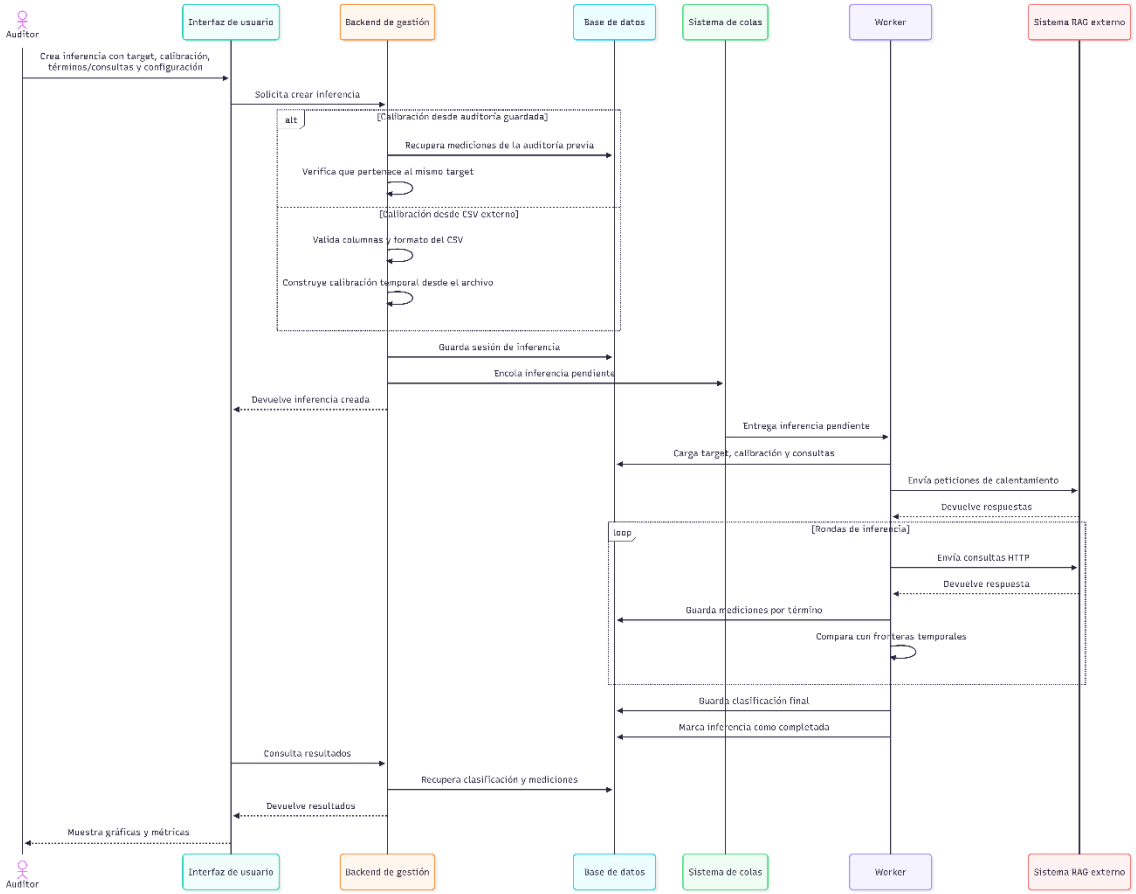


Fig. 5. Flujo funcional de inferencia de términos.

4.4. DISEÑO DEL SISTEMA DE MEDICIÓN Y ANÁLISIS

El sistema de medición y análisis de resultados es muy importante, ya que necesita ser preciso y entregar al usuario todo lo necesario para el análisis en profundidad del ataque. La mayoría de métricas usadas por la herramienta vienen de la fase experimental explicada en el capítulo de pruebas. En ese apartado se justifica por qué se estudian variables como el p-value, la longitud de las consultas, el contenido de las mismas o el efecto de las mediciones repetidas.

4.4.1. MEDICIÓN TEMPORAL Y PROCESAMIENTO ESTADÍSTICO

La métrica principal de la herramienta es el Time-to-First-Byte o TTFB, que es el tiempo que pasa desde que la herramienta envía la consulta hasta que recibe el primer byte de respuesta del sistema auditado. Esta métrica ha sido elegida por encima del tiempo total de respuesta, intentando reducir una parte del ruido de red. Pero, como se ha mencionado anteriormente, esta

medición sigue incluyendo tiempos como el de la vectorización de la consulta, la recuperación de documentos o la generación de la respuesta, entre otros. Esto hace que la medida tenga ruido, pero también la convierte en una métrica realista, ya que es observable desde fuera y no requiere acceso interno al sistema RAG.

Los resultados temporales generados por cada consulta se guardan individualmente junto a datos como la consulta enviada, la frecuencia, la longitud, el target, el estado HTTP, los tiempos medidos y si se ha producido un error. Esta decisión evita que la herramienta dependa solo de medias finales, ya que al conservar cada medición, se puede revisar la distribución completa y detectar picos o valores extraños. En el caso de la inferencia de términos, las mediciones se guardan también asociadas al término que se está evaluando, lo que permite saber qué tiempos pertenecen a cada término y compararlos después con las fronteras temporales obtenidas en la auditoría previa.

Con estos resultados, una vez terminada la auditoría, la herramienta agrupa las mediciones según las etiquetas del dataset para poder comparar las consultas de alta frecuencia frente a las de baja frecuencia, o las consultas cortas frente a las largas. La herramienta calcula métricas como la media, la mediana, la desviación típica y los percentiles para entender cómo se comporta cada grupo. El uso de percentiles permite observar si existen valores extremos que puedan alterar la interpretación de los datos, ya que estos se pueden generar por razones varias, como sobrecargas momentáneas del sistema atacado. Además, para generar una comparación estadística de diferentes grupos, la herramienta calcula el p-value asociado a una prueba estadística, el cual permite estudiar si la diferencia observada entre dos distribuciones tiene suficiente peso o si pudiera explicarse por ruido normal del sistema [22]. El sistema muestra estos resultados de forma agregada para que el auditor pueda interpretar la prueba sin revisar manualmente cada petición. Si el p-value es bajo y las distribuciones presentan separación, la evidencia temporal será más fuerte. Si los grupos se solapan o el p-value es alto, la herramienta mostrará que la señal obtenida es débil o insuficiente.

4.4.2. ANÁLISIS PARA INFERENCIA DE TÉRMINOS

La inferencia de términos se apoya en los resultados de una auditoría anterior, ya sea mediante una guardada en el sistema o un archivo CSV con los datos necesarios, la cual funciona como calibración, ya que contiene las distribuciones temporales de referencia para un target concreto. A partir de esas distribuciones, la herramienta construye unas fronteras temporales que permiten comparar nuevas consultas. Para construir estas fronteras, la herramienta toma como referencia las mediciones de la auditoría previa y calcula una separación entre los grupos de alta y baja presencia. Esta separación funciona como umbral temporal para comparar las nuevas mediciones. También se contempla una zona intermedia en la que la clasificación no se considera suficientemente clara, evitando forzar una decisión cuando los tiempos están demasiado cerca de la frontera.

El objetivo de este flujo es probar la frecuencia de algunos términos en el sistema de una forma más rápida que la realización de una auditoría completa. Para ello, el usuario introduce los términos o consultas que quiere estudiar y la herramienta lanza nuevas mediciones contra el mismo sistema RAG. Cada término se mide varias veces para intentar evitar el ruido al máximo, además, el valor medido es el Time-to-First-Byte, igual que en la auditoría. Con estas mediciones, la herramienta compara cada término con la calibración previa. Si la mediana de las nuevas mediciones se acerca más al comportamiento de las consultas de alta presencia, el término se clasificará como probablemente presente. Si se acercan más al comportamiento de

baja presencia, se clasificará como término probablemente ausente. Cuando los tiempos quedan entre ambas zonas o no existe separación suficiente, la clasificación del término es no concluyente.

Este análisis depende mucho de la calidad de la auditoría usada como referencia. Si la auditoría previa no muestra diferencias claras entre grupos, la inferencia de términos tendrá menor valor. También depende de que el target se mantenga estable entre la auditoría y la inferencia. Si el sistema modifica distintas cosas, como su carga o su caché, las fronteras temporales pueden dejar de representar correctamente el comportamiento del entorno. Por este motivo, el resultado de la inferencia se plantea como una estimación basada en las mediciones obtenidas y en la comparación con la auditoría previa. Su valor está en permitir que el auditor pruebe nuevos términos sin tener que reconstruir todo el dataset inicial, manteniendo una base estadística para interpretar los tiempos.

4.5. DISEÑO DE PERSISTENCIA Y RESULTADOS

El diseño de una base de datos bien estructurada y optimizada ayuda a mejorar el funcionamiento de cualquier sistema. Con este se toman decisiones tan importantes como qué datos deben tener persistencia y cuáles no, o de qué forma deben estructurarse los datos para que su accesibilidad, guardado, modificación y eliminación sean óptimas. En este caso, la herramienta necesita guardar la información generada en cada prueba de forma optimizada, ya que tanto las auditorías como las inferencias de términos pueden producir muchas mediciones individuales. El diseño incluye una base de datos con diferentes tablas en las que se almacenan los elementos principales del sistema: usuarios, targets, datasets, auditorías, mediciones, inferencias y resultados asociados. De este modo, la persistencia permite que el usuario no tenga que repetir una acción varias veces si apaga el sistema, permitiendo usar datasets o targets insertados con anterioridad y consultar pruebas anteriores.

Junto a la base de datos principal, la herramienta cuenta con un sistema de persistencia temporal asociado a la cola de trabajos. Este componente se encarga de almacenar auditorías e inferencias pendientes hasta que el worker pueda procesarlas. Su función no es conservar resultados finales ni datos históricos, sino coordinar la ejecución asíncrona de las pruebas. Por este motivo, no aparece como una entidad del modelo relacional, ya que los datos importantes de cada auditoría se guardan finalmente en las tablas de sesiones, resultados y mediciones.

La tabla de usuarios en el diseño actual no se relaciona directamente con el resto de entidades del modelo, ya que esta herramienta está pensada para el despliegue en un solo ordenador, no es necesario comprobar roles al solo existir el usuario administrador. Igualmente, esta tabla tiene sentido porque permite que el sistema tenga autenticación y evita que cualquier persona pueda usarlo si se deja funcionando en un ordenador. Esta autenticación se implementa de forma profesional con el uso de cookies almacenadas en el navegador que caducan con el tiempo y que se comprueban en cada llamada al sistema, protegiendo el uso de las API de usuarios no autenticados. Además, por seguridad, se siguen reglas básicas como realizar el hash de las contraseñas para guardarlas, ya que, aunque ahora mismo la herramienta solo tiene el usuario administrador, así se evita almacenar credenciales en texto plano y se deja preparada la herramienta para una futura gestión de usuarios.

Para la persistencia de targets, se guarda la información necesaria para comunicarse con el sistema externo, que en este caso es: el endpoint, los headers, la plantilla de petición, el timeout de la comunicación y la necesidad de verificar o no la comunicación TLS [17]. Además de esta

información estrictamente necesaria para que el sistema se comunique con el endpoint externo, se guardan otros datos para facilidad del sistema, como un nombre y un id asociado al target. Esta tabla hace posible que un mismo entorno pueda probarse con distintos datasets o que se puedan repetir pruebas en momentos diferentes, ya que el target no queda ligado a una única auditoría ni a un único dataset.

En cuanto a los datasets, también se guardan de forma independiente para conseguir el mismo comportamiento, en el que un dataset se pueda usar en varias auditorías y contra distintos targets. Al mantenerlo separado de la auditoría, la herramienta permite comparar cómo se comporta una misma batería de consultas en varios sistemas RAG. También permite revisar qué consultas forman parte de la prueba y qué etiquetas tiene cada una. En este caso, se ha decidido que los archivos de datasets se guarden como archivos en vez de guardar fila por fila el contenido en la base de datos; con esto simplificamos el proceso, solo necesitando guardar la ruta donde el archivo queda almacenado en el sistema una vez este haya sido comprobado y se verifique que tiene un formato correcto. Junto a esta ruta se guarda también el nombre original del archivo subido que facilita reconocer el fichero usado por el usuario. Además, guardaremos el ID y el número de queries del archivo para poder acceder a esos datos de forma más rápida y usarlos en la interfaz de usuario para una visualización rápida del contenido del dataset.

Las auditorías necesitan un funcionamiento algo más complejo, ya que estas necesitan varias relaciones con otras tablas en la base de datos. Además, cada auditoría genera n resultados, por lo que es importante separar la sesión de auditoría y sus resultados en dos tablas para no saturar la tabla de las sesiones, que es la que utilizaremos para la visualización rápida de los datos.
Cuando se crea una auditoría, el sistema guarda una sesión propia con su configuración que contiene: el ID del target seleccionado, el ID del dataset usado, el status actual de la sesión que al inicio será pending, ya que simplemente quedará encolado a la espera del worker, la seed aleatoria usada, el número de consultas de calibración, el progreso actual y el progreso total.

Durante la prueba, el worker va registrando cada medición individual en la base de datos. Estas se guardan en la tabla de resultados de las auditorías y van asociadas a la sesión de auditoría en la que se realizan. Para poder interpretar correctamente los resultados, se guardan los siguientes datos junto al Time-to-First-Byte, que es el dato principal de medición: índice de consulta, consulta enviada, frecuencia, longitud, código HTTP, tamaño de la respuesta, si la petición ha producido error, tipo de error, mensaje de error y fecha exacta de la medición y el tiempo completo de respuesta que se guarda y muestra para analizar si ha habido mucho ruido de red. Es importante guardar cada medición de forma individual porque el análisis de los datos muestra una distribución de los tiempos obtenidos.

En el caso de la inferencia de términos, el sistema también guarda la sesión de inferencia por separado de sus resultados por la misma explicación que en la auditoría. En este caso, para guardar la sesión, necesitamos: el target usado, la auditoría o CSV de calibración, el tipo de fuente, los términos o consultas evaluados que se guardarán en el mismo campo normalizado, número de pruebas por término y el máximo de intentos. Si la calibración viene de una auditoría del sistema, se conserva su identificador. Si viene de un CSV externo, ese identificador queda vacío y la referencia se obtiene del propio archivo cargado. La sesión también guarda el perfil de calibración generado, el progreso de ejecución, la semilla aleatoria y los errores.

Para guardar los términos de la inferencia, en este caso necesitamos hacerlo un poco más complejo para mejorar la optimización, ya que en este caso cada término probado puede tener

n pruebas, por lo que necesitamos separar los resultados entre una tabla para los términos que irán unidos a la sesión en la que se prueban y otra para guardar todas las mediciones asociadas, cada una al id del término que se ha probado. En la tabla de resultados se guarda el ID de la sesión, el término que se prueba, su clasificación, que se irá actualizando a medida que avance la prueba, la media del Time-to-First-Byte, si el término actúa como control, la desviación típica observada, el número de mediciones válidas, el total de mediciones realizadas, la distancia respecto a la frontera temporal y la referencia más cercana usada para clasificarlo. Estos campos son importantes porque permiten entender por qué un término ha sido clasificado de una forma concreta y si esa clasificación se apoya en suficientes mediciones o está afectada por errores.

En la tabla de mediciones se guarda, junto al ID del resultado asociado, el índice de la prueba, el número de ronda, la consulta enviada, la latencia medida, el Time-to-First-Byte, el tiempo completo de respuesta, el código HTTP, el tamaño de la respuesta, si ha habido error, el tipo y mensaje del error y la fecha de la medición.

La relación entre tablas permite mantener separado cada nivel de información. Los targets y datasets se relacionan con las sesiones de auditoría, mientras que cada auditoría puede generar muchas mediciones individuales. Por otro lado, las sesiones de inferencia se relacionan con un target y, cuando procede, con una auditoría previa usada como calibración. Cada sesión de inferencia contiene varios resultados, uno por término evaluado, y cada resultado puede tener varias mediciones asociadas. La tabla de usuarios queda separada del resto del modelo porque actualmente se usa para autenticación y gestión de acceso, pero no participa directamente en la ejecución de auditorías ni inferencias. En la Fig. 6 se muestra el modelo de datos completo de la herramienta, incluyendo las tablas, sus columnas y las relaciones entre ellas.

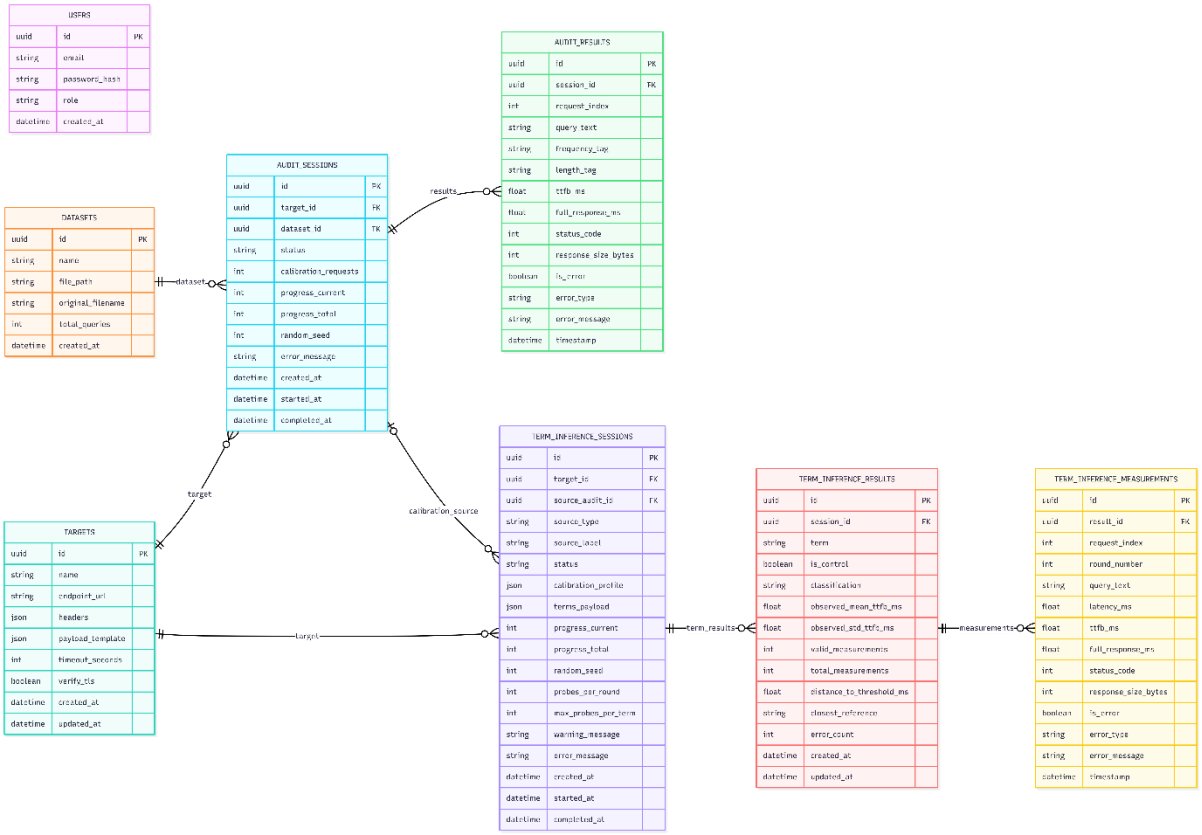


Fig. 6. Modelo de datos de la herramienta de auditoría.

4.6. DISEÑO DE LA INTERFAZ DE USUARIO

El principal propósito de esta herramienta es realizar el ataque investigado de una forma controlada y sin necesitar usar scripts o una terminal. Por esto mismo es necesario que la herramienta separe las principales acciones en distintas pantallas y así el usuario tenga claro dónde se debe realizar cada cosa. Además, la herramienta debe tener una estética profesional y empresarial que deje claras las acciones y utilidades de cada parte, incluyendo explicaciones en las partes que no se entiendan con facilidad por su uso concreto en la aplicación. Los colores escogidos para representar este estilo son el blanco y negro y el rojo para mostrar errores o botones de acciones no reversibles.

La primera pantalla de cualquier herramienta que implemente un mínimo de seguridad es una pantalla para la autenticación. En este caso, como se ha comentado antes, no se implementa un sistema de gestión de usuarios, ya que se cree innecesario para el uso esperado de la herramienta. Por esto, no hay ninguna pantalla enfocada a la creación, eliminación o modificación de usuarios y la única pantalla enfocada a la autenticación es la pantalla del login. Con este login se consigue una seguridad que protege todos los datos sensibles que el auditor haya podido meter en el sistema, como el endpoint y headers del chatbot empresarial o las pruebas ya auditadas.

Una vez dentro de la herramienta, la primera pantalla que se le muestra al usuario es el panel de las auditorías generadas. En esta, lo primero que se le expone al usuario es el número de auditorías en cada estado, como las completadas o fallidas. Además, se le muestra al usuario cada auditoría por separado con información relevante como el progreso, la seed utilizada, la media o el número de consultas de calentamiento. Cada auditoría se podrá abrir mostrando otra pantalla distinta con sus resultados y análisis estadístico, en la que se mostrarán muchos más datos que facilitarán el análisis de la prueba, como gráficas o tablas de datos ordenadas. Por otro lado, en la pantalla de auditorías también hay un botón para crearlas que abrirá otra pantalla distinta que tiene un formulario en el que el usuario tendrá que rellenar todos los datos para poder comenzar la prueba. En este caso, los datos necesarios no son demasiados, ya que el usuario solo tendrá que introducir el target y el dataset, los cuales debe haber insertado con anterioridad en la herramienta, además del número de consultas de calentamiento que quiere que lance el sistema y, si quiere, puede introducir una semilla o dejar que la genere el sistema.

Para insertar el target, el usuario podrá navegar por la barra lateral a la página de targets, la cual le mostrará un formulario con los datos necesarios para crear un target nuevo, que son: el nombre, la URL del endpoint, el timeout establecido, los headers necesarios y el body que necesita enviar. Además, debajo en la misma página se mostrarán los targets que ya están creados, mostrando información como la URL y dando la posibilidad de editarlos, probarlos o eliminarlos.

En cuanto a la creación de datasets, el usuario debe navegar hasta la pantalla destinada para ello, en la que podrá subir un archivo JSON con las consultas clasificadas por frecuencia y longitud. Una vez subido el archivo, aparecerá con su nombre en la parte de debajo de esa pantalla, haciendo posible eliminarlo o analizarlo con los botones asignados. En el análisis o preview del documento, el sistema mostrará información relevante como el número de consultas y cuántas hay de cada grupo, que será útil para que el usuario observe que el archivo ha sido procesado con éxito.

La inferencia de términos cuenta con una pantalla propia en la que el usuario, de manera similar a las auditorías, podrá ver los datos de los estados, así como las pruebas ya realizadas con información básica como el número de términos probados o la semilla utilizada. Además, de nuevo se le da al usuario la posibilidad de ver los resultados detallados de cada prueba realizada con un botón. Por otro lado, el usuario también podrá crear nuevas pruebas de inferencia. Para esto se le mostrará al usuario un formulario que tendrá que rellenar para poder iniciar la prueba. En este caso, el usuario tendrá que seleccionar el target, el tipo de calibración; si es por medio de una auditoría, además deberá elegir cuál y, si es por medio de un archivo, el usuario deberá subir su CSV; también deberá rellenar los términos o las consultas que se quieren probar, el número de pruebas por ronda, el número de rondas y, si se quiere, se podrá introducir una semilla.

La visualización de resultados es una parte muy importante de la interfaz de usuario, ya que esta es la que le da la posibilidad al usuario de saber si la prueba ha sido exitosa o no. Por esto es importante que esté bien ordenada y limpia. En el caso de las auditorías, la interfaz muestra métricas como la mediana y la media, gráficas de distribución, boxplot y gráfica de barras; además, se utilizan comparaciones estadísticas entre grupos utilizando el p-value como medida principal de comparación, al ayudar a interpretar si las diferencias observadas tienen peso estadístico. Las mediciones individuales siguen estando disponibles, pero se quedan en un segundo plano, ya que se le ofrece un análisis completo al usuario. Por otro lado, en la página de resultados de la inferencia de términos, los resultados se muestran centrados en cada término evaluado. Para cada uno se presenta su clasificación, el número de mediciones válidas, la media observada del Time-to-First-Byte y la referencia temporal con la que se ha comparado. Además, se le ofrece al usuario dos gráficas para el análisis. La primera, en la que se pueden observar las fronteras de la prueba y donde se sitúa la media utilizada de cada término, y la segunda, que muestra el avance de la prueba y le dice al usuario qué términos se han clasificado en cada momento de la prueba y cómo esta ha ido avanzando. Con esta página de resultados le permitimos al usuario determinar de una manera rápida si un término se acerca más al comportamiento de alta presencia, baja presencia o si queda en una zona no concluyente. La interfaz también contempla la exportación de resultados en formato CSV para que el auditor conserve las mediciones fuera de la herramienta.

Por otro lado, en este tipo de herramientas sujetas a muchos cambios por utilizar sistemas externos, es muy importante la gestión de errores. Si una petición falla, si un target no responde, si el RAG externo se ralentiza en medio de una prueba, si un archivo no tiene el formato esperado o si se obtiene un error inesperado por cualquier problema, la interfaz debe mostrar el problema de forma clara. Esto es importante porque en una herramienta de este estilo los errores hacen que los análisis fallen, por lo que hay que guardarlos y no tenerlos en cuenta para las mediciones o avisar al usuario de ellos para que este entienda lo que ha ocurrido.

El diseño de la interfaz sigue la misma separación que la arquitectura interna de la herramienta. La interfaz permite configurar, lanzar pruebas y visualizar resultados, pero no ejecuta directamente las mediciones. Esa responsabilidad se mantiene en el worker para evitar que acciones visuales o procesos del navegador afecten a los tiempos recogidos. De esta forma, la interfaz actúa como capa de control y revisión, mientras que la medición se mantiene separada del uso normal de la aplicación.

Las pantallas de la interfaz se incluyen en el Anexo A, donde se muestran las vistas de login, gestión de targets, carga de datasets, panel de auditorías, creación de auditoría, resultados de auditoría, panel de inferencia de términos, creación de inferencia de términos y resultados de

inferencia. Estas capturas permiten ver cómo se materializa visualmente el diseño descrito en este apartado.

4.7. DECISIONES DE DISEÑO Y LIMITACIONES

El diseño de la herramienta se ha tomado buscando que el ataque pueda ejecutarse sobre sistemas RAG externos sin depender de su arquitectura interna. Por esto, la decisión principal ha sido trabajar mediante una comunicación HTTP de caja negra. Esta decisión hace que la herramienta sea más flexible, pero también introduce limitaciones. Al no tener acceso interno al sistema RAG, no puede separar con precisión qué parte del tiempo corresponde a la vectorización, a la recuperación documental, a la generación del modelo o al ruido de red. La métrica principal, el Time-to-First-Byte, recoge todo ese comportamiento inicial como una única señal temporal. Esto encaja con el modelo de amenaza, pero obliga a interpretar los resultados con cuidado.

Otra decisión importante es separar la ejecución de pruebas del backend principal. Las auditorías y las inferencias pueden tardar bastante tiempo, por lo que se ejecutan mediante un worker en segundo plano. Esta separación evita bloquear la interfaz y reduce interferencias producidas por la propia aplicación. También permite que el backend se centre en coordinar el sistema, guardar configuraciones y consultar resultados.

Por otro lado, se ha decidido que los datasets solo sean válidos si mantienen el formato con las variables definidas en la investigación. Desarrollar una manera para identificar JSON con distintos formatos se ha tomado como innecesario, ya que si se le entrega un JSON de ejemplo al usuario, este solo necesitará rellenarlo respetando la estructura.

En cuanto a las limitaciones, la principal se encuentra en la necesidad de un sistema estable para la realización de una prueba de inferencia de términos. La idea de la inferencia de términos es hacer algo parecido a una auditoría, pero centrado en unos términos específicos y de forma más rápida. Por esto, esta prueba es muy sensible a cambios en la carga del servidor o interferencias en la red.

Otra limitación aparece en la interpretación del Time-to-First-Byte. Aunque esta métrica reduce el tiempo de red, no permite separar internamente qué parte del tiempo pertenece a la red, al servidor, al retriever o al modelo de lenguaje. Por tanto, los resultados deben interpretarse como una señal observable de caja negra y no como una medición aislada del algoritmo de recuperación.

En el caso de la inferencia mediante CSV externo, la herramienta depende de que el archivo represente correctamente una auditoría realizada sobre el mismo target o sobre un entorno equivalente. Si el CSV procede de otro sistema o de una ejecución no comparable, las fronteras temporales generadas pueden producir clasificaciones erróneas.

Por último, la herramienta está pensada para auditorías autorizadas sobre sistemas propios o entornos controlados. Su objetivo es ayudar a detectar si una arquitectura RAG puede filtrar información mediante latencias, siempre dentro de auditorías autorizadas y con permiso del propietario del sistema evaluado. Esta limitación es importante porque el valor del trabajo está en convertir una vulnerabilidad experimental en un proceso de evaluación defensiva.

5. IMPLEMENTACIÓN DE LA HERRAMIENTA

Una vez definido el diseño de la herramienta, este capítulo describe cómo se ha construido la aplicación desarrollada. El objetivo de este apartado es mostrar cómo se ha llevado a código cada una de las partes principales del sistema. Para ello, se explicarán las partes más importantes que son: las tecnologías usadas, la organización del proyecto, el backend de gestión, el procesamiento en segundo plano, el módulo de medición temporal, el análisis estadístico, la inferencia de términos y la interfaz de usuario.

5.1. ENTORNO TECNOLÓGICO UTILIZADO

La herramienta se ha desarrollado con una arquitectura web separada entre backend, frontend, base de datos y sistema de trabajos en segundo plano. Esta separación permite que cada parte del sistema tenga una responsabilidad clara y que la ejecución de las pruebas no interfiera con el sistema de gestión y viceversa.

Python se ha usado como lenguaje principal del backend por su integración con librerías de análisis de datos, medición HTTP y procesamiento asíncrono, esto facilita conectar la parte de aplicación con los módulos estadísticos de la herramienta sin tener que separar el análisis en otro entorno. En concreto, el backend usa FastAPI, que es un framework web para construir API en Python, pensado para trabajar con peticiones HTTP, validación de datos y operaciones asíncronas. Esta elección encaja bien con el proyecto porque permite construir una API HTTP de forma sencilla, trabajar con programación asíncrona y definir contratos de entrada y salida mediante esquemas validados. En esta herramienta, FastAPI actúa como punto de entrada de todas las acciones realizadas desde la interfaz: login, gestión de targets, carga de datasets, creación de auditorías, creación de inferencias y consulta de resultados. Para la validación de datos se usa Pydantic, librería de validación que permite definir estructuras de datos con tipos concretos y comprobar automáticamente si la información recibida cumple ese formato. Los esquemas definidos con esta librería permiten comprobar que los datos enviados al backend tienen el formato esperado antes de guardarlos o enviarlos a una prueba. Esto es especialmente importante en la creación de targets, donde la plantilla JSON debe contener la variable de consulta, y en la carga de datasets, donde el archivo debe respetar la estructura de frecuencia y longitud definida en la investigación.

PostgreSQL es una base de datos relacional, adecuada para este caso porque la herramienta trabaja con entidades conectadas entre sí, como targets, datasets, auditorías, mediciones e inferencias. Por esto, la persistencia se realiza con PostgreSQL, que guarda los usuarios, targets, datasets, sesiones de auditoría, mediciones, sesiones de inferencia y resultados asociados. Además, el acceso a la base de datos se implementa mediante SQLAlchemy en modo asíncrono, que actúa como capa de acceso entre el código Python y la base de datos, permitiendo representar las tablas mediante modelos y realizar consultas sin escribir directamente SQL en cada operación, lo que permite trabajar con modelos relacionales desde el backend sin bloquear el flujo normal de la aplicación. Para gestionar los cambios de estructura de la base de datos durante el desarrollo de la aplicación, se ha usado Alembic, herramienta de migraciones que registra los cambios en el modelo de datos y permite aplicarlos de forma controlada sobre la base de datos, manteniendo las migraciones separadas del código de la aplicación.

El sistema de colas y ejecución en segundo plano se ha implementado con Redis y Arq. Redis es una base de datos en memoria que permite guardar información temporal con tiempos de acceso muy bajos, en esta herramienta no se usa como base de datos principal, sino como soporte de la cola de trabajos. Por otro lado, Arq es una librería de tareas asíncronas para Python que trabaja sobre Redis y permite definir funciones que se ejecutan fuera del proceso principal del backend. Por lo que Redis actúa como soporte temporal de la cola, mientras que Arq permite definir trabajos asíncronos que son recogidos por el worker. Gracias a esto, cuando el usuario lanza una auditoría o una inferencia de términos, el backend no ejecuta la prueba directamente, sino que primero guarda la sesión y después encola el trabajo. El worker lo recoge cuando está disponible y se encarga de enviar las peticiones al sistema RAG externo. Para realizar las peticiones HTTP al target auditado se usa httpx, cliente HTTP para Python que permite enviar peticiones de forma asíncrona. Esta librería permite crear clientes HTTP asíncronos y trabajar con respuestas en streaming, lo cual es necesario para medir el Time-to-First-Byte. En la herramienta, httpx se usa tanto para probar la conexión de un target como para ejecutar las consultas reales de auditorías e inferencias.

El análisis de resultados se apoya en librerías de cálculo numérico y estadístico. NumPy es una librería orientada al cálculo numérico y al trabajo con conjuntos de datos en forma de arrays. SciPy amplía esa base con funciones científicas y estadísticas. En este caso, NumPy es usado para calcular métricas como medias, medianas, desviaciones y percentiles, y SciPy es usado para realizar comparaciones estadísticas entre grupos y obtener el p-value [25], [26].

El frontend se ha construido con React y TypeScript. React es una librería de JavaScript para construir interfaces a partir de componentes reutilizables, mientras que TypeScript es una extensión de JavaScript que añade tipado estático, lo que ayuda a detectar errores durante el desarrollo y a mantener más claros los datos que se intercambian con el backend. En la herramienta, esta combinación permite dividir la interfaz en pantallas y componentes asociados a auditorías, targets, datasets, inferencias y resultados. Para la construcción del proyecto se usa Vite, herramienta de desarrollo y construcción para aplicaciones web modernas que se usa para levantar el frontend en local y generar la versión preparada para ejecución. Por otro lado, para el estilo visual se usa Tailwind CSS, framework de estilos basado en clases de utilidad, lo que permite definir la apariencia de los componentes directamente desde la interfaz sin crear hojas de estilo grandes para cada pantalla. En cuanto a las llamadas al backend, estas se gestionan desde un cliente centralizado, y la carga de datos en pantalla se apoya en TanStack Query, que es una librería centrada en la gestión del estado que viene del servidor. A diferencia del estado local de una pantalla, este tipo de estado depende de peticiones HTTP, puede tardar en cargarse, puede fallar y puede quedarse desactualizado. Por esto, la librería ofrece mecanismos para consultar datos, almacenarlos temporalmente en caché, refrescarlos y controlar estados de carga o error sin tener que programar toda esa lógica manualmente en cada componente. La herramienta TanStack Query se usa para consultar auditorías, targets, datasets, inferencias y resultados, lo que permite que la interfaz actualice la información de forma ordenada, especialmente en pantallas donde el estado puede cambiar mientras el worker ejecuta una prueba. Por último, para las gráficas de resultados del frontend se usa ECharts, librería de visualización que permite construir gráficas interactivas en el navegador que permiten representar distribuciones, comparaciones y evolución temporal de las mediciones.

Para la ejecución del sistema se usa Docker Compose, ya que permite definir varios servicios en un único archivo de configuración y arrancarlos de forma conjunta. Esto resulta útil en este proyecto porque la herramienta no funciona con un único proceso, sino con varios componentes que deben estar levantados al mismo tiempo. En concreto, Docker Compose levanta

PostgreSQL, Redis, el backend, el worker, el frontend y un mock local de RAG para pruebas de desarrollo. En esta configuración, el backend y el worker comparten el mismo código base de la aplicación, pero se ejecutan con comandos distintos. El contenedor del backend arranca la API de FastAPI y queda preparado para recibir peticiones desde el frontend. En cambio, el contenedor del worker no expone una API para el usuario, sino que inicia el proceso de Arq y queda escuchando trabajos pendientes en Redis. Por esto, aunque ambos contenedores parten de la misma aplicación, su responsabilidad durante la ejecución es diferente.

El mock local de RAG no forma parte del núcleo de la herramienta, pero permite comprobar el flujo completo sin depender siempre de un sistema externo real. Su uso se limita al desarrollo y a pruebas controladas. El entorno RAG usado para las pruebas más completas se trata en el capítulo de evaluación experimental.

5.2. ORGANIZACIÓN GENERAL DEL PROYECTO

Como ya se ha mencionado, el proyecto se organiza separando la aplicación en dos partes principales: backend y frontend. Esta separación también es visible en la organización de carpetas en el código fuente de la herramienta. La carpeta del backend contiene todo lo necesario para validar datos, acceder a la base de datos, crear auditorías, encolar trabajos, ejecutar mediciones y calcular resultados, y el contenido de la carpeta del frontend se centra en mostrar las pantallas de la herramienta, recoger la información introducida por el usuario, comunicarse con el backend y representar los resultados devueltos por la API. El código fuente completo de la herramienta se encuentra disponible en el repositorio público del proyecto [27].

La carpeta del backend primero tiene una división entre la carpeta de migraciones de Alembic y la app. En esta última se encuentra la aplicación desarrollada con FastAPI y, a su vez, esta está separada en varios módulos agrupados por responsabilidad o rol técnico de cada archivo. Este tipo de separación es muy usada en las aplicaciones, ya que ordena los proyectos por funcionalidades, lo cual hace que las mejoras a futuro no sean demasiado complicadas al tener cada cosa agrupada por lo que hace. Los módulos en los que se ha dividido la app en este caso son: la API, los modelos de base de datos, los esquemas de validación, los servicios de la app, la configuración y funciones de seguridad, y los workers que trabajan en segundo plano. Esta estructura no se centra en la cantidad de código de una sección, sino en la responsabilidad que tiene para agruparlo junto a otros fragmentos, evitando separar funcionalidades del mismo tipo por decisiones externas como el número de líneas.

En la carpeta API encontramos una separación de archivos realizada de la misma forma, obteniendo cinco archivos distintos que controlan las distintas responsabilidades de la API de la herramienta. Los archivos son: autenticación, targets, datasets, auditorías e inferencia de términos, cada uno con todas las rutas y funcionalidades específicas del elemento al que hace referencia su nombre. De esta forma, de nuevo conseguimos que el código sea más modular y fácil de modificar, al solo tener que pensar qué hace la pieza de código que queremos cambiar para encontrar el archivo específico.

Los modelos de base de datos representan las entidades persistentes de la herramienta y se separan en: usuarios, targets, datasets, auditoría e inferencia de términos. En cada archivo no solo encontramos un modelo de datos, sino todos los necesarios asociados a cada elemento; por ejemplo, en el archivo de auditorías, encontramos tanto el modelo que hace referencia a la tabla de sesiones como el que hace referencia a la tabla de los resultados. Como se ha explicado

en las tecnologías utilizadas, estos modelos permiten que el backend trabaje con la base de datos usando objetos de Python.

En cuanto a los esquemas de validación, estos se usan para definir qué información puede entrar y salir de la API, lo cual es útil, ya que muchas operaciones reciben datos estructurados que deben cumplir unas reglas concretas; por ejemplo, un target necesita una URL, una plantilla de petición y unos parámetros de comunicación. Al definir estos contratos mediante esquemas, el backend puede rechazar peticiones incompletas o mal formadas antes de que lleguen a la lógica principal. En este caso se dividen de la misma forma que la API, que es: autenticación, targets, datasets, auditoría e inferencia de términos, ya que tienen que validar las comunicaciones con todas las API. De nuevo, en cada archivo tenemos todos los esquemas necesarios para todas las API de cada responsabilidad específica, encontrando modelos de datos como un modelo específico para leer un dataset en la pantalla general o uno para realizar su análisis en profundidad, ya que los datos devueltos por la API son distintos en cada caso.

En la carpeta de servicios encontramos todas las lógicas de la herramienta. Todos los procesos que tiene que realizar la aplicación, como la inyección de consultas en las plantillas de los cuerpos de petición de los targets, la validación de los datasets, los procesos previos a la inferencia como la creación de un perfil de calibración, la ejecución de las distintas sesiones, la medición temporal de las llamadas de las pruebas y el análisis estadístico, están en esta carpeta. Estos procesos se separan en distintos archivos y se diferencian de las llamadas a la API. De nuevo, el código queda más ordenado y cada módulo puede modificarse sin afectar directamente al resto de la API.

El worker se encuentra en una carpeta distinta del resto, ya que, aunque este utilice los archivos de la carpeta de servicios, principalmente los que corren las pruebas, la sesión de Redis y el worker de Arq se inicializan aparte para mejor organización. El backend no usa los archivos que corren las pruebas, sino que las propias API llaman a los procesos internos de Arq y Redis que encolan los trabajos. El worker solo se encarga de recoger esos procesos y usar las funciones de ejecución para ejecutar y guardar las pruebas en la base de datos.

Por otro lado, encontramos la carpeta del frontend, la cual también está organizada siguiendo una separación por responsabilidades para que todo el proyecto esté en sintonía. En este caso, la carpeta llamada "src" contiene todo el código del frontend, aunque hay algunos archivos obligatorios externos a esta, como el main.tsx o las configuraciones necesarias para TypeScript o Tailwind. Es importante explicar que el archivo main.tsx actúa como punto de entrada del frontend, ya que desde ahí se inicializa la aplicación de React, se configuran las rutas, se integra TanStack Query y se controlan las rutas protegidas que requieren sesión iniciada. En la carpeta "src" encontramos el código completo desarrollado con React, TypeScript y Vite. Además, en este caso la división por responsabilidades nos deja con: la carpeta api, que contiene todas las comunicaciones con el backend; la carpeta de componentes, que contiene los elementos reutilizables de interfaz; y la carpeta de features, que agrupa las pantallas según la funcionalidad principal a la que pertenecen.

En la carpeta API del frontend encontramos el cliente HTTP usado por las distintas pantallas para lanzar peticiones contra la API. Centralizar esta comunicación evita que cada vista construya sus propias llamadas de forma independiente, por lo que operaciones como consultar auditorías, crear targets, subir datasets o recuperar resultados se realizan siempre desde el mismo punto de acceso.

La carpeta de componentes agrupa los tres componentes reutilizables de la interfaz, los cuales no pertenecen a una pantalla concreta, sino que se usan en varias partes de la aplicación. Estos elementos son: el layout general de la aplicación, los componentes visuales comunes como los pop ups de confirmación y el contenedor usado para representar gráficas que utiliza ECharts. Esta separación evita duplicar elementos de interfaz y hace que los cambios visuales comunes puedan aplicarse sin modificar cada pantalla por separado.

Por último, la carpeta features contiene las pantallas principales de la aplicación divididas por su responsabilidad dentro de la herramienta. Cada subcarpeta representa una funcionalidad concreta, como la autenticación, los targets, los datasets, las auditorías y la inferencia de términos. Las carpetas de autenticación, targets y datasets simplemente contienen la página correspondiente para su flujo, siendo las de login, targets y datasets sus páginas respectivamente. Estas contienen un formulario donde el usuario rellena unos datos y uno o varios botones que llaman a la función del archivo de la API, que llamará al backend con la petición necesaria. La carpeta de auditorías contiene las pantallas relacionadas con el flujo de auditoría, que en este caso son más al poder ver los resultados, crear nuevas o verificar los datos en un dashboard. Por tanto, las páginas son: el panel principal de auditorías, la creación de nuevas auditorías y la consulta de resultados. Igualmente, la carpeta de inferencia de términos sigue la misma idea aplicada a su flujo, encontrando las páginas para listar inferencias, crear nuevas pruebas y consultar sus resultados.

La Fig. 7 muestra la estructura de carpetas del proyecto, separando la parte del backend y la parte del frontend. La imagen sirve para visualizar de forma resumida la organización explicada en este apartado y ayuda a entender cómo cada bloque del código se ha agrupado según su responsabilidad dentro de la herramienta.

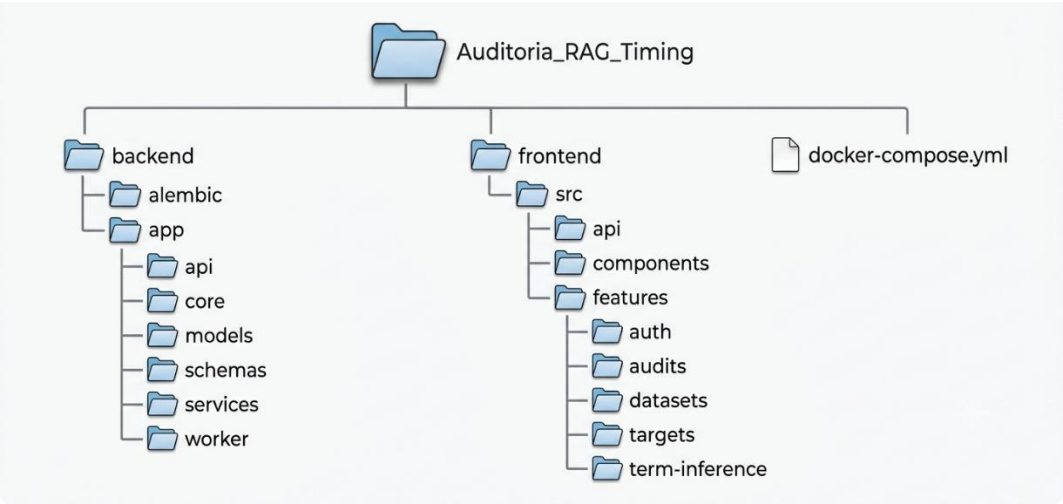


Fig. 7. Estructura de carpetas del desarrollo de la herramienta.
Fuente: elaboración propia con ayuda de ChatGPT.

5.3. IMPLEMENTACIÓN DEL BACKEND

El backend se implementó como la capa encargada de coordinar la herramienta, ya que su responsabilidad principal es recibir las acciones del frontend, comprobar que pueden realizarse, preparar los datos necesarios y devolver respuestas coherentes a la interfaz, entre otras. Por eso, el backend funciona como punto intermedio entre el usuario, la base de datos, la cola de trabajos y los módulos internos de la aplicación. Su implementación se hizo con FastAPI, manteniendo una separación clara entre las rutas HTTP y la lógica interna. Las rutas actúan

como entrada de las peticiones, pero no contienen todo el funcionamiento de la herramienta. Cuando llega una solicitud, la API interpreta la operación, obtiene una sesión de base de datos si la necesita, aplica las dependencias de seguridad y llama al servicio correspondiente. Con esto se evita que cada endpoint tenga que resolver por sí mismo procesos como la lectura de archivos, la creación de sesiones, la preparación de trabajos o la generación de respuestas para la interfaz. Esta forma de implementar el backend hace que se separen las responsabilidades, pero se centralizan las llamadas, haciendo que todas pasen por un mismo patrón. Una operación puede ser una consulta rápida, como listar targets, o puede iniciar una ejecución larga, como una auditoría, pero en ambos casos, el backend responde de forma rápida y deja el sistema en un estado consistente.

5.3.1. API Y SERVICIOS DE GESTIÓN

La API se construyó como una entrada controlada a las funciones de la herramienta. Cada petición recibida desde el frontend sigue un flujo parecido: entra por una ruta de FastAPI, se valida con el esquema correspondiente, se ejecuta la lógica necesaria en un servicio y se devuelve una respuesta estructurada.

Uno de los puntos más importantes de esta implementación es el uso de las dependencias de FastAPI. FastAPI permite declarar elementos que una ruta necesita para ejecutarse, como una conexión a base de datos o un usuario autenticado. Esto se usa para que las rutas no tengan que crear manualmente esos objetos cada vez, sino que puedan recibir lo que necesitan ya preparado y puedan centrarse en llamar al servicio que resuelve la operación, consiguiendo reutilizar el código que autentica a los usuarios y crea las sesiones de la base de datos. Estas sesiones se realizan mediante sesiones asíncronas, lo que permite que las rutas trabajen con PostgreSQL sin bloquear completamente el proceso que atiende la API. El backend usa estas sesiones para crear, leer, actualizar o eliminar entidades, pero la lógica concreta se mantiene en servicios aparte, como por ejemplo, cuando una ruta recibe una petición para crear una auditoría, el trabajo de comprobar los datos, crear la sesión inicial y preparar el encolado se realizará fuera de la propia ruta simplificando su implementación y haciéndola más modular.

Los distintos servicios del backend son los que implementan la lógica interna del mismo para la gestión y realización de las diferentes acciones. Un servicio puede encargarse de normalizar un dataset, preparar una respuesta de resultados, construir un resumen para una pantalla o generar un CSV. De esta forma, si en el futuro se cambia una regla interna, no hace falta modificar todas las rutas, sino el servicio concreto que contiene esa operación, de nuevo haciendo modular el backend y facilitando su mejora en el futuro. En el caso de los targets, es necesario guardar y comprobar todos los campos para que posteriormente el worker pueda reutilizar esta configuración para ejecutar las pruebas correctamente. Por eso, cuando se registra un target, se guardan los datos que permitirán construir peticiones HTTP más adelante, como: endpoint, headers, timeout, verificación TLS y plantilla del body. Los datasets son distintos, ya que en este caso el backend trabaja con subida y tratamiento de archivos. La API envía el archivo desde el frontend y su procesamiento se realiza en el servicio del backend, que lee el JSON, comprueba que puede interpretarse, calcula información útil para la interfaz y guarda una versión preparada para que el worker pueda cargarla después. Por otro lado, las pruebas, ya sean auditorías o inferencias, se implementan como operaciones que involucran al worker realizando una coordinación para su ejecución. En el caso de las auditorías, la función del backend es crear una sesión con la configuración seleccionada, dejarla en estado inicial y enviar a la cola una referencia para que el worker pueda ejecutarla. Esta decisión mantiene la API rápida y evita que una llamada desde el navegador dependa de una prueba que puede durar

mucho tiempo. Las inferencias de términos siguen la misma idea, pero con una preparación más específica al tener que preparar un perfil de calibración antes de la prueba. En este caso, ya sea partiendo de un CSV o de una auditoría previa, el backend se encarga de dejar el perfil de calibración preparado para que el worker lo pueda coger más adelante y trabajar con él en la clasificación de los términos.

Otro punto técnico importante que se ha implementado en el backend es que no siempre se desea devolver todos los datos que aparecen en la base de datos para la visualización de algo, por lo que en muchas rutas se preparan respuestas pensadas para la pantalla que las va a consumir. Por ejemplo, una vista de listado necesita información resumida, mientras que una pantalla de resultados necesita métricas, filtros y mediciones paginadas. De hecho, la paginación también forma parte de esta capa de gestión, ya que, como las auditorías pueden generar muchas mediciones y no tendría sentido devolverlas todas en una única respuesta, las rutas de resultados permiten pedir partes concretas del conjunto de datos.

Por último, el backend también se encarga de preparar las descargas cuando el usuario exporta resultados. En esos casos, la interfaz solicita el CSV y el backend construye el archivo a partir de los datos guardados, dejando que el frontend solo se encargue de la descarga, mientras que el backend hace todos los cálculos internos para generar todos los datos necesarios, evitando duplicar la lógica de exportación en el navegador.

5.3.2. SEGURIDAD Y AUTENTICACIÓN

El backend incorpora un sistema de autenticación para proteger las rutas internas de la herramienta, ya que, aunque el alcance del proyecto no requiere una gestión completa de usuarios con roles avanzados, sí era necesario evitar que cualquiera pudiera acceder a targets, headers, datasets o resultados de auditorías si la aplicación queda levantada en un equipo. El proceso comienza en la ruta de login, donde el usuario introduce sus credenciales desde la interfaz y el backend comprueba si corresponden con un usuario válido. La contraseña no se compara ni se guarda en texto plano, sino que se almacena un hash de la contraseña, y durante el inicio de sesión se calcula la comprobación contra ese hash. Cuando las credenciales son correctas, el backend genera un token de sesión que contiene la información necesaria para identificar al usuario durante las siguientes peticiones y tiene una caducidad temporal definida. En vez de entregar el token al frontend para que lo guarde manualmente en localStorage, el backend lo envía mediante una cookie HTTPOnly para reducir la exposición del token frente a código JavaScript ejecutado en el navegador, ya que una cookie HTTPOnly no puede leerse directamente desde scripts del frontend. A partir de ese momento, las peticiones del frontend se envían con credenciales incluidas. El backend recibe la cookie, extrae el token y comprueba si sigue siendo válido. Si el token no existe, está mal formado o ha caducado, la ruta protegida rechaza la petición. Con esto, el control real de la sesión queda en el backend y no en la interfaz. Además, como se ha explicado en el anterior apartado, las rutas internas de la herramienta usan una dependencia de autenticación que se ejecuta antes de permitir la operación solicitada.

El cierre de sesión se implementa eliminando la cookie desde el backend, haciendo que cuando el usuario pulsa la opción de logout en la interfaz, se llama a la ruta correspondiente y el servidor responde indicando al navegador que borre la cookie de sesión. Desde ese momento, las siguientes peticiones ya no llevan un token válido y el usuario vuelve a quedar fuera de las rutas protegidas.

Este sistema encaja con el tipo de herramienta desarrollada, ya que no se ha creado una plataforma multiusuario compleja porque el objetivo del trabajo es una herramienta de auditoría controlada, manejada por un único auditor. Aun así, usar una tabla de usuarios, hashes de contraseña, tokens con caducidad y cookies HTTPOnly permite que la aplicación tenga protección correctamente implementada. También hay que tener en cuenta que la herramienta puede almacenar configuración sensible dentro de los targets, especialmente si los headers incluyen tokens o claves de acceso al sistema RAG evaluado. Por esto, la autenticación protege no solo la entrada a la aplicación, sino también información que podría permitir llamar a sistemas externos.

5.4. IMPLEMENTACIÓN DEL PROCESAMIENTO ASÍNCRONO

Se ha considerado el procesamiento asíncrono como una de las claves de esta herramienta, ya que, al tener ejecuciones que pueden tardar mucho tiempo, creemos que un sistema de colas asíncrono encaja a la perfección con la lógica e implementación de la plataforma. Crear un target, cargar un dataset o consultar resultados son operaciones que el backend puede resolver en poco tiempo, en cambio, una auditoría puede requerir enviar cientos o miles de consultas a un sistema RAG externo, lo que puede llegar a tardar varias horas de prueba continua, por lo que si todo este proceso se ejecutara dentro de la misma petición HTTP que lanza el usuario, la interfaz quedaría bloqueada y sería más fácil que aparecieran problemas por timeout o por pérdida de conexión con el RAG externo.

Por esto el sistema diferencia entre guardar una sesión en la base de datos y ejecutarla con el worker, ya que de esta manera la API mantiene un comportamiento más controlado al poder entregar la respuesta de sesión creada y encolada correctamente en el momento de generarla, en vez de tener que esperar a que termine. De esta forma, una vez ya empezada la prueba por el worker, la interfaz puede consultar el estado de la prueba y mostrar el progreso poco a poco, de nuevo sin necesidad de esperar a que acabe la ejecución de la prueba, ya que esta consulta no retrasa la ejecución al encontrarse completamente diferenciada en el sistema. En la única parte que el sistema de gestión y el worker se unen o solapan es en las consultas a la base de datos, pero esto tampoco es problemático, ya que las pruebas primero hacen la medición y, una vez realizada, entonces intentan guardarla en la base de datos, por lo que si se genera un retraso al acceder desde el backend de gestión, no influirá en el tiempo de las mediciones, sino únicamente en el tiempo total de la prueba, pero de manera mínima.

En este apartado se explicarán las claves para que la herramienta funcione con este tipo de procesamiento, las cuales son: el encolado de trabajos, los estados de las pruebas y el correcto tratamiento de errores generados en la ejecución de las pruebas.

5.4.1. ENCOLADO DE TRABAJOS

Justo después de recibir los datos necesarios en la llamada de la API enviada desde la interfaz y de realizar las comprobaciones necesarias, como que los datos sean correctos o que existan en la base de datos, el backend guarda todos los datos asociados en una sesión con estado PENDING en la base de datos. Una vez guardada la sesión, el backend se dispone a enviar a Redis un trabajo asociado a esa auditoría, el cual, en vez de tener todos los datos, simplemente contendrá el identificador de la sesión creada. Los trabajos en Redis son muy sencillos de encolar, ya que Redis es una tecnología preparada para este tipo de utilidades, de manera que no es necesario crear un sistema de colas al ya estar implementado y simplemente se consigue usando las funciones internas que Redis nos ofrece. Por otro lado, el worker, cuando esté

esperando, revisará la cola de Redis y encontrará el trabajo pendiente. Este simplemente revisará el tipo de trabajo que es, verificando su nombre, y ejecutará la función que ya hace todo el proceso, desde buscar los datos asociados al ID de la sesión hasta ejecutar la prueba.

Es importante mencionar que el enfoque de no guardar todos los datos de cada trabajo, sino guardar solo el ID asociado al mismo, tiene su lógica, ya que si en la cola se guardara toda la configuración de la auditoría, podrían aparecer diferencias entre lo que está en Redis y lo que está en PostgreSQL. En cambio, al pasar solo el identificador, la base de datos se mantiene como fuente principal de información y la cola solo indica qué trabajo debe ejecutarse. Este enfoque también ayuda para utilizar esto tanto para las auditorías como para las inferencias de términos, ya que si la cola guardara los datos, tendría que guardar datos distintos para las auditorías y las inferencias y de esta forma no es necesario, al solo necesitar el ID y el nombre, en donde el worker sabrá si necesita buscar los datos en la tabla de inferencias o en la de auditorías.

Ambos trabajos realmente utilizan la misma lógica, ya que ambos se ejecutan con el worker con llamadas HTTP al sistema externo, pero realmente las pruebas son muy diferentes, ya que los procesos previos a la propia prueba son muy distintos. Por ejemplo, en una auditoría, el worker necesita cargar las consultas del dataset y mezclarlas según la semilla guardada, mientras que en una inferencia, necesita reconstruir el perfil de calibración, preparar las consultas que se enviarán por término, desordenar las consultas aleatoriamente y organizar las rondas de ejecución. El progreso de las ejecuciones se calcula de forma distinta también dependiendo de si es una auditoría o una inferencia. En las auditorías, esta se calcula a partir del número de consultas del dataset, ya que al crear la sesión se conoce el total de peticiones que se esperan ejecutar, por lo que el backend puede guardar ese valor como referencia. Durante la ejecución, el frontend irá consultando el progreso guardado por el worker para que pueda mostrar un avance real de la prueba sin tener que conocer los detalles internos de cada medición. Por esto se decidió guardar como dato aparte en la base de datos el progreso, ya que si no, para conocerlo, se necesitaría contar todas las filas realizadas, lo que generaría una sobrecarga innecesaria que solucionamos de este modo. Por otro lado, en la inferencia de términos, el progreso es algo menos directo, porque el número real de mediciones puede depender de las rondas y de los parámetros configurados. Por este motivo, la herramienta calcula una estimación inicial y después ajusta el progreso al número real de mediciones ejecutadas. Aunque no sea tan simple como en una auditoría, el planteamiento permite mostrar al usuario que la prueba está avanzando y que el worker sigue activo utilizando la misma columna de progreso actual en la base de datos. Por todo eso se encolan como trabajos distintos, aunque mucha parte del propio envío y medición de las pruebas utilice las mismas partes del código.

5.4.2. FLUJO DE ESTADOS Y TRATAMIENTO DE ERRORES

Para el correcto funcionamiento de la herramienta, los estados de las pruebas son muy importantes, ya que controlan todo el proceso de ejecución. Los estados se utilizan para que el worker y el sistema los comprueben y hagan una acción u otra dependiendo del estado de la prueba en ese momento. Para explicar el flujo por el que pasa el worker, es importante entender por qué estados pueden pasar las pruebas y cómo pueden evolucionar unos en otros.

Cuando una sesión de auditoría se guarda inicialmente, siempre se guarda con el estado PENDING. Antes de ejecutar las consultas reales, el worker realiza el número de peticiones de calentamiento configuradas en la sesión, las cuales no se guardan como resultados de la

auditoría, ya que su objetivo es comprobar que el target responde y preparar el camino antes de empezar a medir el dataset real. Si todas las peticiones de calentamiento fallan, la auditoría se marca como FAILED, porque no tendría sentido continuar con una prueba completa contra un target que no está respondiendo correctamente. Cuando la fase inicial termina correctamente, el worker cambia el estado de la sesión a RUNNING y comienza a ejecutar las consultas del dataset. Para cada consulta, construye la petición HTTP correspondiente, llama al target, mide los tiempos y guarda un resultado en la base de datos. El guardado de resultados se hace de forma periódica. En lugar de esperar al final de toda la auditoría para confirmar los datos, el worker realiza commits cada diez peticiones, lo que reduce el riesgo de perder toda la información si la auditoría falla en una fase avanzada y permite que el progreso y parte de los resultados queden disponibles durante la ejecución. Además de estos tres estados, la auditoría puede tener: el estado COMPLETED, que indica que la prueba ha terminado correctamente; el estado ABORT_REQUESTED, que indica que el usuario ha solicitado detener la ejecución; y el estado ABORTED, que confirma que el worker ha detectado esa solicitud y ha parado la prueba de forma controlada. Además, el estado de FAILED también se utiliza cuando la prueba no se puede seguir ejecutando por distintas razones.

La inferencia de términos sigue el mismo esquema general de estados, aunque su ejecución interna es distinta, ya que hay estados que pueden aparecer en momentos distintos. Cuando se crea una sesión de inferencia, también se guarda inicialmente con el estado PENDING. Cuando el worker recoge la sesión de inferencia, primero carga la configuración guardada en la base de datos: el target, la fuente de calibración, el modo de entrada y los parámetros de ejecución y, antes de enviar consultas nuevas, reconstruye el perfil de calibración. Esta calibración, como ya hemos comentado, puede venir de una auditoría previa o de un CSV y, por lo tanto, puede ser inválida por no cumplir las condiciones necesarias. Si esto ocurre, la sesión se marca como FAILED, ya que no tendría sentido clasificar términos usando una referencia incorrecta. Si la calibración es válida, antes de enviar las consultas de pruebas se utilizan unas consultas que también puede introducir el usuario para observar si el sistema responde con los tiempos esperados. Si estas consultas generan tiempo muy fuera de lo esperado, entonces la prueba de nuevo se marcaría como FAILED, ya que no tiene sentido continuar si el sistema no está funcionando como se espera en ese momento. Si esta comprobación pasa, el estado de la sesión pasa a RUNNING y el worker prepara las consultas que se van a enviar al target. En el modo basado en términos, la herramienta genera las consultas a partir de los términos introducidos, mientras que en el modo basado en consultas manuales, usa directamente las frases agrupadas por el usuario. Durante la ejecución, la inferencia se organiza por rondas. En cada ronda se envían varias consultas hasta alcanzar los límites definidos por los parámetros de la sesión, como el número de pruebas por ronda y el máximo de pruebas por término. Igual que en las auditorías, las mediciones se van guardando durante la ejecución y no solo al final. Cuando termina cada ronda, el worker calcula la clasificación de cada término. Para ello, compara la media observada de sus mediciones con el umbral temporal generado en la calibración. Cuando todas las clasificaciones se han calculado y guardado, la sesión pasa a COMPLETED. Además, el usuario podrá abortar la ejecución de la misma forma que en las auditorías, teniendo los mismos dos estados extras.

El sistema de cancelación diferencia entre ABORT_REQUESTED y ABORTED para implementar un sistema de cancelación más seguro. En este caso, la cancelación no se implementa matando directamente el proceso del worker, sino que el backend cambia el estado de la sesión a ABORT_REQUESTED y el worker consulta ese estado entre peticiones, dejando de enviar nuevas consultas y marcando la sesión como ABORTED. Esta forma de cancelar es

más segura que interrumpir el proceso en cualquier punto, porque permite cerrar la ejecución después de terminar la petición en curso y guardar el estado final.

Como ya se ha mencionado anteriormente, el tratamiento de errores es una parte clave en toda la herramienta. En cuanto al tratamiento de errores de las pruebas, este está integrado en el worker. Las llamadas a sistemas externos pueden fallar de forma aislada por diversas razones, por lo que en este caso es importante que el sistema no dé por fallida una prueba completa por errores aislados. Por esto se ha decidido que en esos casos la herramienta guarde la medición como error y continúe si el fallo no impide seguir. Sin embargo, cuando los errores se acumulan, el worker interpreta que la prueba ya no es fiable, por lo que, por ejemplo, en las auditorías, si se producen veinte errores consecutivos, la sesión se marca como FAILED. Esta regla evita seguir enviando peticiones cuando el target parece caído o mal configurado.

5.5. IMPLEMENTACIÓN DEL MÓDULO DE MEDICIÓN TEMPORAL

El módulo de medición temporal es una de las partes centrales de la herramienta, ya que convierte cada consulta del dataset en una petición HTTP medible contra el sistema RAG externo. Su función es ejecutar cada llamada de forma controlada y guardar los datos necesarios para que después puedan compararse los tiempos para su posterior análisis. El módulo trabaja siempre desde una posición externa al sistema auditado, ya que la herramienta solo envía peticiones HTTP y observa la respuesta que recibe. Por esto, los datos que se guardan son tiempos de comunicación y metadatos de la respuesta, no información interna del sistema evaluado.

Es importante mencionar que este módulo se usa tanto en las auditorías como en la inferencia de términos, ya que en ambos casos el proceso final consiste en enviar consultas a un target RAG externo y medir cómo responde. La diferencia entre una auditoría y una inferencia no está en la forma de medir una petición concreta, sino en cómo se preparan las consultas y en cómo se interpretan después los resultados. En una auditoría, las consultas proceden de un dataset ya cargado y están clasificadas por frecuencia y longitud. En la inferencia de términos, las consultas se generan o se reciben agrupadas por término, y después se comparan con un perfil temporal de calibración. Aun así, cuando llega el momento de enviar una consulta al target, los dos flujos usan la misma lógica: construir la petición HTTP, insertar la consulta en la plantilla del target, medir el Time-to-First-Byte, medir el tiempo completo y guardar el resultado.

Por este motivo, el módulo de medición se ha planteado como una pieza independiente de la lógica de análisis. Su responsabilidad es capturar datos temporales de forma ordenada, ejecutando la llamada, registrando lo que ocurre y dejando la información guardada para que otros módulos la procesen después. Esta separación es importante porque permite que la herramienta mantenga el mismo criterio de medición en todos los flujos, ya que si cada parte midiera las peticiones de una forma distinta, los resultados de auditorías e inferencias no serían comparables, pero al centralizar la medición, se mantiene una base común para todo el sistema.

5.5.1. PREPARACIÓN DE LA PETICIÓN HTTP

El primer paso antes de enviar una consulta al target es construir la petición HTTP de forma correcta [21]. Esta preparación se hace a partir de la configuración guardada en cada target, la cual contiene la URL del endpoint, los headers, el timeout, la configuración de verificación TLS y una plantilla JSON para el cuerpo de la petición [23].

La plantilla del body permite que la herramienta pueda trabajar con distintos sistemas RAG sin cambiar el código. No todos los endpoints reciben la pregunta con el mismo formato, ya que, como se ha explicado anteriormente, un sistema puede esperar un campo llamado question, otro puede usar query y otro puede tener una estructura JSON más anidada. Para resolver esto, la herramienta guarda una plantilla configurable que contiene el marcador {{QUERY}}, el cual en el momento de la ejecución se sustituye por la consulta concreta que se quiere enviar. Esta parte es común para auditorías e inferencias. En una auditoría, la consulta que se inserta viene del dataset. En una inferencia de términos, la consulta puede venir de una generación automática a partir de un término o de una frase introducida manualmente por el usuario. Para hacer esta sustitución, el sistema repasa campo por campo el JSON buscando el marcador específico y, cuando lo encuentra, lo intercambia por la consulta concreta, por lo que el resto de la estructura definida por el usuario queda exactamente igual que estaba en un inicio, lo que permite mantener parámetros fijos dentro del body, como configuraciones propias del endpoint u opciones necesarias en la llamada al RAG externo.

El resto de configuraciones del target se utilizan para la configuración de la petición. Los headers configurados en el target se aplican, ya que pueden incluir información técnica como el tipo de contenido o datos de autenticación necesarios para llamar al sistema externo. El timeout se usa para limitar el tiempo máximo que la herramienta espera una respuesta. Esto también se utiliza en ambos flujos, ya que en una auditoría, una consulta bloqueada podría retrasar toda la sesión y en una inferencia de términos podría afectar al cálculo temporal de un término concreto. Si se supera el timeout, la medición se registra como error y el worker decide si puede continuar o si la sesión debe marcarse como fallida. Por último, la verificación TLS también se lee desde la configuración del target [17]. Este campo permite indicar si la herramienta debe validar el certificado TLS del sistema objetivo. En entornos reales se mantiene activado, ya que garantiza que la conexión HTTPS se establece contra un servidor con certificado válido. Sin embargo, durante pruebas de laboratorio puede ser necesario desactivarlo para auditar servicios internos o endpoints con certificados autofirmados. Esta opción no modifica la lógica de auditoría, solo cambia la forma en la que el cliente HTTP acepta o rechaza la conexión segura con el target.

Una vez preparada la URL, los headers, el body y los parámetros de conexión, la petición se envía mediante HTTP POST. Esta forma de envío es la que utilizan la mayoría de los endpoints de consulta en sistemas RAG, donde la pregunta se manda dentro del cuerpo de la petición.

5.5.2. MEDICIÓN DEL TIME-TO-FIRST-BYTE Y DEL TIEMPO COMPLETO

La medición se realiza con un cliente HTTP asíncrono que permite trabajar con respuestas en streaming, lo que significa que la herramienta puede detectar el primer fragmento de respuesta cuando llega, sin esperar obligatoriamente a descargar el cuerpo completo. Esta característica es la que permite calcular el Time-to-First-Byte desde el punto de vista del cliente externo.

El cronómetro se inicia justo antes de lanzar la petición HTTP al target. Desde ese instante se mide el tiempo que transcurre hasta que la herramienta recibe el primer fragmento del cuerpo de la respuesta. Ese valor se guarda como Time-to-First-Byte. Después, la herramienta sigue leyendo la respuesta hasta el final y calcula el tiempo completo de respuesta. En este proyecto, el TTFB se usa como métrica principal porque reduce la influencia del envío completo del mensaje. Si una respuesta es larga y se transfiere por partes, el tiempo completo puede crecer por el tamaño del cuerpo y por la descarga de todos los fragmentos. El TTFB corta la medición

en la llegada del primer fragmento, por lo que evita que toda la transferencia final pese igual que el inicio de la respuesta. Además, esta métrica se aplica igual en auditorías, en donde cada consulta del dataset genera una medición con su Time-to-First-Byte y en inferencias, en donde cada consulta también genera una medición pero esta va asociada a un término concreto. El tiempo completo de respuesta se conserva como segunda medida ya que indica cuánto tarda la herramienta en recibir todo el cuerpo de la respuesta. En respuestas pequeñas, puede ser muy parecido al Time-to-First-Byte pero en respuestas largas, puede haber más diferencia por lo que guardar ambos valores permite un análisis en mayor profundidad posteriormente.

En la Fig. 8 se observan los dos puntos de medición temporal que se registran durante una petición. Primero se mide el tiempo desde el inicio de la llamada hasta la llegada del primer fragmento de respuesta, que corresponde al TTFB y después se continúa leyendo la respuesta hasta recibir el cuerpo completo, obteniendo así el tiempo total de respuesta.

```
started_at = time.perf_counter()
async with client.stream("POST", target.endpoint_url, json=json_payload) as response:
    chunks = response.aiter_bytes()
    first_chunk = await anext(chunks, b"")
    first_body_byte_at = time.perf_counter()
    response_size = len(first_chunk)
    async for chunk in chunks:
        response_size += len(chunk)
completed_at = time.perf_counter()
```

Se comienza la medición temporal

Una vez recibido el primer chunk de información se mide el TTFB

Cuando se recibe el mensaje completo se mide de nuevo el tiempo

Fig. 8. Puntos de medición del Time-to-First-Byte y del tiempo completo.

5.5.3. REGISTRO DE MEDICIONES Y ERRORES

Después de cada llamada, la herramienta guarda el resultado de la medición. Cada petición deja una evidencia asociada a la sesión que la ha generado, con los tiempos obtenidos y la información necesaria para interpretar lo que ha ocurrido. En el caso de las auditorías, cada medición queda asociada a la sesión de auditoría, a la consulta enviada y a sus etiquetas de frecuencia y longitud, las cuales son necesarias para que después el análisis pueda agrupar los resultados por categorías. Por otro lado, las inferencias de términos asocian cada medición a la sesión de inferencia y al término que se está evaluando. Esto permite calcular después la media observada de cada término, su desviación, la distancia al umbral de calibración y la clasificación final.

En el registro de las mediciones también se registra el código HTTP, el tamaño de la respuesta, la marca temporal, si la petición ha terminado en error y, en ese caso, el tipo y el mensaje del error. Esta información permite revisar tanto ejecuciones correctas como peticiones fallidas. El código HTTP ayuda a distinguir una respuesta válida de una respuesta recibida con error, ya que una petición puede devolver un código 4xx o 5xx y aun así tener un tiempo medido. En ese caso, la medición se conserva, pero se marca como error para que el análisis pueda tratarla de forma adecuada, lo que evita mezclar respuestas correctas con fallos del target y, por lo tanto, analizar erróneamente los datos correctos. Por otro lado, el tamaño de la respuesta se guarda como dato de apoyo, ya que ayuda a interpretar diferencias entre el Time-to-First-Byte y el tiempo completo, generando un análisis más completo de los datos.

Cuando ocurre un error de cualquier tipo, ya sea por timeout, conexión, TLS o cualquier otro fallo, el módulo de medición se encarga de medirlo y registrarlo correctamente. Esto es importante, ya que hace que el módulo de medición no tenga ninguna decisión sobre las

pruebas, guardando cualquier medición de forma ordenada, en este caso marcando la medición como fallida con su código HTTP y error generado. El que se encarga de decidir si la prueba sigue adelante o se debe parar, como hemos explicado anteriormente, es el worker.

Por último, es importante explicar que el guardado de mediciones se hace de forma progresiva durante la ejecución. En las auditorías, los resultados se van confirmando periódicamente para reducir el riesgo de perder información si la sesión falla en una fase avanzada. Estos guardados no se hacen con cada consulta, ya que esto podría generar un tráfico demasiado intenso en la base de datos, por lo que se ha elegido el número de diez mediciones para que se realice un guardado en la base de datos, no perdiendo mucha información si ocurre un fallo y reduciendo considerablemente los accesos a la base de datos. Por otro lado, en las inferencias ocurre lo mismo, guardando los datos cada diez mediciones, pero además guardándolos cada vez que se acaba una ronda.

5.6. IMPLEMENTACIÓN DEL ANÁLISIS ESTADÍSTICO

Una vez terminada una prueba, los módulos de análisis estadístico entran en juego, ya que son los que ayudan a estudiar las mediciones obtenidas y a entender si se puede sacar alguna conclusión útil. Estos módulos se encargan de transformar los registros individuales en gráficas y tablas con datos que son fácilmente comprensibles. Además, estos son diferentes para la inferencia de términos y para las auditorías, por lo que se necesita estudiar en cada uno. En las auditorías es necesario explicar si hay diferencia de latencia entre los diferentes grupos de frecuencia y longitud, mientras que en las inferencias se estudia si los términos en concreto pertenecen a alguno de estos grupos o no. Por lo tanto, las tablas y gráficas utilizadas en cada uno son distintas para poder obtener un análisis más específico.

Es importante explicar que en esta parte, la herramienta trabaja con datos de la base de datos como consultas enviadas, etiquetas de frecuencia y longitud, perfil de calibración, tiempos medidos, códigos HTTP y errores. Con esa información se calculan métricas agregadas, se comparan grupos, se clasifican términos y se prepara la información que después se muestra en la interfaz.

5.6.1. ANÁLISIS ESTADÍSTICO DE AUDITORÍAS

En el análisis de auditorías se parte de los resultados de una sesión concreta guardados en la base de datos. La herramienta recupera las mediciones válidas asociadas a esa auditoría y las agrupa según las etiquetas que tenía cada consulta en el dataset lo que permiten separar las consultas por frecuencia y longitud, que son las dimensiones usadas para comparar comportamientos temporales.

Una vez recuperados los datos, lo primero que se hace es un filtrado de valores extremos cuando hay suficiente número de muestras. Este filtrado se aplica usando los percentiles 1 y 99, y solo cuando el grupo tiene al menos cien mediciones, evitando que una medición aislada muy alta o muy baja distorsione demasiado las métricas agregadas. Una vez limpios los datos, se puede empezar a sacar métricas útiles para su análisis, ya que si lo hiciéramos antes del filtrado, mediciones como la media, sensibles a datos extremos, podrían verse afectadas por las medidas extremas. Es importante mencionar que estas medidas no se eliminan, sino que se quitan para el cálculo de los datos, pero posteriormente se le da la opción al usuario de poder mostrarlos en tablas como la de distribución, marcados de otra forma para poder diferenciarlos.

Además, esta limpieza de datos se realiza por grupo, ya que no tendría sentido limpiar los datos de todos los grupos juntos al tener que diferenciarlos.

Con los datos limpios para cada grupo, se calculan métricas como la media, la mediana, la desviación típica, los percentiles, el valor mínimo y el valor máximo [25]. Estas métricas permiten ver si un grupo tiende a responder más rápido que otro y si las mediciones son estables o tienen mucha dispersión. La media permite comparar tendencias generales, pero la mediana y los percentiles ayudan a interpretar mejor los casos con valores extremos. Además, la herramienta también calcula la tasa de error de cada grupo, lo cual es necesario porque una comparación temporal puede perder valor si uno de los grupos tiene muchos fallos. Por ejemplo, si un conjunto de consultas tiene tiempos más bajos, pero también un porcentaje alto de errores, no se puede interpretar igual que un grupo con respuestas válidas y estables. Por eso, los errores no se ocultan, sino que se integran como parte del resumen.

La comparación principal entre grupos se realiza mediante una prueba estadística que permite obtener un p-value. En este caso, la herramienta aplica comparaciones entre grupos de frecuencia, especialmente entre consultas de alta y baja frecuencia dentro de una misma longitud. Esta comparación busca detectar si las diferencias observadas en los tiempos pueden considerarse relevantes desde el punto de vista estadístico o si podrían deberse al ruido de la medición. El p-value no se interpreta como una prueba absoluta de presencia o ausencia, sino que su función dentro de la herramienta es dar una medida de separación entre grupos temporales. Si el valor es menor a 0.05, la diferencia entre los grupos comparados resulta estadísticamente útil, mientras que si el valor es mayor, se considera que la prueba no ha generado suficiente evidencia para afirmar que los grupos se comportan de forma distinta. Además, para facilitar el análisis de esto, la herramienta clasifica la fuerza de la evidencia en distintos niveles. Cuando no hay suficientes muestras o no se puede calcular el p-value, la evidencia se marca como insuficiente. Si el p-value cae por debajo de ciertos umbrales, la separación se etiqueta como débil, moderada o fuerte. Estas etiquetas sirven para que el usuario pueda interpretar el resultado sin leer únicamente valores numéricos, aunque los valores originales se mantienen disponibles.

Como ya hemos mencionado, el análisis se calcula a partir de las mediciones existentes y no modifica los datos originales, lo que permite que los resultados brutos sigan disponibles para revisión o exportación. Las gráficas y tablas de la interfaz consumen estos resúmenes para mostrar comparaciones entre grupos, distribuciones temporales y evolución de las mediciones durante la auditoría. De esta forma, el análisis estadístico de auditorías convierte las mediciones individuales en resultados interpretables, manteniendo los datos brutos disponibles para trazabilidad y ofreciendo al auditor una visión agregada del comportamiento temporal del sistema.

5.6.2. ANÁLISIS ESTADÍSTICO DE TÉRMINOS INFERIDOS

En la inferencia de términos, lo principal que necesitamos hacer es clasificar los términos en probablemente presentes o ausentes. Esto debe mostrarse en el análisis estadístico para dar al usuario una forma de comprender los datos y entender si el término está más cerca de una clase o de otra. Obviamente, para poder mostrarlo, además de poner la etiqueta a cada término, también se necesitarán varias gráficas. Pero para conseguir todo esto, lo primero que necesitamos es una forma de clasificar los términos, y para ello hemos usado los perfiles temporales de calibración.

Como ya hemos mencionado anteriormente, estos perfiles se pueden construir tanto con un CSV con valores de una auditoría externa como con los valores de una auditoría que ya esté guardada en la base de datos y se haya hecho previamente. Con el resumen de la prueba, lo que se hace es buscar las fronteras para clasificar los términos. En este caso, los únicos dos grupos que se utilizan para clasificar son las consultas de alta y baja frecuencia, ya que intentar clasificar también la clase de media frecuencia es demasiado complicado para la herramienta. Para crear estos perfiles de calibración, la herramienta utiliza el resumen de la auditoría, recogiendo los valores de la media, tanto del grupo de alta frecuencia como del grupo de baja frecuencia, y generando una frontera que será el punto medio entre las medias, la cual funcionará para clasificar los términos. Además, ya que si la media de las consultas usadas para clasificar un término cae muy cercana de esa frontera, no sabríamos si realmente la clasificación ha sido correcta, se crea una zona gris, la cual es la desviación estándar más grande entre la de los dos grupos. Si la clasificación del término cae dentro de esa zona gris que abrazará a la frontera por arriba y por abajo, el término se clasificará como inconcluso, ya que no hay suficiente margen para poder afirmar que ese término pertenece a un grupo o a otro. Con todos esos valores calculados, el perfil de calibración se guarda en la base de datos como un JSONB, junto al resto de datos de la sesión de inferencia.

Una vez tenemos el perfil de calibración creado, ya tendríamos todo lo necesario para clasificar los términos. Lo primero que hace la herramienta antes de lanzar las consultas de la prueba reales es enviar unas consultas de calentamiento que, además, en este caso, servirán para verificar si la herramienta está funcionando correctamente. Posterior a ese primer paso, se realiza toda la prueba mandando consultas por rondas, como ya hemos explicado, utilizando los parámetros que pone el usuario de rondas máximas y número de consultas por rondas. Cuando termina cada ronda, el programa calcula las medias para cada término, y estas son las que se usan para clasificar los términos. Si esa media cae fuera de la zona gris, el término queda clasificado en el grupo en el que se encuentre su media. Pero si cae dentro de la zona gris, el término pasará a la siguiente ronda hasta completar el número máximo de rondas definidas por el usuario. Si al acabar la prueba, la media del término no ha salido de esa zona gris, se clasificará como inconcluso.

Con la prueba terminada, podremos observar el análisis de resultados al completo. En este caso, el análisis estadístico de la inferencia de términos es diferente al de auditorías, ya que presenta distintas gráficas al necesitar explicar la prueba y cómo ha ido avanzando esta a lo largo de las rondas. La primera gráfica ayuda a explicar por qué cada término se ha clasificado de una forma concreta, ya que muestra las fronteras y líneas de las zonas grises con líneas de trazos y con puntos las medias de los distintos términos. Debajo de esta gráfica, encontramos otra que nos explica cómo ha ido evolucionando la prueba a lo largo del tiempo, ya que esta prueba es algo más compleja que las auditorías al poder clasificar los términos en distintas rondas de la prueba, haciendo que solo algunos pasen a la siguiente ronda. Por lo tanto, en esta prueba se nos explica qué términos se han clasificado en cada ronda, cómo se ha clasificado, o si no se ha clasificado ninguno, mostrando una línea de progresión a lo largo de las rondas. Más abajo también hemos incluido tablas con datos útiles para que el usuario entienda los resultados más fácilmente. En estas tablas presentamos los términos y su etiqueta de clasificación. Además, podemos observar datos como la desviación estándar, la media o la mediana que el usuario podrá comprobar para confirmar que están dentro de las fronteras esperadas. Por último, en el análisis de resultados, como en las auditorías, también mostramos todas las consultas enviadas junto a todos sus metadatos adjuntos, como por ejemplo el código HTTP obtenido o la ronda de envío.

Como ya hemos comentado anteriormente, el módulo de inferencia de términos no genera resultados definitivos, ya que se basa en variables que pueden ser modificadas a lo largo del tiempo, como el funcionamiento del sistema RAG externo o el tiempo de red. Además, esta prueba es mucho más sensible que las auditorías al ser mucho más rápida y tener menos envíos de consultas, pero nos da unos resultados que son interpretables y útiles para poder intuir qué términos están presentes y cuáles no en la base de datos.

5.6.3. EXPORTACIÓN DE RESULTADOS

En ambos módulos de análisis estadístico se incluye un módulo aparte que permite exportar distintos resultados en varios formatos para que el usuario los pueda utilizar fuera de la interfaz, ya que, aunque la interfaz de la aplicación muestre gráficas y tablas útiles, a veces estos datos deben extraerse de la herramienta para poder ser enviados a diferentes sitios o analizados con otros programas externos, por lo que la herramienta también lo permite.

En el caso de las auditorías, la herramienta permite exportar dos tipos de CSV [24]. El primero, con mediciones individuales, en donde cada fila del CSV representa cada petición enviada, incluyendo toda la información de la consulta, como la propia query, la frecuencia, la longitud, el tiempo completo o el código HTTP. Con este archivo podremos revisar la ejecución de la prueba al completo, ya que también se incluirán todas las llamadas que hayan generado errores y su explicación. Por otro lado, también se permite exportar otro CSV que nos aporta un resumen estadístico de la auditoría. Este CSV tiene una fila por grupo analizado, y será el que se utilizará si se quiere hacer una inferencia de términos usando un CSV externo. En este encontraremos los datos calculados de cada combinación de frecuencia y longitud, con lo que para la frecuencia alta tendremos una fila para la frecuencia alta y longitud corta, otra para la frecuencia alta y longitud media y otra para la frecuencia alta y longitud larga, y así con todo el resto de grupos. En estas filas se incluirán todos los datos útiles, como la media, la mediana o la desviación estándar de estos grupos para el análisis posterior de los mismos.

En cuanto a la inferencia de términos, la exportación simplemente se centra en los resultados por término y no da la posibilidad de exportar toda la prueba al completo, consulta por consulta. En este caso, creemos que con la información aportada por término es suficiente para entender cómo ha ido la prueba, ya que en esta información irá la información útil como la media observada, la desviación, las mediciones válidas, la clasificación final o el target evaluado. Con esta información será más que suficiente para poder entender la prueba al completo.

Los formatos de entrada y salida más relevantes de la herramienta, incluyendo ejemplos de configuración de targets, datasets, entradas de inferencia y exportaciones CSV, se detallan en el Anexo C.

5.7. IMPLEMENTACIÓN DE LA INTERFAZ DE USUARIO

La interfaz de usuario, como se ha comentado en las tecnologías usadas, se implementó como una aplicación React con TypeScript y Vite. La implementación de la interfaz se ha centrado en construir una capa de control que permitiera lanzar pruebas largas sin bloquear al usuario y sin interferir en las mediciones temporales. Para esto, la aplicación se estructuró como una SPA (Aplicación de Página Única) para las páginas simples, lo que significa que el navegador no carga una página HTML nueva cada vez que el usuario cambia de sección, sino que React cambia el componente visible en función de la ruta. Esta lógica se implementó con React Router, en donde las rutas principales se definen en el punto de entrada de la aplicación y cada

una carga la vista correspondiente, por lo que moverse entre auditorías, targets, datasets o inferencias es más rápido y el estado de la interfaz es controlado por React, decidiendo qué se muestra en cada momento. Por otro lado, para las pantallas más pesadas, como los resultados de auditorías y los resultados de inferencia de términos, se implementó una carga diferida de componentes, lo que significa que estas vistas no se cargan desde el inicio de la aplicación, sino cuando el usuario entra en ellas. Esto tiene sentido porque son pantallas con gráficas, tablas y más lógica de representación que generan cálculos en el backend. De esta forma, la aplicación inicial carga menos código y las partes más complejas solo se piden cuando realmente son necesarias.

Para mejorar la seguridad de la aplicación y evitar que cualquier persona acceda a las pantallas internas, las rutas privadas se envolvieron en un componente de protección, el cual consulta el estado de sesión del usuario antes de mostrar el contenido. Si la sesión existe, carga el layout principal, pero si no existe, redirige al usuario al login, lo que fuerza al usuario a introducir de nuevo las credenciales en el sistema. De esta manera conseguimos optimizar el código de la aplicación, ya que la responsabilidad de comprobar si el usuario está bien logueado o no se queda en la página de rutas en vez de estar en cada una de las páginas con rutas privadas. Además, para comprobar las credenciales del usuario, como se ha explicado en el apartado de seguridad, se implementó un sistema de uso de cookies HTTPOnly gestionadas desde el backend, las cuales caducan con el tiempo y son guardadas en el navegador, evitando usar tokens JWT guardados en *localStorage*, que son más vulnerables a ataques XSS. Por esto, las peticiones del frontend se envían con credenciales incluidas, adjuntando la cookie automáticamente.

Para la comunicación con las API del backend, se realizó un cliente HTTP propio, creando una capa con todas las llamadas API distintas que puede hacer el sistema y centralizando así todas las llamadas en vez de realizar cada llamada en cualquier archivo. Además, este archivo ya se encarga de añadir las credenciales necesarias, interpretar las respuestas JSON, gestionar descargas de archivos y detectar errores de autenticación, controlando por completo la comunicación con el backend.

Otra parte interesante de la implementación de la interfaz de usuario es el uso de TanStack Query, ya que se implementó para manejar datos que cambian con frecuencia. Cuando se lanza cualquier prueba, el backend responde rápido con la sesión creada, pero el trabajo real se queda ejecutando en el worker. Por esto, la interfaz necesita ir preguntando el estado sin recargar la página completa, ya que si se necesitase una recarga, el proceso de consulta sería demasiado lento y perderíamos la visualización de algunos datos estáticos cada vez que la interfaz consultase los datos para actualizarlos. Por esto, la configuración de consultas con refresco automático en las pantallas donde hay auditorías o inferencias activas es muy necesaria, aplicándola también a filtros de resultados, lo que se consiguió haciendo que los filtros formasen parte de los parámetros de consulta, por lo que cuando el usuario cambie cualquier parámetro del filtro, como la frecuencia o el código HTTP, React Query solicitará únicamente esa página filtrada al backend. Esta misma idea se usó al aplicar paginación en las tablas completas de resultados, ya que, como las auditorías pueden tener miles de mediciones guardadas, no tendría sentido cargar todos los datos de una sola vez en el navegador. Por lo tanto, cuando el usuario cambia de página, la interfaz pide únicamente los resultados necesarios para esa vista. Con esto, la tabla puede manejar auditorías grandes sin que la página se vuelva pesada.

Por otro lado, en la implementación de toda la aplicación se ha intentado reutilizar el máximo código posible, simplificando los archivos lo máximo posible. Esta metodología también se ha aplicado en la interfaz de usuario en las partes repetidas a lo largo de la herramienta, como el layout, los diálogos de confirmación, el diseño de las pantallas y apartados y el color y forma de los botones. El layout principal mantiene una barra lateral fija y una zona de contenido desplazable por practicidad al tener páginas que pueden ser muy grandes al contener buena parte de los resultados individuales de algunas pruebas.

Los formularios se implementaron como componentes controlados en los que el campo que escribe el usuario queda reflejado en el estado del componente, lo que permite validar y enviar los datos de forma controlada. Además, en los formularios que reciben JSON, como el body de un target o los términos de una inferencia, se implementó una validación previa en el frontend para detectar errores de sintaxis antes de enviar la petición. Si el texto no se puede convertir a JSON, la pantalla muestra un error y no llama al backend. Aun así, la validación importante se mantiene en el servidor para mayor seguridad.

La visualización de resultados se implementó con ECharts, creando un componente común encargado de renderizar las gráficas dentro de React. Esta decisión se tomó porque las gráficas de la herramienta necesitan más control que una representación simple de barras, pudiendo modificarlas para mejor visualización, centrándose en los datos y añadiendo valores extremos. Por esto, las opciones de cada gráfica se construyen desde las páginas de resultados y se pasan al componente común, que se encarga únicamente de dibujarlas. Estos controles de mejor visualización son claves para facilitar la interpretación de los datos, ya que por defecto algunas representaciones se centran en el rango real de valores, mientras que en las pruebas de latencia las diferencias importantes pueden ser pequeñas, por lo que si el eje empieza siempre en cero, esas diferencias pueden quedar casi invisibles. Por esto se añadió la opción de cambiar la escala de algunas gráficas, permitiendo al usuario decidir si quiere ver la magnitud completa o centrarse en las variaciones observadas durante la prueba.

Por último, es importante explicar que la exportación de resultados se implementó desde la interfaz mediante descargas de archivos generados por el backend. Cuando el usuario solicita un CSV, el frontend no construye el archivo manualmente, sino que pide el contenido al servidor y lo recibe como un archivo descargable. Después, crea una URL temporal en el navegador y lanza la descarga. Esta forma de hacerlo mantiene la generación de datos en el backend, donde están las mediciones y los cálculos, mientras que la interfaz solo se encarga de iniciar la descarga para el usuario.

5.8. EJECUCIÓN DE LA HERRAMIENTA

La herramienta está preparada para ejecutarse en local, ya que el uso predefinido de la misma es para un único usuario que la levante en su ordenador y que quiera probar un sistema externo. Para esto, el primer paso será que el usuario se descargue o clone el repositorio donde está la aplicación y copie el archivo .env.example en un archivo .env, ya que la aplicación recoge de este todas las variables de entorno utilizadas para el funcionamiento de la aplicación. Con esto, el usuario estará preparado para poder ejecutar el comando de Docker especificado en el README del repositorio que le levantará todos los contenedores de la herramienta.

Para la ejecución se ha utilizado Docker Compose, ya que es la manera más sencilla de levantar varios procesos de forma conjunta. En este caso, los componentes que se levantan en Docker Compose son el backend, el frontend, la base de datos, la cola de trabajos y el worker. Además,

durante las pruebas también se añadió un servicio de un sistema local donde poder enviar las peticiones para probar el módulo de medición temporal y envío de peticiones a sistemas externos. Para el correcto funcionamiento de Docker Compose, en cada contenedor se utiliza la tecnología necesaria, como por ejemplo en la base de datos PostgreSQL o en la cola de trabajos Redis. Además, es muy importante mencionar que cada contenedor solo tiene las variables de entorno necesarias, por lo que el frontend no conocerá valores que no necesite y el backend tampoco. Con esto aportamos un sistema profesional en el que se levantan unos contenedores solo con la información indispensable para que funcionen correctamente y no más de la necesaria, ya que darle conocimiento al frontend de variables de entorno que no utilizan es una mala práctica que hay que evitar para la seguridad de los sistemas. En este caso, en los Dockerfiles necesarios para levantar el contenedor de backend y frontend, solo en el del frontend se utiliza el comando CMD para correr la inicialización de la aplicación, ya que el Dockerfile del backend se comparte entre el backend de gestión y el worker, y por tanto necesitamos poner este comando en el Docker Compose para que se corra por separado, aunque las instalaciones de paquetes y las dependencias son las mismas para ambos contenedores. En este caso, uno inicializará la aplicación de Arq y otro inicializará la aplicación de FastAPI con los comandos correspondientes.

En Docker Compose es necesario seguir un orden correcto de arranque, ya que si arrancamos un proceso como el backend, que depende de otros procesos, como por ejemplo la base de datos, antes de arrancar la propia base de datos, pueden generarse problemas en el funcionamiento de la aplicación. Por lo tanto, en este caso, primero se arrancan los servicios de infraestructura, que son la base de datos y la cola de trabajo, y posteriormente se arrancará el backend y el worker, que ambos tienen los mismos requerimientos para su funcionamiento, ya que comparten el mismo código. Y, por último, se arrancará el frontend, ya que depende del backend y, por tanto, de todo lo demás.

El primer arranque de la aplicación es distinto a los demás, ya que la aplicación necesita generar la base de datos aplicando las migraciones definidas por Alembic. Estas migraciones crearán todas las tablas necesarias para el funcionamiento de la aplicación, como son la tabla de usuarios, targets, datasets, auditorías, resultados, inferencias y todas las que les rodean. Además, se generará el único usuario que utiliza el sistema. Ya que, como no tenemos sistema de creación de usuarios al no ser necesario en este tipo de aplicaciones, el sistema creará en el primer arranque el único usuario necesario para el funcionamiento correcto de la aplicación. Cuando el sistema haya arrancado y las migraciones se hayan ejecutado automáticamente, el usuario ya podrá entrar a la herramienta desde la interfaz de login introduciendo las credenciales correctas.

Una vez ese primer arranque esté realizado, el usuario podrá apagar y volver a arrancar la aplicación las veces que quiera sin perder ningún dato, ya que en la configuración de Docker se han utilizado volúmenes para que la persistencia siga operativa, incluso después de apagar la aplicación o limpiar los contenedores. Con la aplicación corriendo, el usuario podrá realizar todos los flujos que se han explicado anteriormente, como registrar un target o crear una auditoría y verificar sus resultados.

Esta forma de ejecución se ha planteado como la más adecuada para el funcionamiento de esta aplicación, ya que, al solo necesitar funcionar en el ordenador local del auditor que va a ejecutar la prueba, no es necesario generar un sistema de ejecución más complejo que se pueda instalar en servidores. Es mejor tenerlo simplificado y que se ejecute de manera sencilla con un comando para que más gente pueda entender su uso y su funcionamiento. Además, al utilizar

Docker, hay menor probabilidad de obtener errores por diferencias en las dependencias al instalarlos en el momento dentro del contenedor creado para ejecutar la aplicación.

Los pasos concretos para preparar el entorno, configurar las variables necesarias, levantar los contenedores y acceder a la herramienta se recogen de forma práctica en el Anexo B.

6. PRUEBAS Y EVALUACIÓN EXPERIMENTAL

La fase de pruebas de este trabajo tiene como objetivo validar de forma experimental la hipótesis planteada durante el desarrollo del proyecto. Como se ha explicado en los capítulos anteriores, el sistema desarrollado no consiste únicamente en una aplicación software convencional, sino en una herramienta orientada a medir y explotar una diferencia temporal nunca antes estudiada dentro de arquitecturas RAG. Por este motivo, antes de evaluar la herramienta final, es necesario demostrar que el canal lateral utilizado existe y que los resultados obtenidos no dependen únicamente de ruido del entorno o de una configuración concreta. Para esto, en un inicio se ha realizado una investigación centrada en comprobar si las consultas asociadas a información presente en la base de datos generan una diferencia de latencias respecto a las consultas asociadas a información no presente. Esta parte es fundamental, ya que permite validar el ataque antes de construir una herramienta completa sobre él.

Una vez estudiado el comportamiento del sistema, se ha utilizado ese conocimiento para orientar el diseño y desarrollo, explicados con anterioridad, de la herramienta software. La aplicación final es utilizada para la automatización de la explotación del ataque encontrado y definido en la investigación previa. Por ello, sus módulos principales, como la ejecución repetida de consultas, la medición de tiempos, el filtrado estadístico y la presentación de resultados, están directamente relacionados con las conclusiones obtenidas en la fase experimental. Por tanto, las pruebas no se pueden limitar a comprobar que el sistema funciona, sino también a observar si la herramienta cumple su objetivo, pudiéndose usar para automatizar la explotación del ataque.

6.1. DISEÑO EXPERIMENTAL PREVIO

Antes de ejecutar las pruebas, se definió un diseño experimental que permitiese medir el canal lateral de forma controlada. Este diseño se centra en cuatro aspectos principales: la hipótesis que se quiere validar, las variables que pueden afectar a la medición, el corpus y las consultas utilizadas, y el protocolo seguido para ejecutar las pruebas. Con esto se busca que los resultados obtenidos no dependan de una consulta aislada o de una configuración concreta, sino de una diferencia temporal repetible asociada al vector de ataque expuesto.

6.1.1. HIPÓTESIS Y MODELO DE AMENAZA

La hipótesis de esta fase experimental parte de observar el comportamiento interno de las bases de datos vectoriales. Cuando un sistema RAG recibe una consulta, no compara el texto con todos los documentos almacenados de forma secuencial, eso colapsaría cualquier servidor y sería demasiado lento, ya que los RAG pueden tener almacenados millones de documentos. En

su lugar, el sistema transforma la pregunta del usuario en un vector numérico y utiliza algoritmos de búsqueda aproximada para encontrar la información más similar a la buscada.

La hipótesis sostiene que el tiempo de ejecución de estos algoritmos de búsqueda no es constante. Las variaciones de milisegundos en el procesamiento interno dependen directamente de si la información buscada ya reside en la base de datos o si es completamente distinta a la almacenada en el sistema.

Para entender dónde se encuentra esta diferencia de latencia, hay que centrarse en el motor que utilizan las bases de datos vectoriales. Estas bases de datos pueden utilizar distintos algoritmos, pero todos ellos están centrados en la búsqueda de los vecinos más cercanos. Entre ellos encontramos algunos como: HNSW (*Hierarchical Navigable Small World*), IVF (*Inverted File Index*) o PQ (*Product Quantization*). Por su combinación entre una baja latencia y una muy buena precisión, como ya hemos mencionado anteriormente, el estándar actual es el algoritmo HNSW. Este algoritmo organiza los vectores de los documentos estructurándolos en un grafo de múltiples capas [6]. La capa superior contiene muy pocos nodos, siendo una forma rápida para cruzar grandes distancias en el espacio vectorial. A medida que descendemos, las capas inferiores se vuelven más densas hasta llegar a la capa base, que contiene todos los datos indexados. Cuando entra una consulta, HNSW empieza buscando en la capa superior. Encuentra el nodo más cercano, desciende a la siguiente capa utilizando ese nodo como punto de entrada, vuelve a buscar los vecinos más próximos y repite el proceso hasta tocar fondo [6]. Aquí se encuentra la vulnerabilidad que explotamos. Si el atacante inyecta una consulta cuyo texto exacto (o similar) ya existe dentro de la base de datos, el algoritmo encuentra con mayor rapidez la zona más cercana del grafo al vector consultado. El motor encuentra rápidamente el nodo objetivo porque las distancias matemáticas que debe calcular son mínimas y los caminos en el grafo coinciden de forma directa. Por el contrario, si la consulta contiene información que no existe en el sistema, HNSW necesita explorar muchos más caminos para encontrar el nodo objetivo. Al no encontrar un nodo exacto, el algoritmo realiza cálculos de distancia adicionales entre múltiples nodos de las capas inferiores, intentando localizar los vecinos aproximados más cercanos disponibles para devolver una respuesta aproximada. Esta necesidad de explorar más rutas y realizar más operaciones genera un retraso medible en la respuesta.

Una vez explicada la existencia de esta diferencia temporal, vamos a explicar cómo se puede aplicar de forma ofensiva en los sistemas. En ciberseguridad, los ataques de Inferencia de Membresía (MIA) intentan confirmar o desmentir si un fragmento de información específico está guardado dentro de un sistema [7], [8]. En nuestro caso, el objetivo se encuentra en la base de conocimiento corporativa. Buscamos confirmar si un documento exacto ha sido indexado por el motor de búsqueda o no. La aplicación de este ataque presenta un riesgo de seguridad crítico. Para explicar el impacto de esta vulnerabilidad, se plantea un escenario de amenaza interno en donde un empleado con acceso básico al RAG de la empresa sospecha que existe un documento confidencial sobre unos despidos. Este atacante no tiene permisos de lectura sobre los archivos de recursos humanos pero, sin embargo, puede automatizar consultas al chatbot corporativo enviando fragmentos de texto muy concretos, mezclando términos como despido con fechas o nombres de sus compañeros. Al medir la latencia de cada petición, busca identificar qué combinaciones generan una respuesta más rápida. Si la consulta "despido Carlos Gómez 30 de noviembre" devuelve un tiempo medio de respuesta menor que una consulta inventada al enviarla en repetidas ocasiones, el atacante podría inferir la existencia del documento y a quién afecta, evadiendo por completo las restricciones de acceso de la empresa.

Para garantizar el realismo de la investigación, el atacante simulado debe encontrarse dentro de la propia red corporativa, ya que las API de los sistemas RAG empresariales rara vez están expuestas a internet público, por lo que partimos de la premisa de que la amenaza proviene de un empleado o de un actor que ya ha comprometido la red local. Al operar desde la intranet, el atacante evita el ruido que generaría internet abierto, una variable que hemos decidido dejar fuera del alcance de este trabajo para futuras líneas de investigación. A pesar de esta ventaja, sus capacidades siguen estando muy restringidas para hacerlas lo más realistas posible, ya que no posee ningún tipo de credencial a sistemas de control, desconoce el modelo de *embeddings* que utiliza la empresa para vectorizar el texto, no sabe qué estrategia de fragmentación de texto usa el sistema y no tiene acceso a las métricas de hardware o memoria del servidor. El atacante se limita a mandar consultas a la API del chatbot y medir el tiempo que este tarda en responder, pudiendo registrar el tiempo exacto que transcurre desde que su máquina envía la petición hasta que recibe el primer byte de respuesta (*Time-to-First-Byte*).

6.1.2. IDENTIFICACIÓN Y CONTROL DE VARIABLES

Para este tipo de ataques en el que se mide un canal lateral, es muy importante ser muy precisos en la medición, ya que es la única información que tenemos para valorar el éxito. Por esto, en un inicio el escenario de ataque se plantea aislando las variables importantes para así poderlas clasificar entre variables independientes, dependientes y de confusión y poder asegurar que la distinción de tiempos realmente proviene de la diferencia en la latencia del algoritmo de búsqueda y no de alguna otra variable que no tuviéramos en cuenta.

Las variables independientes son todos aquellos parámetros que el atacante puede manipular para provocar un comportamiento determinado del sistema. La principal de estas variables independientes es el contenido de la query, más en concreto, si la consulta trata de algo que aparezca en el RAG o no. El atacante cambia la finalidad de la prueba dependiendo de si la consulta está rellenada con información completamente ficticia o si esta contiene algo de información que puede aparecer en el RAG, ya que en el primer caso la prueba simplemente mide el tiempo para usarlo como comparación para saber si en el segundo caso la información está o no presente en el RAG.

En este estudio también queremos saber si otras variables manipulables por el atacante influyen directamente en el éxito de la explotación. Por esto, se ha decidido que como variables secundarias se estudien otras dos métricas relevantes para esta experimentación. La primera variable secundaria es la longitud de la consulta. Esta métrica se ha considerado importante, ya que en la fase inicial de cualquier RAG el texto del usuario se tiene que convertir en un vector matemático [2], [3], [28] y enviar una frase de 4 palabras no consume el mismo tiempo que enviar una de más de 20. Por lo tanto, se ha considerado relevante el estudio de esta variable para verificar que el ruido aportado por el tiempo de vectorización no tapa el ataque. Además, nos ayudará a comprobar con qué longitud de queries el ataque es más exitoso.

La última variable independiente secundaria identificada como importante para el estudio es la frecuencia de aparición del dato. Esta es estudiada para entender si un término que se repita en la mayoría de documentos genera un éxito distinto a uno que esté presente, pero no con un número de apariciones tan grande. Con esto, posteriormente podremos optimizar de mejor manera el ataque en la herramienta, sabiendo qué tipo de dato es mejor incluir en las consultas.

Las variables dependientes son las métricas que el atacante mide para poder inferir o descubrir información sobre el sistema. Estas variables no son manipulables por el atacante de forma directa, ya que son generadas por el sistema. En esta investigación, la variable dependiente principal es el Tiempo Total de Respuesta. Esta métrica registra el intervalo completo desde que el cliente lanza la consulta hasta que recibe la respuesta generada por el chatbot. Hemos decidido utilizar el tiempo total porque se busca investigar el peor de los casos para el ataque, pero también el más realista, ya que el tiempo completo en ocasiones es el único medible por el usuario. Esto hace que no solo se mida el proceso de búsqueda en la base de datos vectorial, sino también la transformación del texto en un vector matemático y la fase de generación del LLM. Para mejorar esta medición en la herramienta, se ha realizado una medición de la métrica *Time-to-First-Byte* ya que esta evita en gran medida el ruido aportado por la red. En este caso, esto solo se ha hecho en la herramienta, ya que es en el único sitio en donde se utilizan llamadas HTTP para llamar al modelo de manera realista y útil para las empresas del sector. Igualmente, es importante mencionar que esta variable también es una variable de caja negra que se puede medir en todos los casos en los que se usen llamadas a API.

Aparte de esa variable dependiente, nos dimos cuenta de que había que estudiar cómo afectaban las distintas configuraciones de los sistemas en la efectividad del ataque. Más en concreto, pudimos observar que los tiempos variaban mucho dependiendo del número de vecinos usados como contexto en los prompts del sistema. Por esto, la variable dependiente secundaria estudiada en esta investigación es k, que es el número de vecinos que el sistema de recuperación devuelve para que el LLM los use de contexto en la generación de la respuesta.

Por último, se han debido tener en cuenta las variables de confusión, que son todos aquellos factores externos o no controlados que alteran las respuestas del sistema y pueden distorsionar las mediciones del atacante. En este caso hemos identificado múltiples factores que inyectan ruido en las métricas. El principal, obviamente, son todos los procesos por los que el sistema RAG pasa al procesar una consulta, pero como hemos mencionado anteriormente, estos se han tenido en cuenta en las variables independientes y de esta manera se ha conseguido estudiar de qué forma obstaculizan la medición. Otro factor es el estado del propio sistema, ya que si este está siendo muy utilizado, los picos en la CPU y GPU añaden latencia al proceso. De hecho, estos picos generan valores completamente fuera de las mediciones realizadas en el estudio, ya que generan retrasos muy superiores a las variaciones de milisegundos que se están midiendo. Por esto, se han creado mecanismos estadísticos que quiten todas estas mediciones corrompidas para que no afecten al resultado final. Por último, el uso de cachés en el sistema también puede ser problemático, ya que modifican los valores de latencia por completo [12]-[15], por lo que esto se ha tenido en cuenta para la realización del ataque, observando qué ocurre con distintos tipos de cachés y si el ataque sigue funcionando con ellas.

6.1.3. DISEÑO DEL CORPUS Y BANCO DE CONSULTAS

El entorno diseñado para poder demostrar que la diferencia temporal en la búsqueda de documentos existe tiene que simular un entorno realista, pero manteniendo un control absoluto sobre qué información hay en el sistema. Este entorno se compone de dos elementos: el corpus, que son los documentos estáticos que indexaremos en la base de datos vectorial y el conjunto de consultas, que son las utilizadas para realizar las distintas pruebas de medición temporal.

Los documentos elegidos para formar la base de conocimiento del sistema objetivo no se seleccionaron de manera aleatoria. En un principio, el diseño buscaba replicar los archivos habituales que maneja una empresa, como documentos de procedimientos técnicos, políticas

internas, contratos de confidencialidad y plantillas de informes financieros. Pero, como hemos mencionado antes, se ha decidido estudiar el caso en el que el ataque funcionase peor, ya que creemos que si el corpus trata un tema específico, el ataque será más sencillo, porque todos los documentos estarán indexados muy cerca unos de los otros. Por esto se ha seleccionado un corpus que tuviera muchos temas variados llamado Project Gutenberg o PG. El corpus PG es un dataset que contiene una gran variedad temática al recopilar más de 75,000 obras literarias clásicas en formato plano [29]. Además, para la primera prueba de concepto se seleccionó la forma reducida de este dataset con solo 18 libros.

Como los textos de este dataset tienen distintas temáticas, vocabularios, estructuras y, en general, son gramaticalmente distintos, al convertirlos en vectores con el modelo de *embeddings*, los datos no se concentran en una región temática compacta [28]. Se dispersan de forma desordenada por todo el espacio vectorial y esto obligará al algoritmo HNSW a construir un grafo mucho más complejo. Por lo tanto, si logramos demostrar la diferencia temporal en este entorno, confirmaremos la viabilidad del ataque en escenarios reales.

Para la realización de la batería de pruebas se utiliza un modelo de IA para poder automatizar la creación de pruebas masivas que sigan unas reglas estrictas. Se realizan una cantidad elevada de queries distintas para poder calcular una media de las mediciones realizadas y poder llegar a la conclusión correcta haciendo que el ruido no sea un problema en la medición estadística, ya que si utilizáramos una cantidad baja, cualquier latencia en la velocidad de generación o en la CPU modificaría los datos recogidos, mostrando datos falsos. Además, la cantidad nos ayuda a hacer que el ruido provocado por la generación del LLM sea, de media, similar en todas las consultas, por lo que se diluya y la señal temporal de la base de datos vectorial destaque con claridad.

Una vez hemos entendido que el número de consultas debe ser alto, siempre dentro de un límite razonable para que las pruebas no sean infinitas, la creación de estas queries no se hace de manera aleatoria, sino que, como hemos mencionado anteriormente, tendremos en cuenta las variables independientes especificadas para su desarrollo. Por lo tanto, las queries se plantean como un reflejo de las palabras más utilizadas en el propio corpus de Project Gutenberg, teniendo que realizar un procesamiento previo de todo el texto indexado para analizar el número de apariciones de los sustantivos más utilizados. Elegimos centrarnos específicamente en los sustantivos porque actúan como las palabras clave que estructuran la base de conocimiento en un sistema RAG [30]. Aunque los modelos de embeddings procesan todos los componentes gramaticales por igual, los sustantivos son los conceptos más estables en el espacio vectorial, facilitando el análisis de cómo se indexa y recupera la información crítica. Este procesamiento nos permite aislar elementos y ordenarlos por su volumen de aparición, separando dos categorías: alta_frecuencia, solo con las tres palabras más utilizadas del corpus y media_frecuencia, con varias palabras con un uso medio en el corpus. Para acabar de completar la variable independiente de la frecuencia, se añade una categoría de baja_frecuencia en la que se usan palabras variadas que no deberían aparecer en el corpus por su contexto moderno.

Una vez tenemos las clases de frecuencia realizadas, nos falta aplicar la variable de la longitud de las consultas. Esta variable hemos decidido también separarla en tres clases para poder medir con exactitud qué es lo mejor para el éxito del ataque. En este caso, las clases seleccionadas son: corta, con consultas de 4 a 5 palabras; media, con longitudes de 13 a 15 palabras; y larga, con queries de más de 25 palabras. Esta división creemos que es la mejor, ya que nos ayuda a probar todas las longitudes de consultas para nuestra investigación.

6.1.4. PROTOCOLO DE EJECUCIÓN Y MEDICIÓN

Para que las pruebas que hemos diseñado funcionen correctamente, es muy importante el diseño de un protocolo tanto para la ejecución de las mismas como para la medición de los resultados. Con esto conseguimos que la ejecución de todas las pruebas se haga de la misma forma, además de evitar problemas con los mecanismos de optimización internos de los sistemas.

Para realizar el protocolo de ejecución, es importante conocer a fondo las tecnologías que estamos utilizando, ya que estas tienen implementado de serie algunos mecanismos que hacen que este ataque pueda fallar. En este protocolo, el procedimiento consiste en esquivar estos mecanismos de una forma que pueda hacer el atacante sin conocimiento alguno sobre el sistema. En este caso, el principal mecanismo que tenemos que evitar son las memorias caché de las distintas partes del sistema. Estas memorias, como ya hemos mencionado, se utilizan para la optimización de recursos y en estos sistemas puede haber varias. Por ejemplo, bases de datos como ChromaDB tienen ya implementada de serie una memoria caché interna que agiliza las consultas, ya que si se entregan dos consultas iguales, la segunda no accederá a la base de datos para realizar el proceso de recuperación, sino que simplemente devolverá el resultado cacheado. Estas memorias no son las más problemáticas, dado que si las consultas no son completamente iguales, no generan el mismo vector y, por lo tanto, no aciertan en la caché. Las memorias más problemáticas para el ataque son las que se implementan en muchos sistemas de forma externa a la base de datos. Muchas de estas memorias son cachés semánticas que, como ya se ha mencionado, trabajan con un threshold para que también consultas similares acierten en caché. Para evitar estas memorias no basta con hacer muchas consultas distintas pero similares, ya que también generan muchos hits, sino que además hace falta establecer un protocolo de ejecución para que antes de cada prueba se mezclen las consultas de manera aleatoria. Obviamente, esta solución genera pruebas distintas cada vez que se ejecutan, pero eso no es malo, ya que comprobamos que no es el orden de ejecución lo que genera el vector de ataque. Además, está claro que esta decisión de diseño hace que, si varias consultas similares se envían seguidas, se generen más hits en la caché, pero eso no es problemático, ya que lo único que mostrará es aún más diferencia entre los términos que están en la base de datos y los términos que no están presentes.

Otro proceso que hay que tener en cuenta al hacer pruebas contra estos sistemas es el arranque del motor del modelo de lenguaje. Cuando se realiza la primera consulta al LLM, el tiempo total es mucho mayor, ya que para optimizar todo el proceso hay algunos mecanismos que solo se hacen la primera vez que se activa el sistema. Algunas de las operaciones que se realizan al enviar la primera consulta son: levantar las estructuras internas del modelo, inicializar las matrices de pesos en la VRAM, activar los mecanismos de atención, compilar el grafo de inferencia, entre otros. Además, componentes como la *KV-Cache* necesitan configurarse y reservar espacio físico en el hardware [12]. Si medimos esta primera interacción y la tomamos como un dato más, el pico de latencia provocaría una alteración de las medidas finales. Para evitar esto, en el protocolo de ejecución se incluye una fase de calibración mediante consultas de calentamiento. Antes de iniciar la batería de pruebas oficial, enviamos una pregunta inicial al sistema para que la procese y realice todos los procedimientos de inicio. Posteriormente a esta ya podremos comenzar las pruebas con normalidad.

Por último, para el protocolo de medición de resultados, como se ha mencionado en las variables de confusión, es necesario implementar un mecanismo para descartar los resultados afectados por las interferencias del sistema, ya que la máquina a la que se le ataca puede sufrir picos momentáneos de sobrecarga debido a diversas razones. Para intentar aislar la señal temporal del algoritmo HNSW lo máximo posible frente a estas ralentizaciones, el análisis de resultados necesita un filtrado estadístico posterior a las pruebas. Este se realizará usando un percentil 99 como umbral de corte. Al ordenar las latencias de menor a mayor, el percentil permite establecer el límite superior donde se concentra la inmensa mayoría de las ejecuciones del algoritmo. De este modo, se descartan los picos extremos de latencia que representan únicamente al 1% de los casos más lentos, garantizando que las variaciones no distorsionen las conclusiones sobre la eficiencia real del ataque. Una vez que estos datos están limpiados, podremos proceder al análisis de los resultados con gráficas y comparaciones.

6.2. EJECUCIÓN DE LA INVESTIGACIÓN

Una vez definido el diseño experimental, el siguiente paso fue ejecutar las pruebas necesarias para comprobar si la diferencia temporal podía observarse en distintos escenarios. La investigación no se realizó directamente sobre la herramienta final, sino mediante varios scripts de prueba que permitieron estudiar cada parte del sistema por separado. Primero se validó el comportamiento en un entorno local reducido, después se escaló el corpus en RunPod y, a partir de ahí, se ejecutaron las pruebas principales sobre recuperación vectorial, RAG completo y caché.

6.2.1. VALIDACIÓN LOCAL INICIAL CON PROJECT GUTENBERG

La primera fase de la investigación se realizó en local sobre el corpus reducido mencionado en el apartado anterior. Esta prueba tenía el objetivo de comprobar si *ChromaDB* mostraba las diferencias temporales explicadas en la hipótesis del ataque. Por lo tanto, con esto no se buscaba validar el ataque completo, sino comprobar con una prueba de concepto si la hipótesis ocurría en un entorno reducido o no, para posteriormente escalarlo.

Para construir el entorno, se creó una base persistente de *ChromaDB* en disco. Los documentos se dividieron en fragmentos de 1000 caracteres, ya que era un tamaño suficientemente grande para conservar contexto textual, pero pequeño para generar bastantes puntos dentro del índice vectorial. Después, cada fragmento se insertó en una colección local. En esta primera prueba se usó la función de *embeddings* por defecto de *ChromaDB*, porque permitía crear una prueba de concepto rápida sin tener que cargar manualmente un modelo externo. Estas configuraciones sencillas fueron suficientes para esta primera fase basada en observar el comportamiento del índice HNSW [6], [18]. Además, en el entorno se usaron solo dos tipos de consultas en vez de las tres explicadas anteriormente, por simplificar el ataque centrándolo únicamente en ver las diferencias más que en optimizarlo. Las consultas desarrolladas para esta prueba de concepto fueron pocas por lo mismo.

La medición se hizo aislando la recuperación vectorial. Antes de medir cada consulta, el texto se convertía en vector mediante la misma función de *embeddings* usada por la colección. Después se lanzaba la consulta contra *ChromaDB* y se medía únicamente el tiempo de ejecución de la búsqueda. De esta forma, la prueba no incluía generación de respuesta por parte de un LLM. Esto era importante porque, si se hubiera introducido el modelo desde el principio, el ruido de generación habría tapado la diferencia de respuesta al ser la base de datos tan

reducida. El script ejecutó 300 iteraciones, alternando consultas de acierto y fallo seleccionadas aleatoriamente. En cada iteración se registró el tiempo de respuesta en milisegundos y al final se calcularon las medias de ambos grupos. También se generó una primera visualización de las distribuciones de latencia para comprobar si existía separación entre ellas. Esta fase no se tomó como resultado definitivo, pero sí sirvió para confirmar que había una señal temporal inicial y que el siguiente paso debía ser probar el mismo planteamiento con un corpus mucho mayor y un entorno más profesionalizado.

6.2.2. ESCALADO DEL CORPUS EN RUNPOD

Una vez comprobado el fenómeno en local, el siguiente paso fue escalar el entorno de pruebas. La prueba anterior servía para validar la idea, pero su limitación era que la base de datos era demasiado pequeña para representar un sistema RAG empresarial. Por esto, se decidió construir una colección mucho mayor usando PG-19, que es un conjunto de datos basado en los libros de *Project Gutenberg* [29] y accesible desde *Hugging Face*. Para poder procesar este corpus se usó *RunPod*, un servicio de alquiler de GPUs en la nube, ya que la ingesta completa requería mucha más capacidad de cómputo y almacenamiento que la prueba local. En este entorno se creó una base persistente de *ChromaDB* y se utilizó el modo *streaming* para la ingesta de datos, lo que permitía procesar los libros de forma progresiva sin tener que descargar todo el conjunto de datos antes de empezar.

El proceso de fragmentación se mantuvo igual que en la prueba local, dividiendo los textos en bloques de 1000 caracteres y asignando un identificador incremental con el formato doc_000000000 a cada fragmento, lo que facilitaba después recuperar muestras concretas de la colección para analizarlas. Además, en esta fase ya no se usó la función de *embeddings* por defecto de *ChromaDB*. Se escogió *all-MiniLM-L6-v2*, cargado mediante *SentenceTransformer*, porque es un modelo ligero y suficientemente rápido para generar millones de vectores [28], [31] y así poder crear una base vectorial grande en un tiempo razonable y con un comportamiento estable para las pruebas de latencia.

La inserción se realizó por lotes de 5000 fragmentos, ya que el modelo no podía soportar lotes mayores al ser un modelo pequeño. Con todo esto se consiguió construir una colección de aproximadamente tres millones de fragmentos, con un tamaño en disco cercano a 34 GB. No se utilizó todo PG-19 porque no era necesario para la investigación, ya que con ese volumen de datos ya se conseguía una base suficientemente grande para comprobar si la señal temporal seguía apareciendo en un escenario mucho más exigente que el local.

6.2.3. CARACTERIZACIÓN DEL CORPUS Y CONSTRUCCIÓN DEL BANCO DE CONSULTAS

Con la colección grande ya creada, el siguiente problema fue decidir qué consultas debían lanzarse contra ella. Para que la prueba tuviera sentido, como hemos explicado anteriormente, era necesario conocer qué términos aparecían con mucha frecuencia en el corpus, cuáles tenían una presencia media y qué conceptos podían considerarse ajenos al corpus, para así poder realizar las queries en base a las variables definidas en el diseño de la investigación.

Para hacer este análisis se creó un script específico que extraía una muestra aleatoria de hasta un millón de fragmentos y descargaba los textos en lotes de 5000 documentos para evitar problemas con *SQLite* y con el tamaño de las consultas internas de *ChromaDB*. Después, los textos se procesaban con *spaCy*, una biblioteca de código abierto para el procesamiento de

lenguaje natural diseñada específicamente para el análisis avanzado de datos, usando el modelo *en_core_web_sm*, ya que PG-19 está formado por textos en inglés y era necesario analizar las palabras en ese idioma. Además, se desactivaron los módulos de análisis sintáctico y reconocimiento de entidades para reducir el tiempo de procesamiento, porque la prueba solo necesitaba identificar sustantivos y nombres propios. Con esto se generó un histograma con los sustantivos más frecuentes, que sirvió para elegir los términos de las consultas de prueba. También se realizaron pruebas visuales con *UMAP* [32], un algoritmo de reducción de dimensionalidad que permite visualizar datos complejos en dos o tres dimensiones, sobre una muestra de *embeddings* para observar cómo se distribuían los documentos en el espacio vectorial, lo que nos ayudó a visualizar la distribución de los datos y a verificar que efectivamente este corpus no formaba una única zona temática compacta, sino muchas regiones distintas.

A partir de este análisis, se construyó el banco de consultas, el cual acabó conteniendo 2356 consultas. Dentro de este se mantuvo la división tanto por frecuencia como por longitud explicada en el diseño experimental. Para generarlas se usó un agente de IA al cual se le dieron reglas de formato, términos representativos extraídos del análisis de frecuencia y restricciones sobre la longitud de las frases.

6.2.4. EJECUCIÓN DEL EXPERIMENTO DE RECUPERACIÓN VECTORIAL

La siguiente fase para esta investigación es la ejecución de la prueba en el entorno realista creado. Esta trataba de ejecutar una prueba similar a la ejecutada en local, pero en el nuevo entorno, y tenía el mismo objetivo: observar si la diferencia temporal también podía medirse en un entorno más grande con bases de datos muy pobladas. Por esto, este experimento se diseñó para aislar al máximo la llamada a *ChromaDB* y evitar que otros procesos modificasen los tiempos recogidos.

En esta prueba se trabajó con el archivo Querys.json completo. Además, se tuvo en cuenta el número de documentos recuperados, ya que era importante estudiar su relevancia, como se ha explicado en el apartado 6.1.2, y se probaron 5, 50, 100 y 200 como valores de k. La ejecución se realizó con un script que recorría todas las consultas del banco de pruebas y repetía cada configuración 30 veces para comprobar que las diferencias observadas no dependían de una ejecución puntual, sino de un comportamiento repetido del sistema. De esta forma, los valores medios obtenidos tienen más estabilidad y se reduce el peso de posibles picos aislados de latencia. Antes de iniciar la medición, cada consulta se transformaba en vector mediante el mismo modelo de *embeddings* usado durante la ingesta, lo cual consume tiempo que para este experimento se dejó fuera para aislar correctamente las diferencias temporales del algoritmo HNSW. Si se hubiera incluido dentro del cronómetro, se habría mezclado la latencia del modelo de *embeddings* con la latencia de búsqueda en *ChromaDB*, haciendo más difícil saber si los resultados provenían del sitio esperado o no. Por tanto, la medición se centró únicamente en la llamada de recuperación contra la colección, iniciando el cronómetro justo antes de ejecutar la búsqueda con el vector ya calculado y deteniéndolo al recibir los resultados devueltos por *ChromaDB*. Además, se aplicó el protocolo de ejecución desordenando las consultas de manera aleatoria para generar los menos aciertos posibles en la caché de *ChromaDB* y así poder obtener unos tiempos de medición realistas.

6.2.5. INTEGRACIÓN DEL PIPELINE RAG CON MODELO DE LENGUAJE

Después de ejecutar la recuperación vectorial de forma aislada en una base de datos poblada y sin temática específica, el siguiente paso fue comprobar qué ocurría al introducir un modelo de lenguaje en el proceso. Los sistemas RAG no se utilizan solos en las empresas, ya que en lo que se centran es en devolver el contexto necesario para que un LLM lo interprete y responda a una consulta de un usuario. Por esto, esta fase era muy importante, ya que es en la que observaríamos si el ataque seguía siendo viable y si las diferencias temporales se podían seguir midiendo al haber un LLM por medio. Además, en estas pruebas se midió el tiempo completo de ejecución del pipeline, desde que se lanzaba la consulta hasta que el modelo terminaba de generar la respuesta.

Para realizar esta prueba, se construyó un pipeline RAG mediante *LangChain,* un *framework para desarrollar aplicaciones con modelos de lenguaje*. Se mantuvo *ChromaDB* como base vectorial y se siguió usando *all-MiniLM-L6-v2* como modelo de *embeddings* para conservar la misma representación vectorial que en las pruebas anteriores. La diferencia principal fue que, después de recuperar los fragmentos más cercanos, estos se introducían como contexto dentro de un *prompt* que era enviado al modelo de lenguaje. De esta forma, la medición ya no representaba solo el tiempo de búsqueda, sino el tiempo total necesario para ejecutar la consulta dentro de un sistema RAG completo.

La primera versión de esta prueba se realizó con el modelo *Qwen2.5-7B-Instruct* [33]. Se escogió este modelo porque tenía un tamaño grande, pero seguía siendo posible ejecutarlo dentro del entorno de pruebas. En esta fase se probaron los valores de k: 5, 20, 50 y 100, descartando k=200 porque generaba contextos demasiado grandes para esta fase del experimento. El problema que apareció en esta prueba fue que el modelo introducía una cantidad de latencia muy alta en comparación con la recuperación vectorial. Esto no invalidaba el ataque, ya que seguía apareciendo, pero sí hacía que la señal temporal fuese más difícil de observar en el tiempo total. Además, el modelo consumía demasiado en el entorno de pruebas, lo que hacía que este se ralentizase y generase valores irregulares, provocando picos de latencia muy altos y prolongados. Por este motivo se realizó una segunda prueba con *SmolLM2-135M-Instruct* [34]. Este modelo es mucho más pequeño y tiene un tiempo de generación menor, por lo que permitía estudiar el mismo pipeline RAG con menos ruido añadido por el LLM. En este caso se probaron los valores de k: 5, 10, 25 y 50, que son más bajos, ya que el objetivo era mantener el proceso de generación controlado y evitar que el contexto recuperado fuese demasiado grande. Además, estos valores más pequeños nos acercaban a valores de k que los sistemas reales utilizan. Los sistemas suelen usar una k=5, pero los tamaños de documentos suelen ser del doble de tamaño que en nuestro sistema, ya que los modelos de embeddings que se suelen implementar tienen como máximo número de tokens 512, mientras que nuestro modelo *all-MiniLM-L6-v2* tiene un máximo de 256, por lo que una k=5 en los sistemas reales es similar a una k=10-20 en nuestro sistema, ya que no todos los documentos insertados en el RAG ocupan por completo el máximo número de tokens.

A diferencia de la recuperación vectorial aislada, en este escenario no se realizaron 30 repeticiones por configuración. La introducción del modelo de lenguaje aumentaba mucho la duración de cada prueba al generar una respuesta completa. Por esto, se decidió realizar dos ejecuciones completas por configuración. Con estas dos ejecuciones se podía comprobar si el comportamiento observado se repetía en una segunda medición, sin convertir el experimento en una prueba demasiado larga para el entorno disponible.

6.2.6. EJECUCIÓN DEL EXPERIMENTO CON CACHÉ

La última fase de la investigación consistió en comprobar qué ocurría al introducir mecanismos de caché dentro del pipeline RAG. Muchos sistemas empresariales incorporan cachés para reducir tiempos de respuesta y coste computacional. Sin embargo, como se ha explicado en el estado del arte, estas cachés también pueden generar nuevos comportamientos temporales que un atacante podría medir. Por esto, después de probar la recuperación vectorial aislada y el pipeline con LLM, se decidió repetir el experimento añadiendo distintas configuraciones de caché.

Para esta prueba se mantuvo el pipeline RAG creado con *LangChain* y se usó *SmolLM2-135M-Instruct* como modelo de lenguaje. Este experimento comparó tres tipos de comportamiento. Primero se ejecutó el sistema sin caché, ejecutando la misma prueba que en la anterior fase. Después se activó una caché exacta en memoria mediante la caché propia de *LangChain*, que solo devuelve un acierto cuando la entrada coincide con una consulta ya procesada. Por último, se probó una caché semántica con *Redis*, una base de datos en memoria de alta velocidad, usando *Redis Stack,* una extensión que añade modelos de datos avanzados, y el módulo *RediSearch*, que permite crear índices de búsqueda y realizar consultas por similitud vectorial sobre los *embeddings* almacenados, ya que este tipo de caché no necesita que las consultas sean idénticas, sino suficientemente parecidas en el espacio vectorial. Además, en estas pruebas se configuraron varios umbrales de similitud: 0.95, 0.90, 0.85 y 0.80. Estos valores se escogieron para observar cómo cambiaba el comportamiento del sistema cuando la caché era más estricta o más permisiva. Un umbral alto exige que la consulta nueva sea muy parecida a una anterior para generar un hit. Un umbral menor permite que más consultas entren en caché, pero también puede provocar aciertos con preguntas que no son tan similares. Con cada configuración se probaron los valores de k 5, 20, 50 y 100 y para cada valor de k se recorrió el banco completo de 2356 consultas repitiendo el proceso dos veces y manteniendo el mismo criterio que en las pruebas con LLM, para el número de repeticiones por el coste temporal de cada ejecución completa.

6.3. CRITERIOS DE ANÁLISIS DE RESULTADOS

Antes de presentar los resultados obtenidos, es necesario aclarar cómo se van a interpretar. En esta investigación no basta con observar si una consulta concreta ha sido más rápida o más lenta que otra, ya que las latencias pueden verse afectadas por muchos factores puntuales del sistema. Por esto, los resultados se analizarán buscando patrones repetidos entre grupos de consultas.

El criterio principal será comprobar si las consultas asociadas a mayor presencia en el corpus muestran tiempos menores que las consultas asociadas a baja presencia, que es la relación esperada según la hipótesis planteada. Si el grupo de alta frecuencia responde de forma más rápida y ese comportamiento se mantiene en varias configuraciones, se considerará que existe evidencia a favor del canal lateral temporal, pero si la diferencia aparece solo en una configuración concreta o depende de unos pocos valores extremos, el resultado se interpretará con más prudencia. Además, se tendrá en cuenta la separación entre distribuciones, ya que una diferencia de medias puede ser útil, pero no suficiente por sí sola si las distribuciones se solapan demasiado, porque el ataque sería más difícil de aplicar aunque existiera diferencia estadística. Por esto, en los resultados se observarán tanto las métricas agregadas como la forma de las gráficas, dando más valor a los casos donde la separación entre grupos sea visible y se repita de forma consistente.

Por otro lado, el p-value se usará como apoyo estadístico, pero no como único criterio de decisión. Un valor bajo ayuda a defender que la diferencia entre grupos no se debe únicamente al ruido de la muestra, pero, sin embargo, este valor se interpretará junto con las gráficas y con la magnitud real de la diferencia temporal, lo que es importante porque una diferencia puede ser estadísticamente significativa y, aun así, ser poco útil si los tiempos están demasiado próximos entre sí.

Los resultados se estudiarán por escenarios. Primero se analizará la recuperación vectorial aislada, porque es donde se observa de forma más directa el comportamiento de la base de datos vectorial. Después se estudiará el pipeline RAG con modelo de lenguaje, donde aparece ruido adicional por la generación de respuestas. Por último, se revisarán las pruebas con caché, ya que estos mecanismos pueden modificar mucho los tiempos y cambiar la forma en la que aparece la señal temporal. Con este criterio, el análisis intenta comprobar que existe una diferencia temporal repetible entre grupos de consultas y si esa diferencia se mantiene cuando se añaden componentes más cercanos a un sistema RAG real.

6.4. RESULTADOS EXPERIMENTALES Y DISCUSIÓN

Una vez definidos los criterios de análisis, en este apartado se presentan los resultados obtenidos durante la experimentación. En este caso se muestran aquellas gráficas que permiten interpretar mejor si la hipótesis planteada se cumple en cada escenario, por lo que se seguirá el orden de ejecución de la investigación: primero la recuperación vectorial aislada, después la integración con un pipeline RAG completo, posteriormente las pruebas con caché y finalmente una comparación global entre los distintos escenarios. Además, la parte de las pruebas locales se mostrará de forma resumida, ya que, aunque llevó bastante tiempo, al ser el primer encuentro con este tipo de sistemas, solo se utilizó para comprobar que el ataque aparecía en una situación muy sencilla.

El estudio de los resultados por fases es clave, ya que cada uno nos mostrará avances del ataque implementando partes importantes de un sistema real en cada salto. Por ejemplo, en la recuperación vectorial aislada, se estudia únicamente el comportamiento de ChromaDB al buscar vecinos en la colección, lo que nos mostrará que el ataque existe, confirmando la hipótesis, mientras que en el pipeline RAG con modelo de lenguaje y en las pruebas con caché, el ruido provocado por elementos de un sistema real confirmará que el ataque no solo existe, sino que es efectivo sobre sistemas reales de gran escala, por lo que no tendría sentido mezclar todos los resultados en una única comparación.

6.4.1. RESULTADOS EN RECUPERACIÓN VECTORIAL AISLADA

La primera parte de los resultados se centra en la recuperación vectorial aislada, la cual es especialmente importante porque elimina el ruido del modelo de lenguaje y permite estudiar si la diferencia temporal aparece directamente en el motor de búsqueda vectorial. Dentro de esta fase hubo dos momentos distintos: una validación local reducida con una colección pequeña de Project Gutenberg y una ejecución posterior sobre el corpus ampliado de PG-19 en RunPod. La primera sirvió para comprobar si el fenómeno aparecía en un entorno controlado, mientras que la segunda permitió estudiar si la señal seguía existiendo al aumentar mucho el tamaño de la base vectorial.

En la validación local, como ya se ha explicado anteriormente, se trabajó con una colección reducida construida a partir de 18 libros de Project Gutenberg, ya que esta prueba solo buscaba

comprobar si la hipótesis tenía sentido antes de invertir tiempo en crear una base de datos mucho mayor. La Fig. 9 muestra los resultados conservados de esta validación local, en la que puede observarse que la separación no es tan grande, ya que localmente y con un corpus pequeño la recuperación vectorial es mucho más rápida, pero igualmente se puede intuir que el ataque sí aparece al observar diferencia entre las consultas cortas de alta frecuencia y cortas de baja frecuencia. Este tipo de gráfica es el que usaremos para todas las comparaciones de latencia, por lo que es preciso entender cómo funciona. En el eje horizontal aparecen las combinaciones de frecuencia y longitud que forman los distintos grupos de las consultas. Para cada combinación se muestran varias barras, una por cada valor de k probado. El eje vertical representa la latencia media en milisegundos. Por tanto, una barra más alta indica que ese grupo de consultas tardó más, de media, en recuperar los resultados. De esta forma, las k más altas deben dar por norma un mayor valor de media, ya que el sistema tiene que hacer una recuperación de más documentos. Además, es preciso explicar que la comparación en esta gráfica se puede realizar centrada en una u otra variable. Por ejemplo, si queremos comparar los grupos por frecuencia, se deben mirar los grupos de High o alta frecuencia, Medi o media frecuencia y Low o baja frecuencia, que compartan la misma longitud de consulta. De esta forma se compararía la media del grupo High/Short con la del grupo Medi/Short y con la del grupo Low/Short, pero no con otros grupos que tengan distinta longitud, como podría ser el grupo Low/Long. Pero si se necesita comparar por longitud de consulta, nos centraríamos en comparar grupos que tengan distinta longitud y la misma frecuencia. Por lo tanto, esta representación permite comparar de manera rápida si las consultas de baja frecuencia son más lentas que las consultas con otras frecuencias y si esa diferencia aumenta al incrementar el número de vecinos recuperados o el tamaño de la consulta.

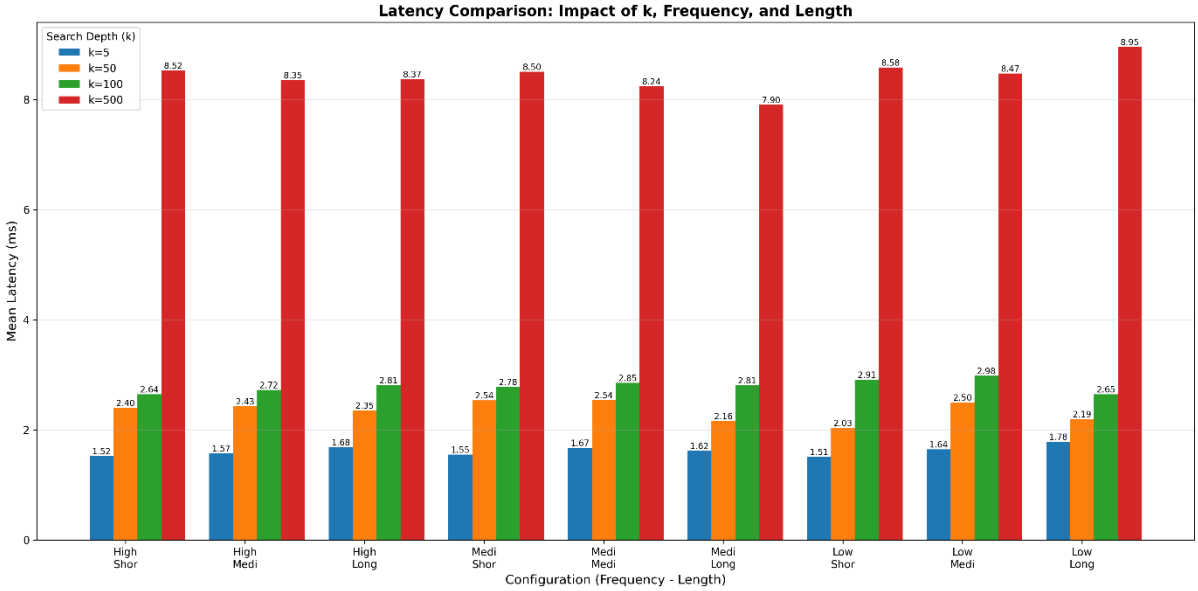


Fig. 9. Comparación de latencias medias en recuperación vectorial aislada en ejecución local simplificada.

En esta primera prueba también se vieron algunos comportamientos que se confirmaron posteriormente con las pruebas en sistemas realistas. En este caso, como se observa en la gráfica, cuando los valores de k son demasiado altos, el sistema ya no funciona como se espera, ya que al recuperar tantos documentos, este acaba devolviendo muchos documentos de ruido, tanto si la consulta tiene alta frecuencia como si tiene una baja frecuencia, por lo que el tiempo de recuperación es más aleatorio y menos determinista. Por otra parte, también vemos lo que ya se ha comentado anteriormente sobre los picos de sobrecarga o variabilidades del sistema,

lo que hace que no siempre las medias de todos los grupos vayan acordes a lo esperado, pudiendo tener algún grupo donde haya afectado más el ruido de la máquina.

Habiendo conseguido estos resultados en los que se podía observar mínimamente la hipótesis planteada, se repitió la prueba sobre la colección ampliada en RunPod. En este caso ya se usó el banco completo de consultas y una base vectorial de aproximadamente tres millones de fragmentos. A partir de este punto, los resultados sí se consideran la evidencia principal de que la hipótesis planteada es real, ya que el entorno se acerca mucho más a un sistema RAG con una base de conocimiento grande. Además, en los siguientes apartados de los resultados se añaden partes importantes del sistema para acabar probando un sistema completo y profesional. La Fig. 10 muestra la comparación global de latencias medias para todas las combinaciones de frecuencia, longitud y valor de k.

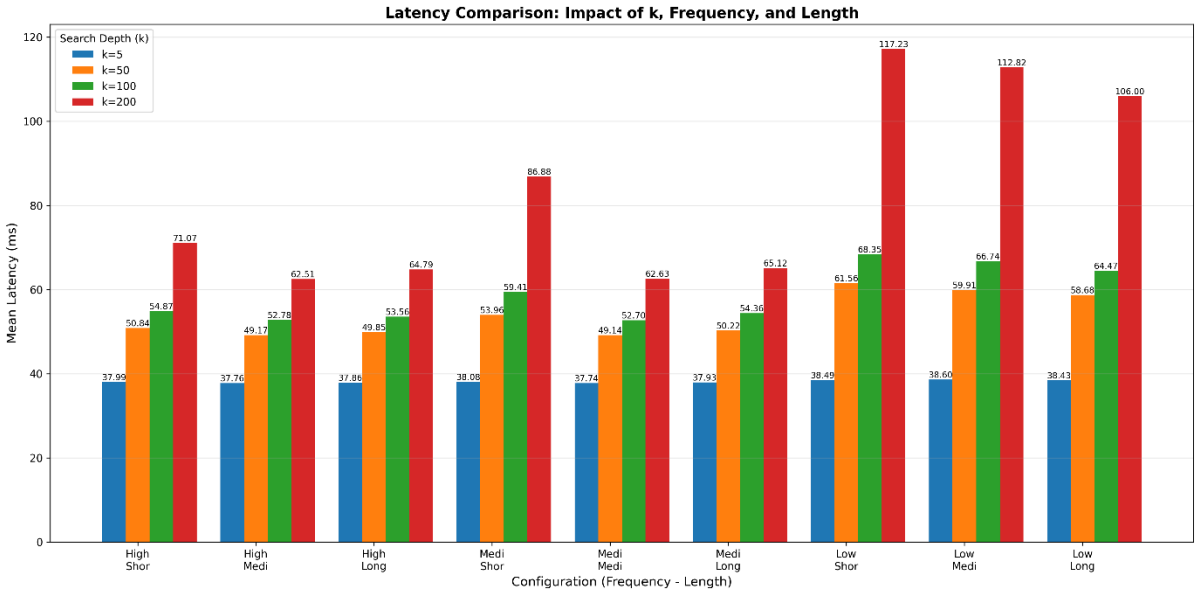


Fig. 10. Comparación de latencias medias en recuperación vectorial aislada en ejecución completa en RunPod.

Esta figura es la más importante de este apartado porque resume el comportamiento general del experimento y nos deja observar y comprobar algunos comportamientos de los sistemas RAG.

Con k=5, las diferencias entre grupos son reducidas, ya que, aunque el orden de las medias coincida con la hipótesis, la separación es pequeña y difícil de explotar de forma fiable, lo cual tiene sentido porque con un valor de k tan bajo el sistema devuelve pocos vecinos y el recorrido de búsqueda es menor. Al aumentar la k, la diferencia empieza a crecer, ya que, por ejemplo, para k=50, la diferencia entre las medias de alta y baja frecuencia en grupos como el de consultas de longitud corta es de más de 10 ms y en k=100, la separación incluso aumenta un poco, siendo mayor a 13 ms. En este caso se decide probar como k más grande, k=200 en vez de k=500, ya que aunque ambos sean magnitudes desproporcionadamente grandes, k=500 tardaba demasiado con pruebas con el corpus completo, mientras que k=200 ofrecía un comportamiento más realista al no devolver tanto ruido y, de hecho, aunque hay algún pico de sobrecarga como en el grupo de longitud corta y media frecuencia, el comportamiento de k=200 aumenta mucho más la diferencia entre los grupos de baja frecuencia y el resto, llegando a superar los 45 ms. Por todo esto podemos afirmar que el valor de k tiene una influencia directa en la visibilidad del canal lateral, ya que con k demasiado pequeñas, el coste temporal entre consultar una zona muy poblada o una zona poco relacionada del espacio vectorial es muy

pequeño, mientras que cuando se obliga al sistema a devolver más vecinos, el recorrido de búsqueda y la recuperación de resultados aumentan, y las consultas de baja frecuencia parecen verse más afectadas. Esto es un punto muy bueno, ya que los sistemas profesionales, como ya hemos comentado, suelen utilizar una k en el rango donde estamos viendo la distinción, evitando usar k demasiado bajas, por su poco contexto para la respuesta, ni demasiado altas, por su alto tiempo de recuperación.

Por otro lado, en esta prueba se observa que la longitud de la consulta no tiene un efecto tan determinante como la frecuencia de la misma, al funcionar el ataque bien con los tres tipos de longitudes. Aunque es verdad que se puede observar una diferencia mayor en las consultas de longitud corta, esta variable es más útil cuando la generación de un LLM entra en juego, ya que una consulta más larga puede hacer que el prompt sea demasiado largo, introduciendo un ruido mayor en la generación de la respuesta del modelo y tapando el ataque real de recuperación vectorial.

Estas observaciones se ven con claridad en el gráfico presentado, pero además hay otros, como los mapas de calor o los diagramas de cajas, que también pueden ayudar al análisis de los fenómenos observados. Los mapas de calor se usan para representar la misma información de una forma más compacta. En estas gráficas, las filas indican la frecuencia de la consulta y las columnas indican su longitud. Cada celda contiene la latencia media de esa combinación y el color muestra si el valor es más bajo o más alto dentro de la configuración analizada, representando los tiempos menores con tonos más claros y los mayores con tonos más intensos y oscuros, lo que facilita ver si existe un patrón por filas, que en este caso sería lo esperado si la frecuencia tiene más peso que la longitud. En estos diagramas es importante fijarse en la leyenda que se muestra a la derecha para entender el rango de valores estudiado en cada uno, ya que la diferencia de colores puede confundir. A continuación, se puede visualizar la Fig. 11, donde se muestran los distintos mapas de latencia de las distintas k utilizadas en este experimento.

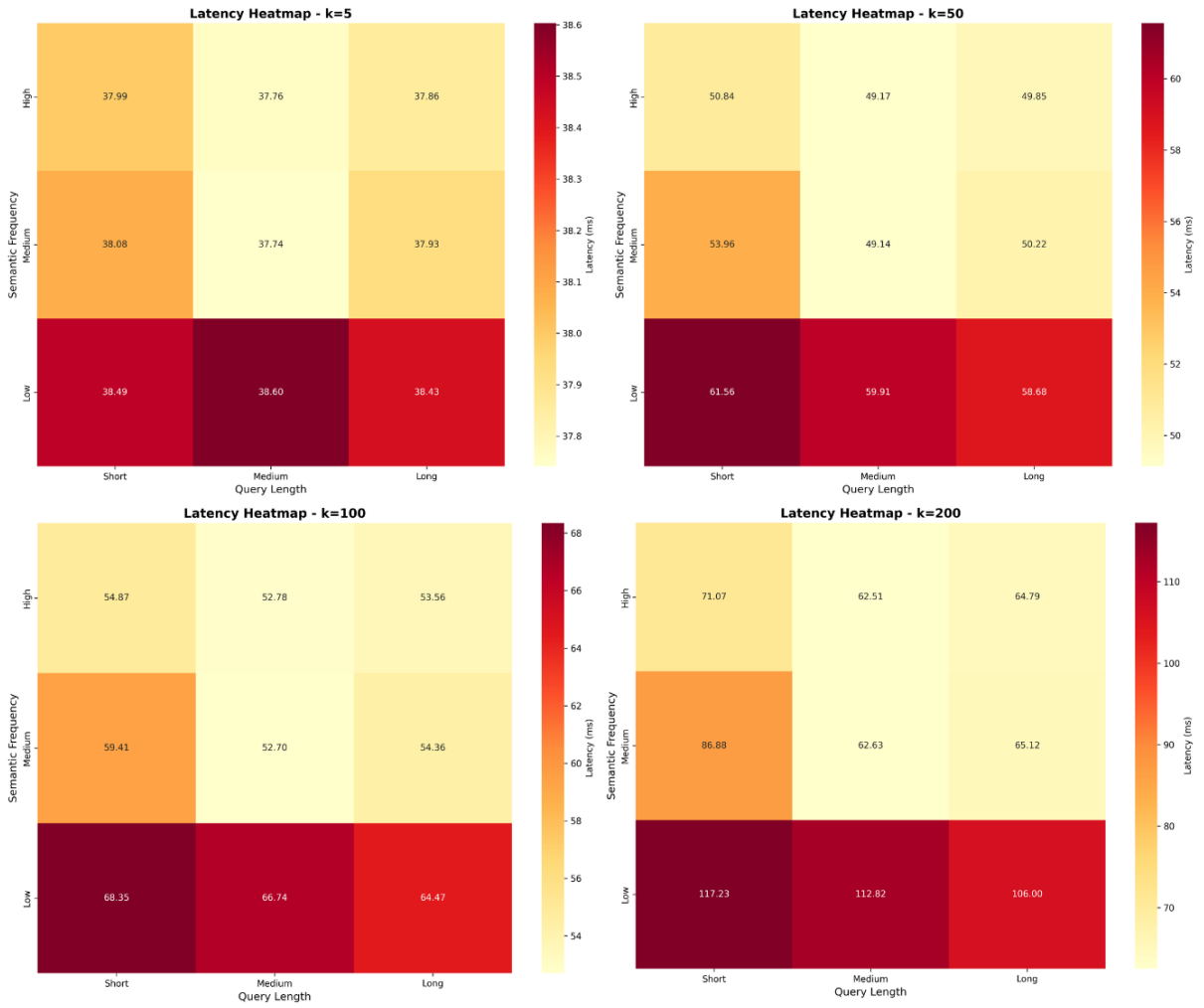


Fig. 11. Mapas de calor de latencias medias en recuperación vectorial aislada con k=5, 50, 100 y 200.

Lo interesante de estos mapas es observar cómo la diferencia aumenta mucho más entre filas que entre columnas, mostrando que la diferencia por frecuencia de aparición es mucho mayor que la de la longitud de la consulta. Además, comparando las leyendas de las gráficas, también se observa que las diferencias de medición son mucho mayores cuanto mayor es la k utilizada, ya que, por ejemplo, en k=5 el rango va de 37,9 a 38,6 ms, mientras que en la gráfica de k=200 encontramos un rango mucho más amplio que va desde 60 hasta 120 ms.
También con esta gráfica nos podemos dar cuenta de que las consultas que usan términos de media frecuencia se comportan más similarmente a las de alta frecuencia que a las de baja frecuencia, ya que lo más importante es que por lo menos el término buscado aparezca con algo de frecuencia en el corpus y no tanto si el término tiene un ratio de aparición altísimo. Esto favorece mucho al ataque, ya que si solo se diferenciasen los términos con altísima frecuencia de aparición, el ataque perdería utilidad al solo poder buscar la información más general, pero de esta forma el ataque puede centrarse en términos más específicos que también aparecen, pero no de una manera demasiado exagerada.

Como hemos mencionado antes, además de estas gráficas, las gráficas de cajas son muy útiles para este análisis al ayudarnos a estudiar la dispersión de los datos y la separación entre los grupos. En estos gráficos, la caja representa la zona central donde se concentra buena parte de los valores, la línea interior marca la mediana y los puntos que aparecen fuera muestran valores extremos. Esta gráfica es útil porque permite comprobar si las diferencias entre grupos se encuentran solapadas, lo que dificultaría mucho el ataque al no saber clasificar correctamente

las consultas, o si no lo están, facilitando en gran medida el ataque al necesitar menos mediciones para su funcionamiento. La Fig. 12 mostrada a continuación muestra los diagramas de cajas de la k menor y mayor utilizada en esta prueba.

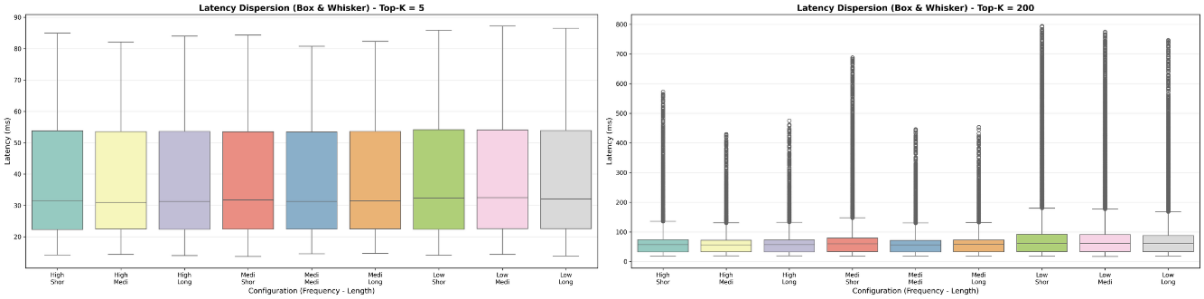


Fig. 12. Diagramas de cajas de las latencias en recuperación vectorial aislada con k=5 y 200.

En este caso podemos observar que, incluso en la prueba con mayor k, el solapamiento entre los distintos grupos es casi completo. Esto no es algo bueno para la investigación, ya que puede hacer que, cuando intentemos hacer un ataque con menor número de consultas, estas se acaben clasificando mal. En cualquier caso, esto no rompe la investigación, ya que, como vamos a ver, este solapamiento se reduce en experimentos posteriores como el que usa el LLM, pero si se mantuviese de esta forma, podría indicar un problema al profesionalizar las pruebas, ya que se necesitarían demasiadas consultas para clasificar los términos. Además, vemos que en la medición con k=200 aparecen muchos valores considerados extremos que se generan porque esta prueba mide una operación muy pequeña y muy sensible al estado interno del sistema. Al medir solo la llamada de búsqueda contra ChromaDB, el tiempo normal se mueve en decenas de milisegundos, por lo que cualquier pausa puntual del proceso, acceso a memoria, gestión interna de ChromaDB, intervención del sistema operativo o variación en la caché aparece como un pico muy grande respecto al tiempo base, por lo que, al aislar tanto la medición, también se hacen mucho más visibles las interrupciones internas del entorno.

Esta interpretación se observa mejor en la gráfica de densidad, ya que se observa cómo la mayor parte de las mediciones se concentra en la zona baja de latencia, pero aparece una cola larga hacia la derecha, especialmente en las consultas de baja frecuencia. Las gráficas de densidad permiten observar la forma de la distribución de latencias, ya que en lugar de mostrar cada medición individual, representan dónde se concentran los valores de cada grupo. Cuando una curva está más desplazada hacia la derecha, significa que ese grupo tiende a generar tiempos mayores. Si una curva tiene una cola larga hacia la derecha, indica que aparecen más mediciones lentas y valores extremos. La Fig. 13 muestra las densidades superpuestas para k=200, separadas por longitud de consulta.

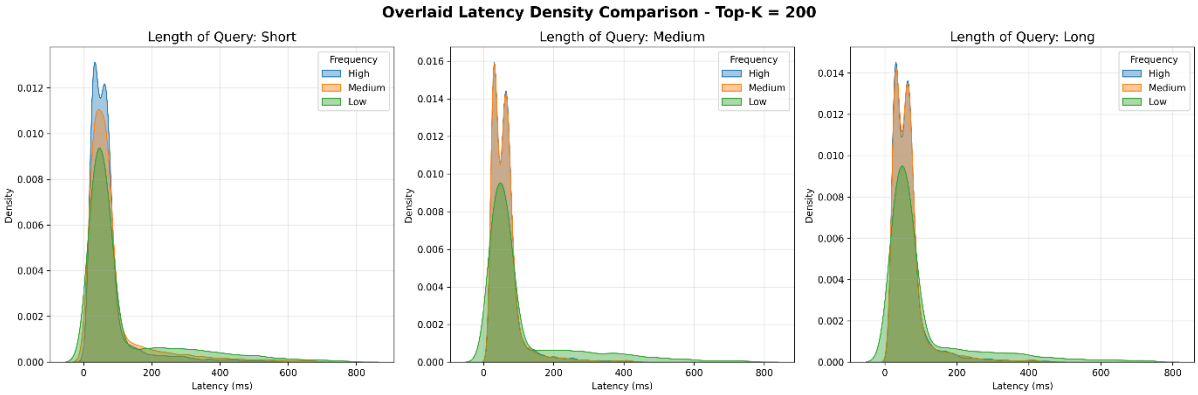


Fig. 13. Densidades superpuestas de latencia para alta, media y baja frecuencia en recuperación vectorial aislada con k=200.

6.4.2. RESULTADOS EN PIPELINE RAG CON MODELO DE LENGUAJE

Después de comprobar que la diferencia temporal sí aparece al medir únicamente la recuperación vectorial, el siguiente paso fue estudiar si esa señal seguía siendo visible al introducir un modelo de lenguaje dentro del pipeline. Esta prueba ya imita a algunos sistemas reales, los cuales, al acabar la búsqueda vectorial, rellenan un prompt de sistema con la consulta y los documentos obtenidos como contexto y se lo envían a un LLM, el cual genera una respuesta basándose en el contexto obtenido de la base de datos, por lo que la medición deja de representar únicamente el tiempo de búsqueda vectorial y pasa a representar también el tiempo de generación del modelo de lenguaje. En este caso, esta prueba solo representa a algunos sistemas reales, ya que hay muchos que aplican sistemas de caché para optimizar su funcionamiento, pero esto no es algo que se haga en todos los casos, al ser algo más costoso de desarrollar y en muchos casos innecesario. Por otro lado, como se explicó en el apartado 6.2.5, al pasar al pipeline RAG completo se redujeron los valores de k respecto a la recuperación aislada, ya que cada documento recuperado aumenta el contexto entregado al LLM y, por tanto, el tiempo de generación de la respuesta.

La introducción del modelo de lenguaje en la medición de la latencia hace que la prueba sea más difícil para el ataque, ya que la recuperación vectorial genera diferencias de milisegundos, mientras que la generación del modelo puede añadir cientos o miles de milisegundos a la prueba. Si el tiempo de generación es demasiado alto o demasiado variable, la señal de la base de datos puede quedar escondida dentro del tiempo total, por lo que en esta fase se decidió probar dos modelos distintos. Primero se usó *Qwen2.5-7B-Instruct* [33], un modelo grande que permitía estudiar un pipeline más pesado y después este se sustituyó por *SmolLM2-135M-Instruct* [34], mucho más pequeño, para comprobar si al reducir el coste de generación la señal temporal volvía a ser más visible.

Con el modelo *Qwen2.5-7B-Instruct* se hicieron varias pruebas para ver si los resultados obtenidos eran estables, ya que, como se observa en la Fig. 14, en la primera ejecución se pudo ver que la diferencia temporal seguía apareciendo, pero que el coste del modelo dominaba por completo la medición, haciendo la diferencia temporal muy pequeña y susceptible a pequeños cambios.

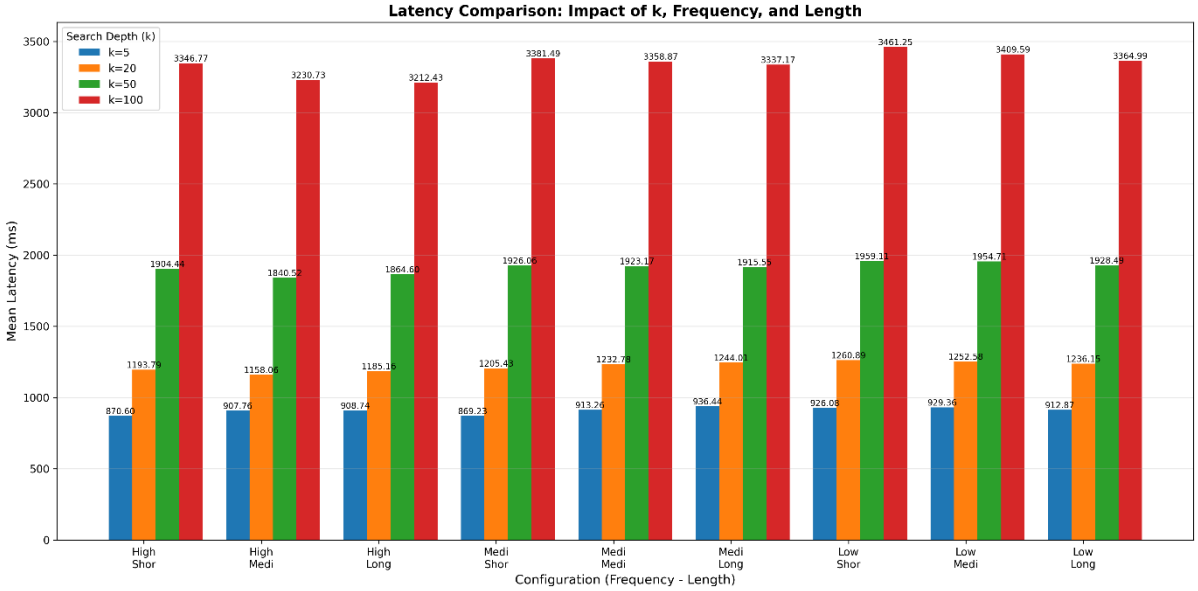


Fig. 14. Comparación de latencias medias de la primera ejecución en pipeline RAG completo usando Qwen2.5-7B-Instruct.

En esta primera ejecución se puede ver que las consultas de baja frecuencia suelen quedar por encima de las de alta frecuencia, pero esa diferencia es mínima comparada con el coste de la generación, haciendo que, por ejemplo, con k=100, todas las categorías se muevan alrededor de los 3,2 y 3,4 segundos. Esto indica que el tiempo de respuesta está muy condicionado por la cantidad de contexto que se entrega al modelo y por el propio proceso de generación.

Sabiendo esto, se realizó una segunda ejecución con Qwen que confirmó este problema. Como se puede observar en la Fig. 15, los tiempos absolutos cambian respecto a la primera ejecución y algunas combinaciones presentan comportamientos poco estables al variar respecto a otros grupos.

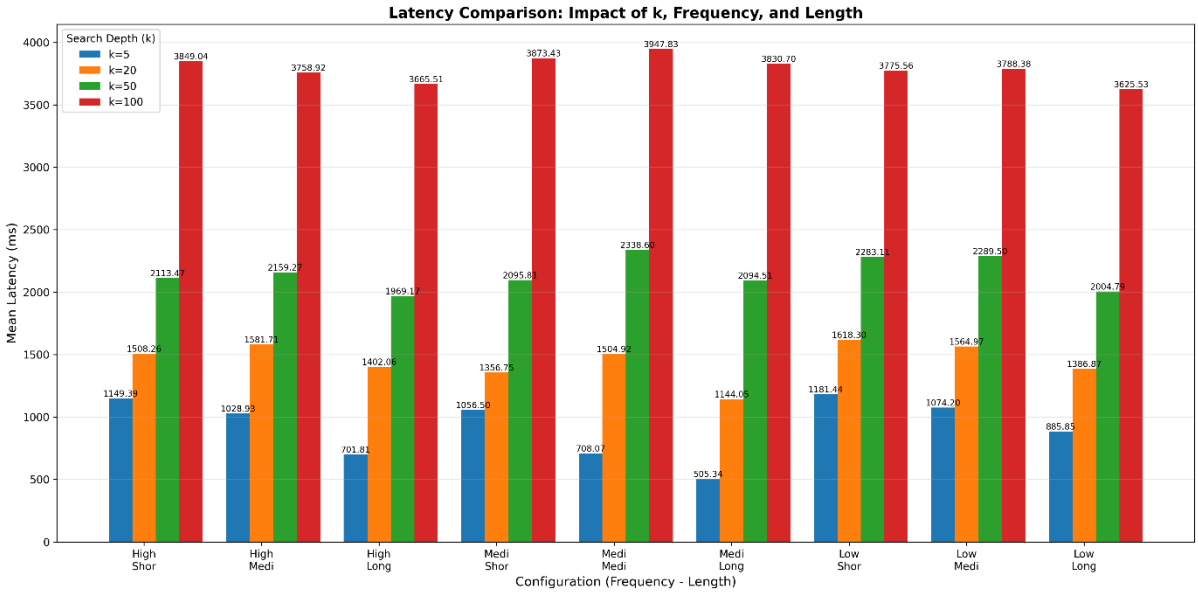


Fig. 15. Comparación de latencias medias de la segunda ejecución en pipeline RAG completo usando Qwen2.5-7B-Instruct.

En este caso, la diferencia temporal no desaparece por completo, pero queda muy condicionada por el LLM, ya que el tiempo añadido por la generación es tan grande que hace más difícil

distinguir si una latencia mayor viene de la recuperación vectorial o de la respuesta generada por el modelo. Por esto, se decidió repetir el experimento con un modelo más pequeño, manteniendo todas las configuraciones de la misma forma, pero reduciendo los valores de k utilizados al no tener sentido el uso de valores muy altos con modelos tan pequeños. En la Fig. 16 se muestran los resultados obtenidos con *SmolLM2-135M-Instruct* con valores de k 5, 10, 25 y 50.

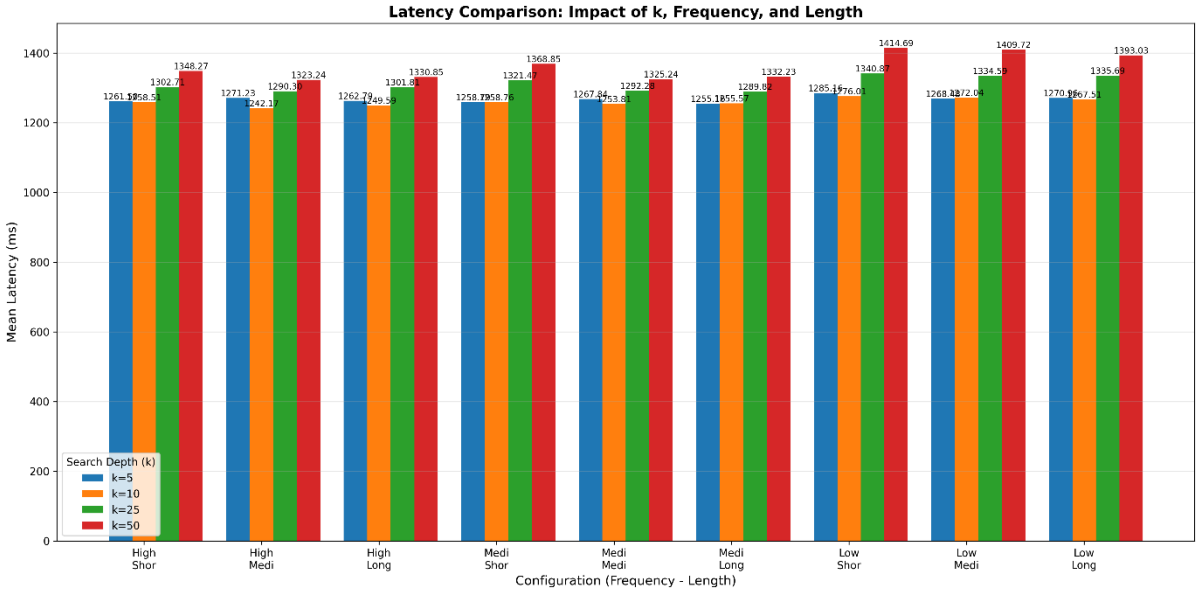


Fig. 16. Comparación de latencias medias en pipeline RAG completo usando SmolLM2-135M-Instruct.

En esta comparativa, la prueba vuelve a funcionar de una forma muy estable al obtener resultados muy similares y ordenados cada vez que se ejecutó. En ella se observa que, de nuevo con valores de k en un rango normal como k=25 o k=50, la frecuencia baja vuelve a quedar por encima de la frecuencia alta con diferencias suficientemente grandes. En el caso de k=25, diferencias de alrededor de los 40 ms y para k=50 de alrededor de los 80 ms, lo que demuestra que la señal temporal de la recuperación puede propagarse hasta la respuesta completa del sistema cuando el modelo de lenguaje no introduce un ruido excesivo.

Además, en esta prueba se observa que la variable de longitud, cuando hay un LLM por el medio, sí que tiene algo más de importancia, observando diferencias algo más notables entre los diferentes grupos que usan consultas cortas y no tanto en los de longitud media o larga. Esto ocurre, ya que cuando la consulta pasa de cierto número de palabras, empieza a aportar demasiado ruido en el prompt, haciendo que el LLM tarde demasiado y difumine algo más la diferencia de recuperación vectorial. En el mapa de calor de k=50, mostrado en la Fig. 17, se permite ver este comportamiento de forma más directa, pudiendo observar cómo en la columna de consultas cortas hay más diferencia de colores entre los grupos tanto de alta y baja frecuencia como de media frecuencia, mientras que en las columnas de las otras longitudes la diferencia solo es notable en el grupo de baja frecuencia.

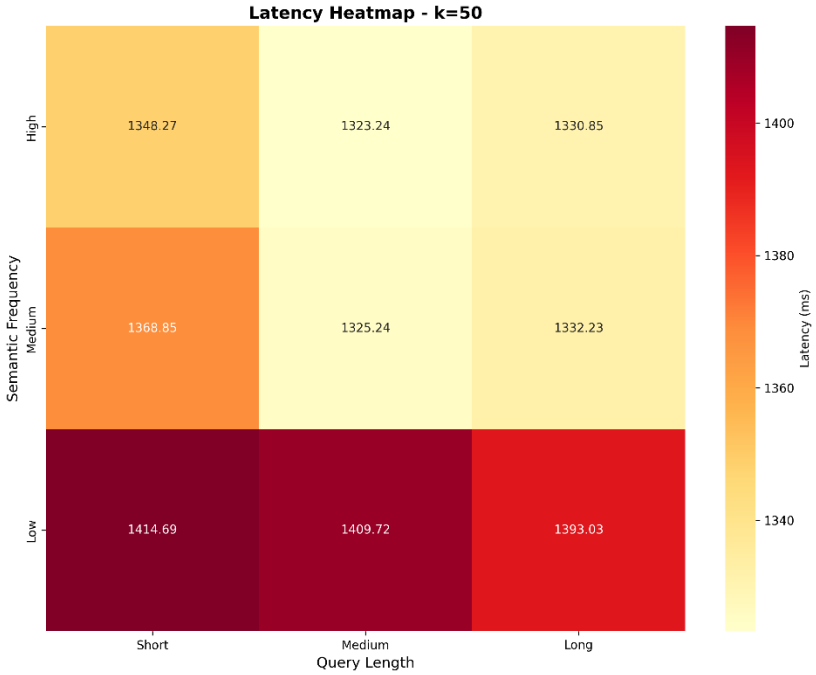


Fig. 17. Mapa de calor de latencias medias en pipeline RAG con SmolLM2-135M-Instruct y k=50.

Además de esta útil observación, en esta prueba se estudió un comportamiento que ayudaba mucho al ataque realizado, ya que se vio que los grupos de consultas en este caso no se encontraban tan solapados y que realmente había una diferencia útil para aplicar una clasificación real. Este solapamiento, como hemos explicado en el apartado anterior, ocurría porque, al estudiar solo un proceso del sistema, cada mínima modificación de este generaba un ruido muy alto para la prueba, pero al introducir un LLM por medio que genera valores de ruido mucho mayores a estos, los valores se estabilizaban, no generando tantos valores extremos y, por tanto, generando un desplazamiento de los grupos. En la Fig. 18 se muestran varios diagramas de cajas con SmolLM2 que muestran cómo, con valores de k en un rango real, los grupos no se encuentran demasiado solapados.

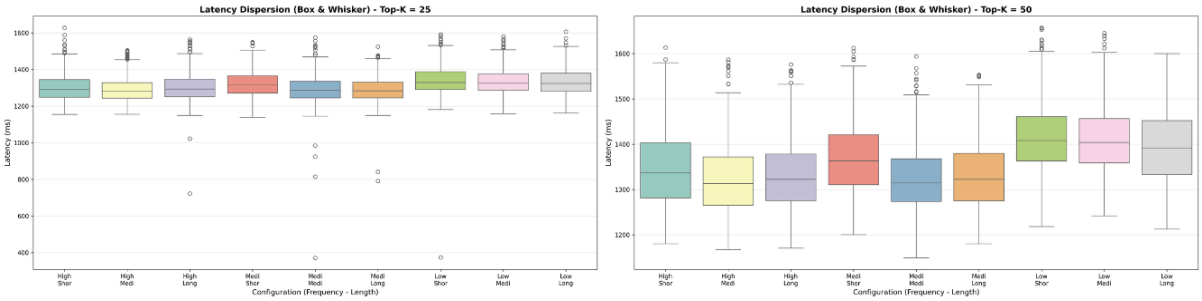


Fig. 18. Diagramas de cajas de latencias en pipeline RAG con SmolLM2-135M-Instruct y k=25 y 50.

Los resultados de esta prueba nos ayudan a concluir que el canal lateral detectado en la recuperación vectorial aislada no desaparece al introducir un LLM, pero su visibilidad depende mucho del modelo usado. Con modelos grandes, el coste de generación y la variabilidad del modelo pueden hacer que la señal sea más difícil de categorizar, aunque es posible que con un mayor enfoque en la configuración del modelo y un mayor número de pruebas se pudiera conseguir el ataque también con este tipo de modelos al no desaparecer la diferencia temporal. Por otro lado, con modelos pequeños, la señal se conserva mejor porque el tiempo de generación es menor y más estable, llegando, en algunos casos, el ruido del modelo a ayudar en la estabilización del tiempo total de respuesta.

6.4.3. RESULTADOS CON MECANISMOS DE CACHÉ

En esta última fase se evaluó cómo cambia el comportamiento temporal del sistema cuando se introduce un mecanismo de caché en el pipeline RAG. En este caso se estudió tanto si las cachés hacían que el tiempo medio se redujese como si esta reducción afectaba por igual a todos los grupos de consultas o si, por el contrario, afectaba de forma distinta si la frecuencia era mayor. Esto es importante porque, si las consultas relacionadas con contenido presente en la base de datos tienden a generar más aciertos de caché que las consultas de baja frecuencia, los sistemas estarían incrementando la facilidad del ataque, haciendo la diferencia temporal más grande.

Para poder interpretar correctamente los resultados, necesitamos comparar los resultados de un sistema sin caché, como el anterior, con distintos sistemas con diferentes tipos de caché, ya que, además de entender si los sistemas de caché incrementan o reducen la diferencia de latencia, también se quería ver qué sistema lo hacía en mayor medida. Por esto se realizó la mayor prueba hasta el momento, ejecutando de forma seguida una prueba sin caché como la anteriormente explicada y una prueba con cada configuración de caché escogida. En la prueba sin caché se observaron diferencias muy similares a las anteriormente estudiadas, encontrando de nuevo diferencias alrededor de los 80 ms con k=50, aunque en este caso con valores más bajos, ya que la gráfica alquilada para esta prueba era algo mejor, haciendo los cálculos más rápidos, pero lo importante, que son las diferencias entre grupos, se mantienen. En la Fig. 19 se muestra la prueba realizada sin caché.

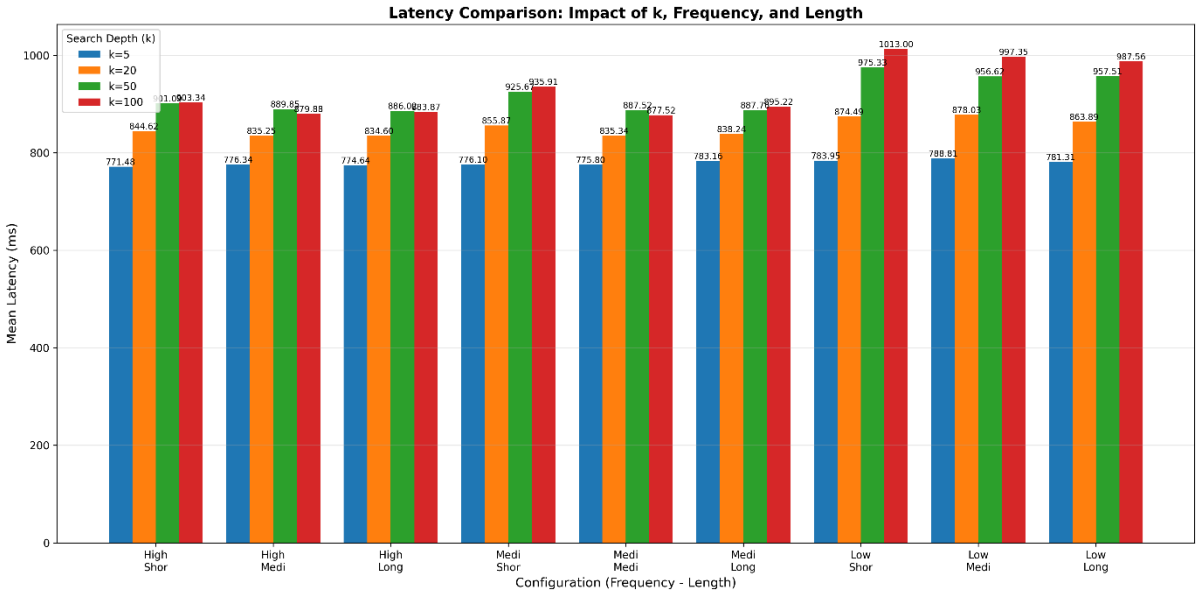


Fig. 19. Comparación de latencias del pipeline RAG sin caché durante el experimento con mecanismos de caché.

En esta prueba se volvieron a ajustar los valores de k, ya que, como se iban a usar cachés, se podían probar valores más altos como 100 al evitar la generación en muchas ocasiones.

La primera caché probada en esta prueba fue la caché exacta, la cual ya mostró una reducción notable de los tiempos medios en todas las consultas, ya que, al probar la mayoría de consultas parecidas y solo algunas iguales, muy pocas de ellas caían en caché tanto en las consultas de alta como de baja frecuencia. En estas pruebas también se midieron los aciertos en caché con un contador que observaba los tiempos y, si estos bajaban de un umbral, se consideraban

acierto, por lo que no es un contador perfecto, pero debería ser bastante ajustado, ya que las diferencias temporales entre una consulta cacheada y una no cacheada son de más de 400 ms. En este contador observamos un mayor acierto por parte de las consultas de alta frecuencia, obteniendo alrededor de un 9% de aciertos, y de media frecuencia, obteniendo un acierto de alrededor del 8% frente a las consultas de baja frecuencia, que obtuvieron alrededor de un 4% de acierto de caché. Esto hace que la diferencia temporal obtenida en la prueba sea similar o un poco mayor respecto a la comparación entre baja frecuencia y el resto de las frecuencias, al tener tan poco porcentaje de aciertos en caché, pero un menor porcentaje en ese grupo que en los otros. Por otro lado, la diferencia entre las consultas de alta y media frecuencia desaparece al completo al tener un acierto en caché tan similar como se puede ver a continuación en la Fig. 20.

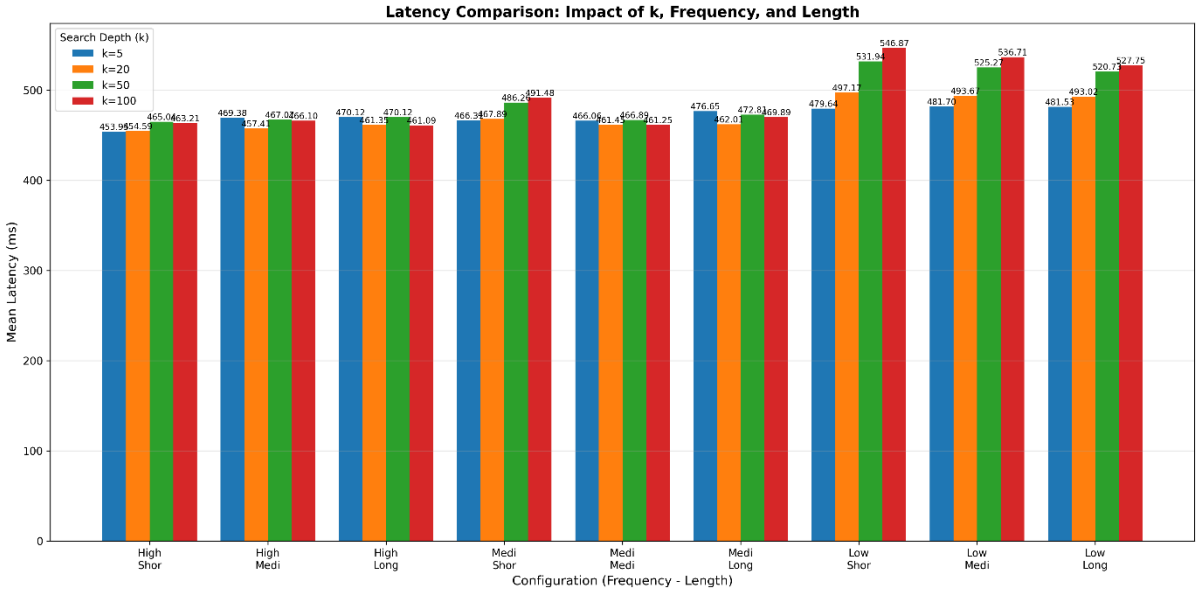


Fig. 20. Comparación de latencias del pipeline RAG con caché exacta durante el experimento con mecanismos de caché.

Si analizamos la gráfica, nos podemos dar cuenta de que la diferencia temporal entre el grupo de baja frecuencia y el resto sigue estando presente, pero realmente esto se debe más a que las consultas de baja frecuencia son más distintas entre sí, al tratar de temas variados y modernos que no deberían aparecer en el corpus, que por la diferencia temporal de recuperación vectorial. Es verdad que esta diferencia en la recuperación vectorial ayuda a que esto se vea más, pero realmente en esta y las siguientes pruebas donde se usan cachés, los tiempos son controlados casi por completo por las cachés al reducir en mayor medida la temporalidad. Por lo tanto, en este caso es verdad que, si la prueba está bien diseñada, puede seguir siendo útil al diferenciar bastante entre términos distintos o similares, pero tampoco debe interpretarse automáticamente que esto signifique que un término se encuentra presente en la base de datos, ya que la caché nos podría estar engañando si diseñamos erróneamente la prueba.

Por otro lado, como se ha comentado en el apartado 6.2.6, también se probaron configuraciones de caché semántica con distintos umbrales. En este caso, todas las configuraciones se comportaron de forma similar, ya que, aunque con diferencias menores, de nuevo mostraron las mayores diferencias entre los grupos de baja frecuencia y el resto, pero en este caso sí que mostraron diferencias algo menores entre los grupos de media y alta frecuencia al tener más aciertos los de alta frecuencia. En la Fig. 21 se pueden observar las diferencias de latencias de la caché con umbral 0.85.

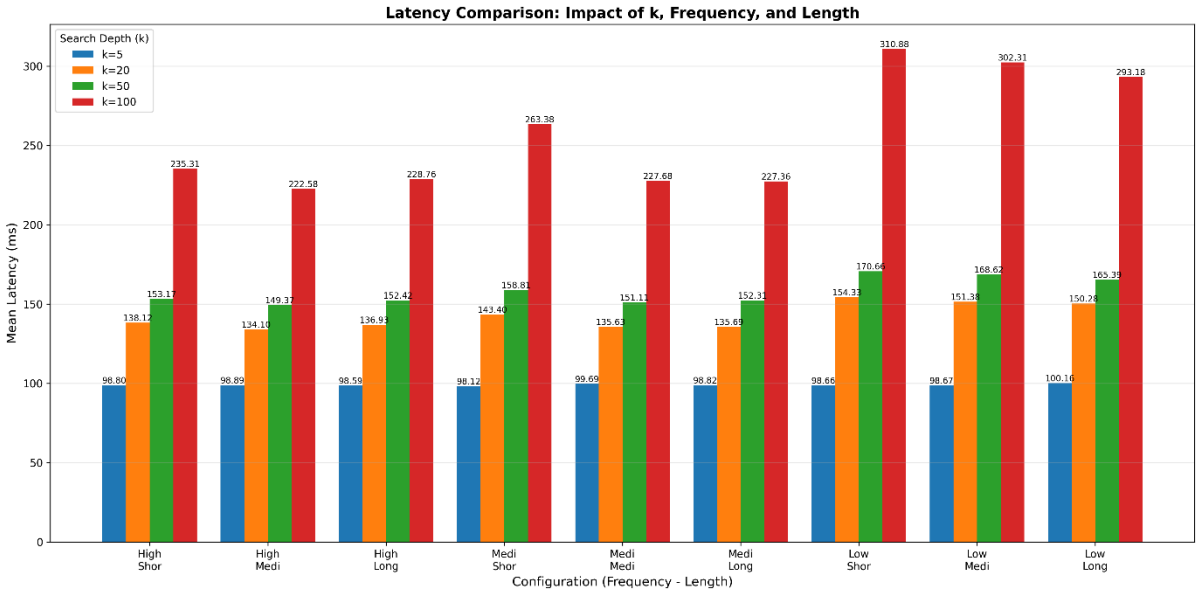


Fig. 21. Comparación de latencias del pipeline RAG con caché semántica y umbral 0.85 durante el experimento con mecanismos de caché.

En este caso llegamos a la misma conclusión que con la caché exacta, que es que con un buen diseño de prueba el ataque puede seguir siendo funcional, aunque es verdad que la caché podría engañarnos si repetimos demasiado los términos usados para las consultas de baja frecuencia.

Por último, es importante mencionar que, en el caso de todas las cachés semánticas, los aciertos a caché fueron mayores que con la caché exacta. En este caso, variaron respecto al umbral aceptado, yendo desde el 35% en la caché con el umbral más bajo hasta el 15% en la que tiene el umbral más estricto. Estos porcentajes hacen que, si un grupo de consultas consigue más aciertos, su tiempo medio baje de forma notable. Estos resultados muestran que la caché no debe tratarse únicamente como una mejora de rendimiento, ya que en un sistema RAG puede introducir un comportamiento temporal diferencial, especialmente cuando las consultas relacionadas con información presente se repiten más, son más parecidas entre sí o activan con más facilidad entradas ya almacenadas.

6.4.4. COMPARACIÓN GLOBAL ENTRE ESCENARIOS

Una vez analizados los resultados de cada escenario, se puede hacer una lectura conjunta de la investigación. El comportamiento observado no es igual en todas las pruebas, pero sí mantiene una misma idea principal: las consultas asociadas a menor presencia en el corpus tienden a generar tiempos superiores a las consultas asociadas a información más presente. Esta diferencia aparece primero en la recuperación vectorial aislada y después se mantiene, con más o menos claridad, al añadir otros componentes del sistema RAG, lo que valida la hipótesis planteada y da pie a crear la herramienta de auditorías ya explicada, como forma de profesionalizar este tipo de pruebas en entornos profesionales por parte de auditores experimentados.

La recuperación vectorial aislada es el escenario que permite entender mejor el origen del canal lateral. Al medir únicamente la llamada a ChromaDB, se observa que el motor de búsqueda vectorial no responde siempre igual y que el valor de k afecta directamente a la visibilidad de la diferencia temporal. Con valores bajos de k, la separación entre grupos es pequeña, pero al

aumentar el número de vecinos recuperados, el sistema necesita hacer más trabajo y la diferencia entre consultas de baja frecuencia y el resto se vuelve más visible. Por tanto, este escenario valida la base de la hipótesis, aunque al solo medir la recuperación vectorial muestra que cualquier variación del sistema genera una interferencia en la prueba.

El escenario con pipeline RAG completo aporta una visión más cercana al uso real del sistema. En este caso, la señal temporal no desaparece, pero queda condicionada por el modelo de lenguaje. Con Qwen2.5-7B-Instruct, el tiempo de generación domina gran parte de la medición, haciendo que la diferencia de recuperación sea más difícil de separar del resto del tiempo total. En cambio, con SmolLM2-135M-Instruct, al reducirse el coste del modelo, la diferencia vuelve a aparecer de forma más estable, lo que muestra que el ataque no depende únicamente de un entorno aislado. Además, se observa que su efectividad está muy relacionada con el peso que tenga la generación del LLM dentro del tiempo total.

Las pruebas con caché muestran un comportamiento distinto, ya que en estos casos, la diferencia temporal ya no depende solo del recorrido en la base vectorial, sino también de si unas consultas activan más aciertos en caché que otras. Esto puede hacer que la señal se mantenga o incluso aumente, pero también puede modificar su interpretación porque esta diferencia no siempre significa que el motor vectorial haya trabajado menos tiempo, sino que la caché ha intervenido más en el resultado. Por esto, la caché no elimina necesariamente el ataque, pero sí obliga a diseñar las pruebas con más cuidado.

Comparando todos los escenarios, se observa que el ataque es más claro cuando el trabajo de recuperación vectorial tiene mayor importancia en la medición final y el ruido no es tan grande como para ocultar las diferencias. La recuperación aislada demuestra el fenómeno, el pipeline RAG confirma que la señal puede propagarse al tiempo completo de respuesta y las pruebas con caché muestran que los mecanismos de rendimiento pueden introducir nuevas diferencias temporales. Por tanto, el resultado más importante es que la hipótesis no se apoya en una única prueba, sino que la señal temporal aparece en fases distintas y con configuraciones diferentes. Aun así, los resultados también muestran que no se puede interpretar una medición individual como prueba directa de presencia, ya que el ataque necesita muchas consultas, comparación entre grupos y un entorno suficientemente estable para realizar cualquier clasificación de términos de forma correcta al haber algo de solapamiento entre los grupos siempre. Esta idea es clave para la herramienta, ya que su objetivo es automatizar un proceso de medición repetida que permita estudiar si el sistema auditado presenta una diferencia temporal explotable y si en este sistema se puede inferir el uso de algunos términos más que otros.

6.5. VALIDACIÓN DE LA HERRAMIENTA DESARROLLADA

Una vez terminada la parte experimental, se comenzó con el desarrollo de la herramienta. A lo largo de este y con su finalización, se fueron haciendo fases de validación para ir comprobando que los módulos implementados en cada fase de desarrollo funcionasen como se esperaba. Estas validaciones se plantearon como la combinación de pruebas manuales y pruebas automatizadas. Las pruebas manuales se realizaron más centradas en la última parte de desarrollo, donde se necesitaba probar la experiencia de usuario con la interfaz, al necesitar que todo funcionase no solo bien, sino que de la forma esperada por el usuario. Además, en estas pruebas también se probaron algunas acciones que eran imprescindibles, como la comunicación HTTP con sistemas externos, el comportamiento del worker, el avance de las auditorías e inferencias, la visualización de resultados y la exportación en formato CSV. Por otro lado, las pruebas automatizadas se centraron en la realización de diferentes tipos de tests

para probar todas las partes del sistema. Entre estos tests se encuentran tests unitarios tanto para partes internas del backend como para los componentes del frontend, tests de integración con componentes como la base de datos o el sistema de colas, pruebas de las API que comprueban que todo funcione como se espera y algunas pruebas de flujo que imitan algunos comportamientos de un usuario. Esta combinación tiene sentido para el tipo de aplicación desarrollada, ya que un test unitario puede comprobar que una función haga lo esperado en distintos escenarios o que algo se cree o borre correctamente, pero no puede asegurar que una auditoría larga se ejecute bien contra un RAG real desplegado en otra máquina o que la interfaz del usuario se vea limpia y clara.

6.5.1. ENTORNO RAG DE VALIDACIÓN EN RUNPOD

Para las pruebas manuales de ejecución de auditorías o inferencias de términos, era necesario atacar un sistema externo real. En este caso, como no se disponía de uno, se desarrolló un sistema RAG que imitase a un sistema externo real y se desplegó en RunPod. La idea principal de este sistema es que se pudieran hacer pruebas completamente reales al tener la herramienta desplegada en local y mandarle las consultas a un endpoint HTTP que estaría esperando peticiones en RunPod. Por esto, con estas pruebas íbamos a dar el último paso necesario para hacer pruebas completamente reales al introducir por medio el ruido de red.

Todas las pruebas de la investigación, como se ha explicado anteriormente, se realizaron con scripts que se conectaban directamente a la base de datos y medían el tiempo desde que el sistema iniciaba el proceso de recuperación hasta que el sistema finalizaba la generación de la respuesta. En este caso, con el ruido de red por el medio, además de ese proceso, se mediría también el tiempo que tarda la petición en enviarse y el tiempo que la primera parte de la respuesta tarda en llegar, por lo que de nuevo introduciríamos más ruido al ataque, haciéndolo más difícil y viendo por primera vez si la funcionalidad de la herramienta sería buena con sistemas reales. Por otro lado, es bueno explicar que el ruido introducido en estas pruebas es mayor del que se introduciría en un sistema real, ya que esta herramienta está preparada para pruebas en redes locales que se conecten directamente al sistema RAG y no como en este caso, que el sistema al que se llama externamente está en una máquina alquilada en Rumania. De esta manera, si en estas pruebas se consigue hacer que el sistema funcione correctamente, se podrá asumir que en el entorno esperado el rendimiento de la herramienta será algo mejor.

Este sistema externo que imita al endpoint de un chatbot empresarial se construyó a partir del código usado durante la investigación, ya que en los scripts experimentales anteriores ya se desarrolló un pipeline RAG completo conectado a la base de datos vectorial del corpus PG-19. Por lo que, para esta fase, ese código se reorganizó como una aplicación FastAPI independiente, haciendo que ya no se ejecutase como un script que lanza consultas y mide tiempos, sino como un servicio que se queda escuchando peticiones, eliminando por completo el sistema de medición temporal al ser responsabilidad de la herramienta. Con esto, el servicio carga la colección gutenberg_completo de la base de datos de ChromaDB cargada en RunPod, que contiene el corpus preparado durante la investigación y mantiene el modelo all-MiniLM-L6-v2 para realizar la vectorización de las consultas. En este entorno de validación no se incorporó una capa de caché adicional para que la señal temporal observada dependiera principalmente del pipeline RAG y del ruido de red introducido por la comunicación con RunPod. Sobre esa base se construye un retriever de LangChain igual que en la investigación, con un valor fijo de k=20, que es un valor realista de este parámetro. Además, para la generación de la respuesta se utiliza de la misma manera que en la investigación el modelo

SmolLM2-135M-Instruct cargado desde Hugging Face y con el mismo prompt del sistema que hace que su comportamiento imitara al de cualquier sistema RAG empresarial.

Con el pipeline ya profesionalizado, se creó el endpoint principal expuesto por el servicio. En este caso se usó el /chat como el final de la URL y el método POST con un cuerpo JSON que contiene la pregunta. Además, para hacerlo más realista y evitar que el endpoint quedase completamente abierto, se añadió una comprobación simple mediante una API key enviada en la cabecera X-API-Key. Esta clave se introdujo en el sistema como una variable de entorno, ya que RunPod da la posibilidad de crear variables de entorno para que todo el sistema las pueda tener a su disposición y que no se quede escrita en el código, manteniendo las buenas prácticas del proyecto. Con todo esto, el RAG primero recibe la petición con la API key y la consulta, posteriormente, comprueba la clave y, si esta es correcta, pasa la consulta por todo el pipeline RAG generando una respuesta y, por último, devuelve una respuesta JSON con el texto generado.

Más adelante, para facilitar la depuración de algunos problemas encontrados con la compatibilidad de algunas dependencias en las gráficas alquiladas, se implementó un endpoint /health, que permite comprobar información útil del sistema, como si el servicio está levantado, qué colección está cargada y si PyTorch detecta CUDA o no, lo que fue útil durante el despliegue, ya que permitió detectar problemas como ejecuciones que usaban la CPU en vez de la GPU porque esta no estaba siendo reconocida por las dependencias instaladas.

Durante el arranque, el servicio carga el modelo, conecta con ChromaDB y construye la cadena RAG de LangChain. Después ejecuta una consulta de calentamiento para iniciar el modelo y preparar el pipeline antes de recibir peticiones reales, aunque esto ya lo hacen las pruebas, aquí también se realiza para asegurarnos de que las primeras mediciones no son erróneas. El despliegue en RunPod se realizó colocando la aplicación dentro de la máquina donde ya estaba disponible la base de datos vectorial, apuntando la ruta usada por el servicio directamente a la carpeta de la colección de ChromaDB. Para ejecutar el servicio, se instaló el entorno de Python con las dependencias necesarias y se lanzó FastAPI mediante Uvicorn, dejando el proceso escuchando en el puerto configurado de la máquina.

Con este montaje, la herramienta podía tratar el RAG de RunPod como un target externo real. Para hacerlo accesible desde la herramienta local, fue necesario abrir el puerto del servicio desde la interfaz de RunPod. Una vez expuesto el puerto, RunPod proporcionaba una URL pública asociada a esa máquina, la cual se configuró como la URL del target seguida del endpoint correspondiente, en este caso /chat. A partir de ahí, las auditorías y las inferencias de términos se ejecutaban desde la herramienta sin conocer internamente cómo estaba construido el RAG, lo que reproduce el modelo de black-box esperado por el sistema.

6.5.2. VALIDACIÓN DE LOS FLUJOS PRINCIPALES DE LA HERRAMIENTA

Una vez desplegado el RAG externo en RunPod, ya se podían realizar las pruebas manuales necesarias para poder probar que la ejecución de las distintas pruebas funcionase correctamente. En este caso, primero nos centramos en las auditorías, ya que es el flujo principal de la aplicación, y posteriormente probamos el funcionamiento de la inferencia de términos.

En cuanto a las auditorías, para que las pruebas fueran completas y nos aseguraran que la ejecución realmente funcionaba, se validaron por separado las piezas que forman todo el flujo

de una auditoría: la creación de targets, la carga de datasets, el lanzamiento de sesiones, la ejecución en segundo plano y la consulta de resultados.

La gestión de targets y dataset no son partes de la ejecución de una auditoría, pero ya que son necesarias para ejecutar una, se probaron las primeras antes de implementar el RAG externo, ya que estas partes no necesitan realizar llamadas a sistemas externos. La única función que se esperó probar con el RAG implementado fue la de enviar un test al target, ya que, aunque sí se probara con el RAG local, esto no nos aseguraba que con una URL externa al sistema de Docker esto fuera a funcionar correctamente.

En las pruebas de los targets se probó principalmente la introducción de los datos de forma correcta, como la URL del endpoint, los headers y el cuerpo de la petición. Además, se verificó que el target pudiera crearse desde la interfaz, guardarse en la base de datos y borrarse cuando el usuario lo solicitase, probando las principales implementaciones de la API del backend. Además, se hicieron todas las pruebas necesarias para generar todos los tipos de fallos distintos del sistema. Entre estos encontramos fallos como la falta de algún campo obligatorio o la introducción de caracteres inválidos en algunos campos. Por otro lado, en la carga de datasets, se probó a subir archivos correctos e incorrectos para comprobar que la herramienta aceptaba únicamente los formatos esperados. En este caso, la herramienta debía guardar el archivo, registrar el dataset en la base de datos y añadir su vista previa a la interfaz de usuario para que el usuario la pudiese consultar en cualquier momento que quisiese, cuando el archivo tenía el formato y contenido adecuado. Cuando esto no ocurría, la aplicación debía mostrar un error claro y no permitir que ese archivo se usara en una auditoría. Estas pruebas nos permitieron detectar errores de estos dos módulos de la aplicación para que, posteriormente, en la ejecución de auditorías, supiéramos que los errores no venían generados por estos módulos.

Con los targets y datasets comprobados, ya se pasó a validar la auditoría. En esta validación teníamos tres partes distintas: el lanzamiento, la ejecución desde el worker y la consulta de resultados. En la primera parte, las pruebas manuales realizadas fueron similares a las anteriormente ejecutadas con los targets, ya que en las pruebas correctas lo principal era poder crear una auditoría y que esta se metiese en la cola y en la base de datos, y en las pruebas erróneas se probaron todos los errores que podía sacar la herramienta, como que el target o el dataset ya no existiesen o que algún campo obligatorio estuviera vacío.

Una vez realizado el flujo correcto de lanzamiento de auditorías, la prueba de ejecución comenzaba sola, ya que es en este momento en el que el worker iniciaba la ejecución de este trabajo enviando consultas al target seleccionado. En esta parte se probaron muchas cosas, al ser una parte algo compleja con la adquisición de trabajos por parte del worker. Las pruebas consiguieron demostrar que todo el flujo era funcional, comprobando que la auditoría estaba siendo ejecutada por el worker, que el resto de la aplicación no se quedaba congelada, que al crear otro trabajo mientras uno se estaba ejecutando, el worker lo empezaba a ejecutar cuando quedaba libre, intercalando sus ejecuciones correctamente, que el RAG externo desarrollado recibía las peticiones enviadas y que el worker recibía las respuestas y guardaba las mediciones en la base de datos. Además, se pudieron comprobar todos los flujos del frontend, como que las barras de progreso de la ejecución se ajustasen a la realidad y actualizasen cuando tocaba o que los botones de abortar prueba funcionasen correctamente. Por último, en esta fase se buscó generar errores de la misma forma que en el resto de pruebas, cortando la conexión con el RAG o forzando que el sistema funcionase más lento por la alta demanda de peticiones al backend de gestión.

Mientras se ejecutaba la auditoría de prueba, se fue validando la pantalla de resultados, comprobando que las mediciones se mostrasen en las tablas paginadas, que los filtros funcionasen correctamente, que las gráficas se generasen a partir de los datos guardados y que los datos se fuesen actualizando a lo largo de la prueba. Una vez acabó la prueba, también se verificó que el resumen estadístico agrupase los resultados por frecuencia y longitud, mostrando valores como media, mediana, desviación y p-value en las comparaciones. Además, se probó que la exportación en CSV permitiera descargar tanto el resumen como los datos completos de la auditoría.

Es importante comentar que se observó una diferencia importante entre las ejecuciones experimentales y las auditorías realizadas desde la herramienta. En la fase de investigación, algunas pruebas llegaron a durar más de 12 horas, ya que determinadas configuraciones se repetían muchas veces para estudiar la estabilidad de los resultados y reducir el efecto de comportamientos puntuales. En cambio, la herramienta está pensada para un uso más práctico: una auditoría ejecuta una vez el JSON de consultas cargado por el usuario y guarda las mediciones obtenidas. Por esto, aunque la duración depende del tamaño del dataset, del target y del tiempo de respuesta del modelo, una prueba larga podía completarse aproximadamente en 2 horas. Esta reducción de tiempo hace que la herramienta sea más manejable para un auditor, manteniendo la lógica de medición necesaria para estudiar si existe una señal temporal explotable.

Verificados todos los procesos que se centraban en la auditoría y comprobado que el flujo funcionaba a la perfección, el desarrollo se centró en realizar el módulo de inferencias. Una vez desarrollado este módulo, la validación del mismo fue algo distinta al no tener que comprobar algunos módulos reutilizados como el de medición de consultas o el de exportación de CSV, pero sí otros nuevos como toda la interfaz desarrollada o el sistema de clasificación de términos centrado en la media temporal obtenida. Estas pruebas de nuevo se dividieron en las etapas, no comprobadas con anterioridad, que conforman el flujo de inferencia, que son: importación de CSV, formación del perfil de calibración, ejecución de la prueba por rondas, clasificación de términos en clases temporales.

La primera parte que se probó fue el formulario en el cual el usuario debía meter todos los datos. Esta prueba se realizó de la misma manera que en las auditorías para comprobar que todos los campos obligatorios debían ser rellenados correctamente, pero en este caso se incluía un campo donde el usuario podía subir un archivo CSV. Con este campo se hicieron más pruebas en profundidad, ya que era la primera vez que la aplicación aceptaba un archivo con este tipo de formato y la comprobación debía estar bien hecha al ser necesaria para la construcción del perfil de calibración de la prueba. Además, en estas pruebas también se observó que, una vez validado el CSV, el perfil de calibración se construyera correctamente y que los datos calculados se guardaran en la base de datos como se esperaba que lo hicieran.

Una vez verificado que se construía el perfil de calibración correctamente, se probaron los dos tipos de entradas que podía tener la prueba, pudiendo ser términos o consultas, y observando si, cuando se escogían términos, el programa formaba las consultas de la forma esperada.

Con la formación de la prueba probada por completo, se comprobó el funcionamiento por rondas de la ejecución al tener que lanzar un conjunto de consultas por cada ronda y tener que volver a evaluar únicamente los términos que siguen sin una clasificación clara en ese momento. Durante este proceso se prestó atención a varias cosas importantes a probar. La primera de ellas fue que el sistema no quedase congelado y que se pudiese encolar una

inferencia de términos a la vez que se estaba ejecutando una auditoría y que posteriormente el worker la recogía correctamente al quedar disponible. Esto fue clave, ya que este momento era el primero en el que el worker estaba trabajando con dos procesos que ejecutaban funciones distintas y era necesario comprobar que el comportamiento no fuese errático. Por otro lado, también se comprobó que la caché interna del sistema, que presentan las bases de datos como ChromaDB, no influyera demasiado al enviar consultas similares de manera seguida. Para reducir este fenómeno, se había desarrollado un sistema de desorden no aleatorio, el cual era importante probar su funcionamiento, en el que, si al desordenar las consultas se encontraban varias consultas similares seguidas, estas se movían de sitio introduciendo consultas de términos inventados para facilitar la separación.

Por último, como en las auditorías, se verificó toda la interfaz de resultados, comprobando que cada término mostrase su clasificación, la media temporal observada, el número de mediciones válidas y la referencia más cercana. También se revisaron las gráficas, tanto al final como en medio de la prueba, especialmente la que muestra el avance de la clasificación a lo largo de la prueba. Además, se comprobó la posibilidad de revisar qué consulta concreta se envió en cada momento, qué término representaba, qué latencia obtuvo y si hubo error o no, viendo las consultas individuales en una tabla paginada.

Como se ha comentado junto a todas las pruebas manuales, se prepararon distintas pruebas automatizadas o tests para validar partes y lógicas internas del backend, las cuales cubren distintos tipos de comportamiento: validación de targets, validación y normalización de datasets, cálculo estadístico, lógica de medición, generación de resultados y gestión de estados.

La validación completa de la herramienta permitió comprobar que el flujo principal de la herramienta funcionase de extremo a extremo, haciendo que el usuario pueda registrar un RAG externo, subir un dataset, lanzar una auditoría, observar el progreso, detenerla si es necesario, analizar los resultados y exportarlos. Además, mostró que la inferencia de términos es más sensible que la auditoría general al intentar tomar decisiones sobre la clasificación de términos concretos, pero que igualmente todos sus procesos, como son generar un perfil de calibración, desordenar consultas de forma no aleatoria, ejecutar una prueba por rondas y poder verificar los resultados en gráficas, entre otros, funcionaban correctamente.

6.5.3. EVALUACIÓN DE RESULTADOS DE LA HERRAMIENTA

Una vez comprobado que los flujos principales de la herramienta funcionan correctamente, también es necesario analizar si los resultados que genera son útiles para el objetivo del proyecto, que es transformar esas mediciones en información que pueda interpretar un auditor. Por esto, en este apartado se observarán distintas ejecuciones, tanto de auditorías como de inferencias, y se evaluarán los resultados, obteniendo conclusiones sobre la utilidad de la herramienta desarrollada.

La primera ejecución realizada desde la herramienta ya mostró resultados útiles para analizar si el uso de la herramienta permitía observar el canal lateral temporal. En la Fig. 22 se visualizan los resultados obtenidos en esta prueba. La gráfica de barras que se observa muestra el Time-to-First-Byte medio agrupado por frecuencia y longitud de consulta, igual que en los resultados de la experimentación. En esta ejecución, el comportamiento esperado aparece de forma clara, ya que los grupos de baja frecuencia presentan valores mayores en las tres longitudes de consulta.

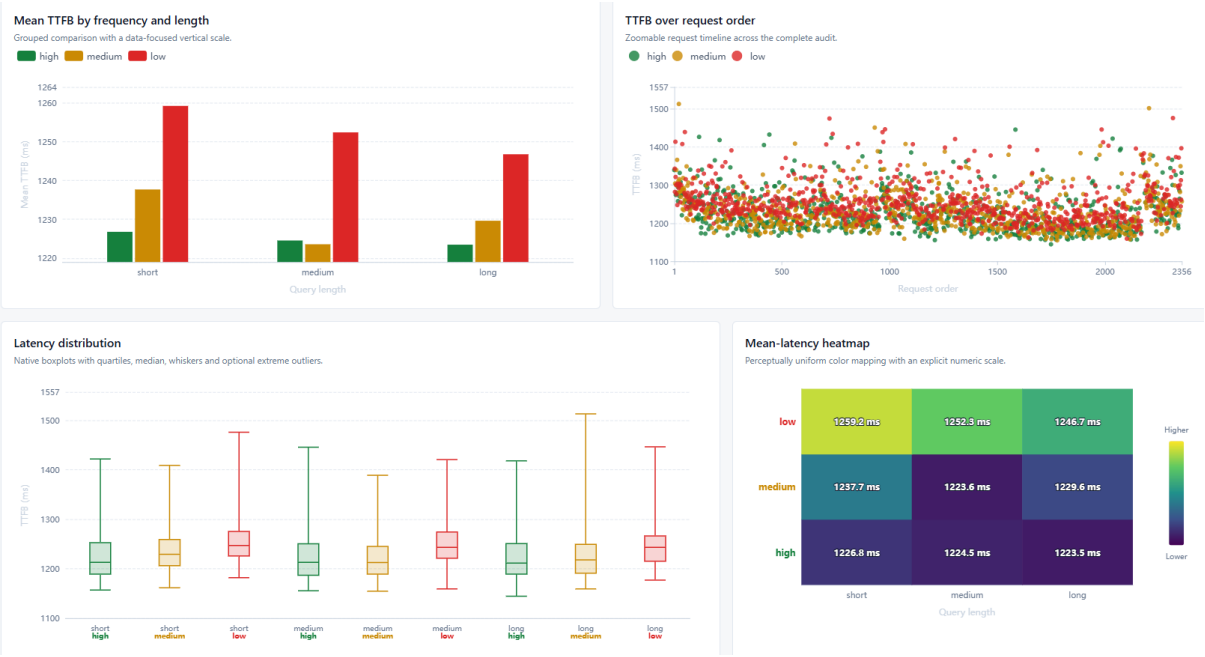


Fig. 22. Resultados gráficos de la primera auditoría ejecutada desde la herramienta.

La misma pantalla también aporta otras gráficas que ayudan a entender mejor la ejecución, como la de distribución del TTFB a lo largo del tiempo, la cual permite comprobar que las mediciones no aparecen agrupadas artificialmente por frecuencia, sino mezcladas durante la auditoría. Por otro lado, el diagrama de cajas muestra la dispersión de cada grupo y permite ver que existe ruido, ya que las distribuciones se solapan en algunas zonas. Por último, el mapa de calor facilita una lectura rápida del patrón general, mostrando que los valores más altos se concentran en la fila de baja frecuencia. Esta combinación de gráficas es similar a la utilizada en la experimentación y es muy útil al permitir la comparación de la tendencia, la dispersión y la estabilidad temporal.

Además de las gráficas en la pantalla de resultados de las ejecuciones, el auditor recibe una interpretación estadística. En este caso, el análisis estadístico de esta primera ejecución se muestra en la Fig. 23. En ella se observa cómo en esta prueba las comparaciones más relevantes para el ataque, que son las que enfrentan los grupos de alta o media frecuencia contra el grupo de baja frecuencia, obtienen una evidencia fuerte. Esto indica que el p-value encontrado en la comparación de estos grupos es menor al umbral de 0.001 establecido en la herramienta, lo que significa que la herramienta es capaz de detectar una separación temporal entre consultas relacionadas con contenido presente en el corpus y consultas más alejadas de él.

Statistical comparisons by frequency

The p-value comes from Welch's test. Lower values indicate stronger evidence of a temporal difference.

Groups	Mean difference	Median difference	P-value	Evidence
high vs medium	5.54 ms	8.85 ms	2.90e-2	weak
high vs low	28.68 ms	31.68 ms	3.69e-29	strong
medium vs low	23.14 ms	22.84 ms	1.76e-19	strong

Fig. 23. Comparaciones estadísticas por frecuencia en la primera auditoría.

Por otro lado, esta misma tabla también muestra una limitación importante respecto a la comparación entre alta y media frecuencia, ya que tiene una evidencia, aunque también grande, mucho más débil que el resto. Este resultado se explica también estudiando las gráficas de la Fig. 22, donde podemos observar que el grupo de longitud media no ha tenido diferencia temporal entre el grupo de alta y media frecuencia. Por este comportamiento y para probar más

en profundidad la herramienta, se realizaron más pruebas con las que se consiguió llegar a una conclusión sólida sobre el funcionamiento de la misma.

Antes de continuar con el estudio de la segunda ronda de pruebas, es interesante observar cómo la opción de modificar el eje vertical permite al auditor visualizar con mayor claridad los resultados de las pruebas. En la Fig. 24, se muestra la misma gráfica de barras que en la Fig. 22, pero con la opción de eje vertical iniciado en cero activada.

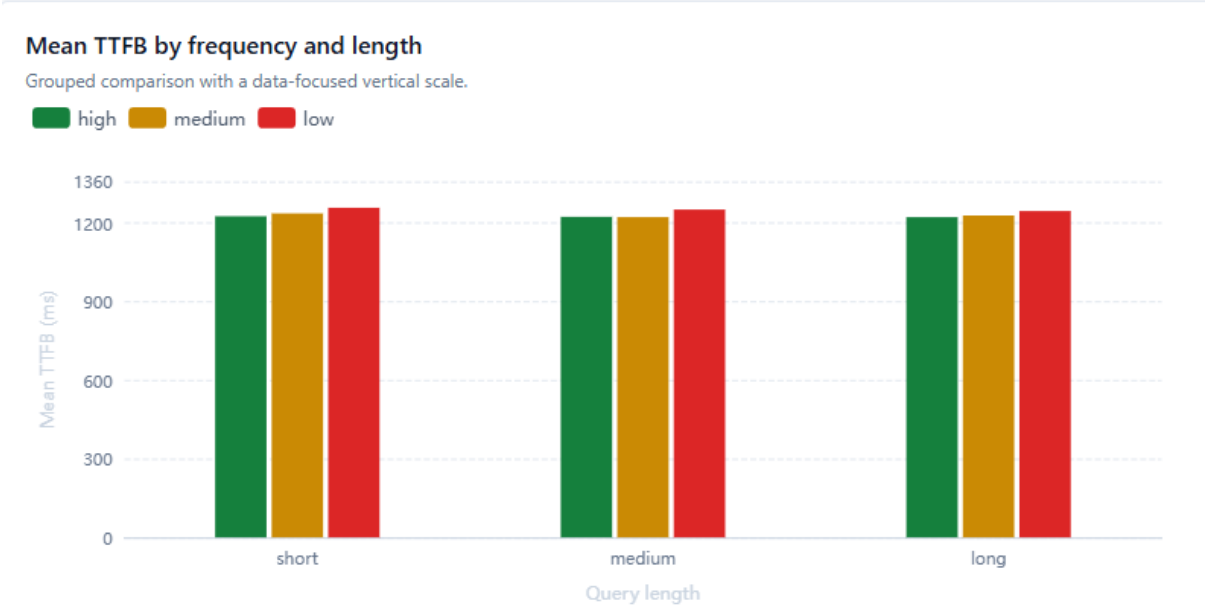


Fig. 24. Resultados de la primera auditoría con eje vertical iniciado en cero.

Esta vista muestra que, aunque la diferencia entre grupos es medible y estadísticamente significativa, su magnitud absoluta es pequeña en comparación con la latencia total del sistema. Al ver la gráfica de esta manera, las barras quedan visualmente muy próximas, lo que ayuda a entender que el canal lateral se basa en variaciones pequeñas que se repiten de forma consistente. Además, en la herramienta, las diferencias de latencias son menores que en la experimentación respecto al tiempo total, ya que se introduce el ruido de red por medio.

En este caso, la comparación entre la gráfica con la escala ajustada y la gráfica con el eje iniciado en 0 nos permite observar que la decisión de añadir la posibilidad de ajustar los ejes es clave para que el auditor pueda observar con claridad las diferencias, pero sin olvidarse de la representación real de los datos.

Para continuar con las pruebas de las auditorías, se realizó una segunda ejecución, la cual se interrumpió para observar el comportamiento de la herramienta frente a este tipo de situaciones. Como se muestra en la Fig. 25, la petición que se realizó en el momento de la interrupción quedó registrada como error, sin valores temporales válidos asociados. Para encontrarla, se utilizó la tabla paginada con filtros de la parte final de la pantalla de resultados, donde se pudo ver cómo se conservaba la petición junto con su frecuencia, longitud, consulta y estado.

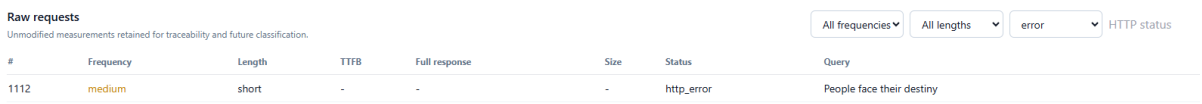


Fig. 25. Petición fallida registrada durante la segunda auditoría.

Además de esta comprobación, esta prueba nos permitió obtener resultados útiles, ya que el sistema continuó sin problemas la auditoría, los cuales utilizamos para obtener más conclusiones sobre el sistema. Los resultados se muestran en la Fig. 26, donde se puede comprobar cómo en esta ejecución la separación temporal aparece todavía más ordenada que en la primera. Las consultas de baja frecuencia vuelven a presentar los tiempos medios más altos en las distintas longitudes, mientras que las consultas de alta frecuencia quedan por debajo. Este comportamiento nos permite ver cómo la prueba anterior no había sido un caso aislado, sino que, aunque las diferencias sean pequeñas, son estables y se pueden medir correctamente en cada prueba ejecutada.

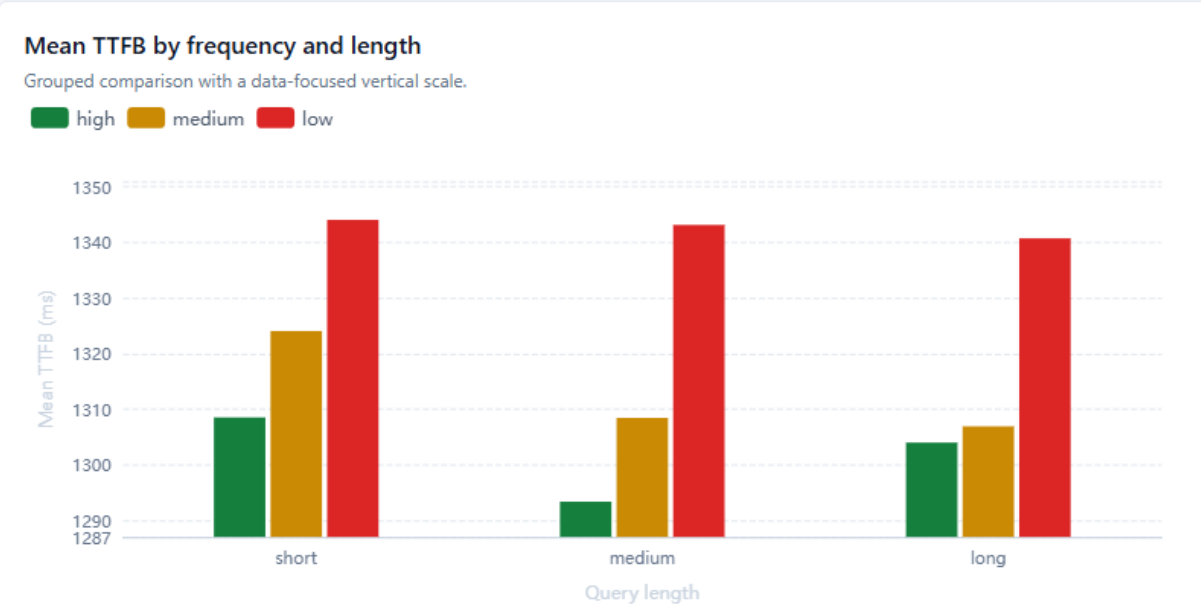


Fig. 26. Comparación de latencias medias en la segunda auditoría.

Es importante mencionar que estas pruebas se realizaron utilizando distintas semillas, por lo que, aunque el sistema atacado sea el mismo y las consultas lanzadas también, el orden de lanzamiento de estas es aleatorio en cada ejecución.

En esta ejecución, el mapa de calor mostró de nuevo cómo los valores de las consultas con frecuencias bajas siempre son mayores que los del resto de consultas. Este mapa de calor se muestra en la Fig. 27 y, además de este comportamiento, muestra un fenómeno que ya observamos en la investigación, que es la diferencia de funcionamiento del grupo de consultas cortas con el resto. Aunque en esta prueba todos los grupos muestran una progresión correcta de tiempos, ya empezamos a observar cómo el grupo de longitud corta es el más estable, teniendo siempre una progresión similar entre grupos de frecuencia.

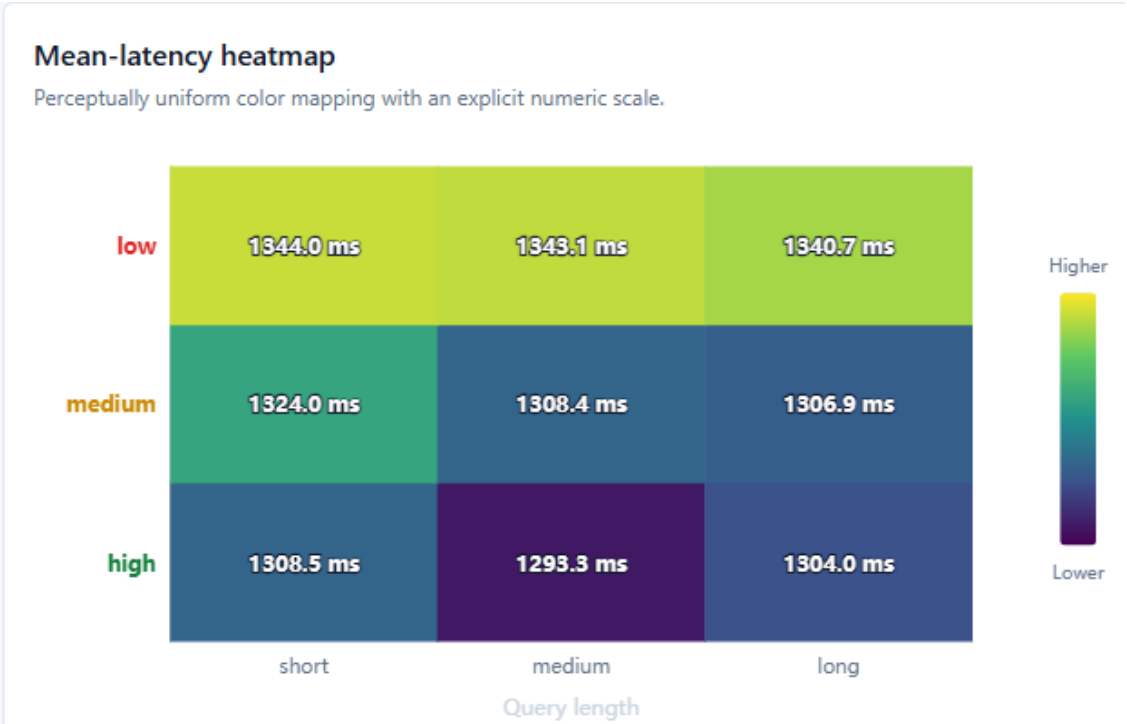


Fig. 27. Mapa de calor de latencias medias en la segunda auditoría.

En esta segunda ejecución, como se muestra a continuación en la tabla estadística de la Fig. 28, las comparaciones entre grupos aparecen con una evidencia fuerte, incluyendo también la separación entre alta y media frecuencia, ya que en este caso, como se ha mencionado, todos los grupos mostraron una progresión de latencias correctas.

Statistical comparisons by frequency

The p-value comes from Welch's test. Lower values indicate stronger evidence of a temporal difference.

Groups	Mean difference	Median difference	P-value	Evidence
high vs medium	12.46 ms	10.23 ms	2.93e-5	strong
high vs low	41.13 ms	41.01 ms	3.93e-47	strong
medium vs low	28.67 ms	30.78 ms	1.71e-20	strong

Fig. 28. Comparaciones estadísticas por frecuencia en la segunda auditoría.

Al comparar la primera y la segunda auditoría, se puede extraer la conclusión de que, aunque el sistema tenga interferencias o sobrecargas, este sigue mostrando las diferencias entre grupos. Como se puede observar, los tiempos de la segunda ejecución son en promedio algo mayores que en la primera, lo que puede deberse a muchos factores internos del sistema, pero las consultas de baja frecuencia siguen presentando un mayor Time-to-First-Byte que las consultas de alta y media frecuencia. Esta estabilidad del patrón, incluso con diferencias entre ejecuciones, es lo que hace que la herramienta sea útil para un auditor.

En este caso también se realizó una tercera ejecución para poner a prueba la herramienta en un contexto con mayores interferencias y en una máquina distinta para poder comprobar que el sistema mostraba diferencias de la misma forma ejecutándolo en distintos hardwares, ya sean más rápidos o más lentos. Como se observa a continuación en la Fig. 29, la máquina escogida era algo menos potente, por lo que los tiempos medios son mayores. A pesar de ello, la diferencia principal entre grupos sigue apareciendo, demostrando que la herramienta sigue siendo capaz de detectar la señal temporal incluso cuando el sistema tiene más carga o es más lento.

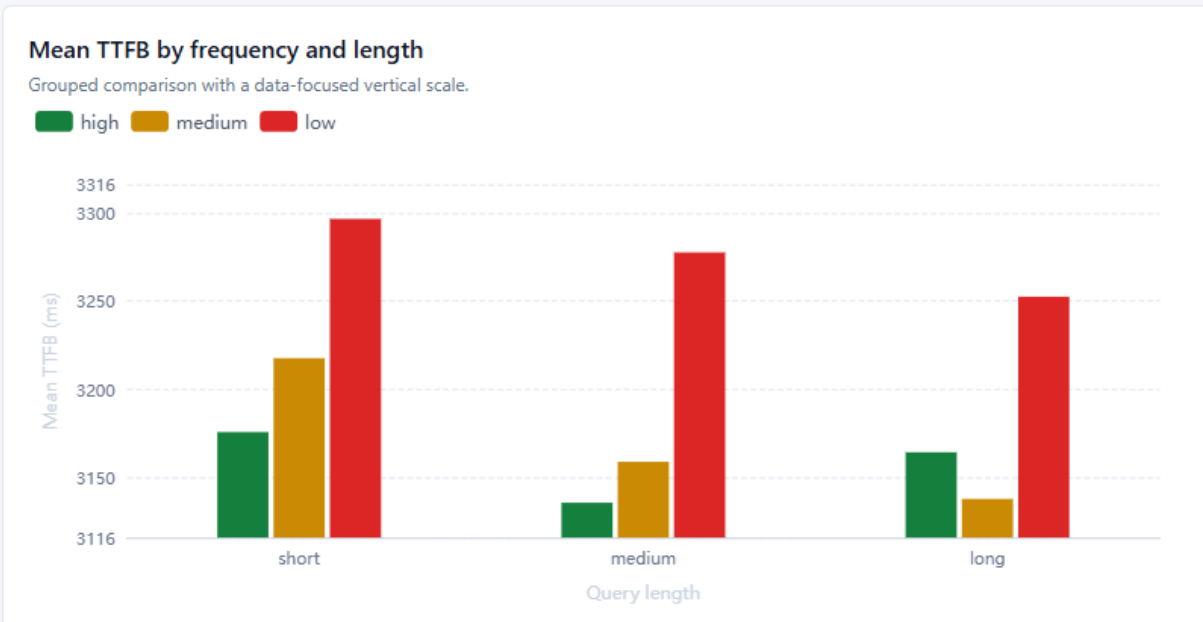


Fig. 29. Comparación de latencias medias en la tercera auditoría.

La Fig. 30 presentada a continuación muestra que, en esta tercera ejecución, las comparaciones contra el grupo de baja frecuencia vuelven a obtener una evidencia fuerte, pero la comparación entre alta y media frecuencia vuelve a aparecer como débil. En este caso, el grupo de consultas que genera esto es el de longitud larga, ya que no obtuvo una progresión temporal correcta.

Statistical comparisons by frequency

The p-value comes from Welch's test. Lower values indicate stronger evidence of a temporal difference.

Groups	Mean difference	Median difference	P-value	Evidence
high vs medium	19.09 ms	19.71 ms	1.91e-2	weak
high vs low	121.05 ms	142.72 ms	1.06e-56	strong
medium vs low	101.97 ms	123.00 ms	2.11e-40	strong

Fig. 30. Comparaciones estadísticas por frecuencia en la tercera auditoría.

Al unir las tres ejecuciones, se puede observar que la señal temporal existe y aparece de forma repetida, pero no todos los grupos tienen la misma utilidad para explotarla. La separación entre los grupos de baja frecuencia y los otros grupos es la más estable. En cambio, la diferencia entre alta y media frecuencia puede variar más dependiendo de la ejecución, del ruido del sistema y de la longitud de consulta. También se observa que las consultas cortas son las más útiles para construir una referencia temporal. Este comportamiento es coherente con lo obtenido previamente en la experimentación del apartado 6.4, donde se observó que las consultas cortas eran las que ofrecían una separación más útil entre grupos. En la validación realizada con la herramienta se vuelve a apreciar esa misma tendencia, ya que la longitud corta mantiene una diferenciación más clara y parece menos afectada por el ruido añadido por el propio texto de la consulta. Las consultas medias y largas también pueden mostrar diferencias, pero al incluir más contenido introducen más variabilidad en el procesamiento del sistema. Por este motivo, tiene sentido que el módulo de inferencia de términos únicamente use el comportamiento de las consultas cortas para construir sus fronteras temporales, al ofrecer una referencia más estable para comparar nuevos términos.

Por tanto, la validación de las auditorías demuestra que la herramienta funciona a nivel técnico y que los resultados que genera son interpretables. Las gráficas permiten observar el patrón temporal, las tablas estadísticas permiten medir la fuerza de la separación y las mediciones individuales permiten revisar errores sin perder trazabilidad. También se confirma que la

utilidad principal de la herramienta es convertir una señal temporal pequeña, difícil de observar individualmente, en un conjunto de resultados que el auditor puede analizar de forma ordenada.

La utilidad de la herramienta también se evaluó mediante el flujo de inferencia de términos, realizando pruebas para estudiar si los resultados obtenidos son útiles clasificando nuevos términos con las referencias temporales construidas previamente a partir de una auditoría. Como estas pruebas envían un menor número de consultas, resultan mucho más inestables, ya que la diferencia tan pequeña entre los grupos de frecuencia no tiene tiempo a estabilizarse y diferenciarse bien. Además, como esta prueba se realiza en base a una anterior, si la carga del sistema se ha modificado y los tiempos medidos son más altos, esta prueba no funcionará, generando resultados erróneos, como veremos a continuación.

En una ejecución realizada bajo condiciones estables, con el sistema recibiendo el menor ruido posible, la herramienta fue capaz de clasificar correctamente los términos evaluados. En la Fig. 31 se muestra la gráfica de clasificación de esa ejecución, donde se observa cómo las medias de los términos probados quedan situadas en relación con las fronteras temporales calculadas previamente. En este tipo de ejecución, la utilidad de la representación gráfica es clara, ya que permite ver de forma inmediata si un término queda próximo al grupo de alta presencia, al de baja presencia o si se mueve en una zona intermedia. La tabla asociada, incluida en la Fig. 32, complementa esa visualización mostrando la clasificación final, el número de mediciones válidas, la dispersión de los resultados y la referencia temporal más cercana. Con ello, la herramienta devuelve una etiqueta junto con el contexto necesario para que el auditor pueda interpretar por qué se ha tomado esa decisión.

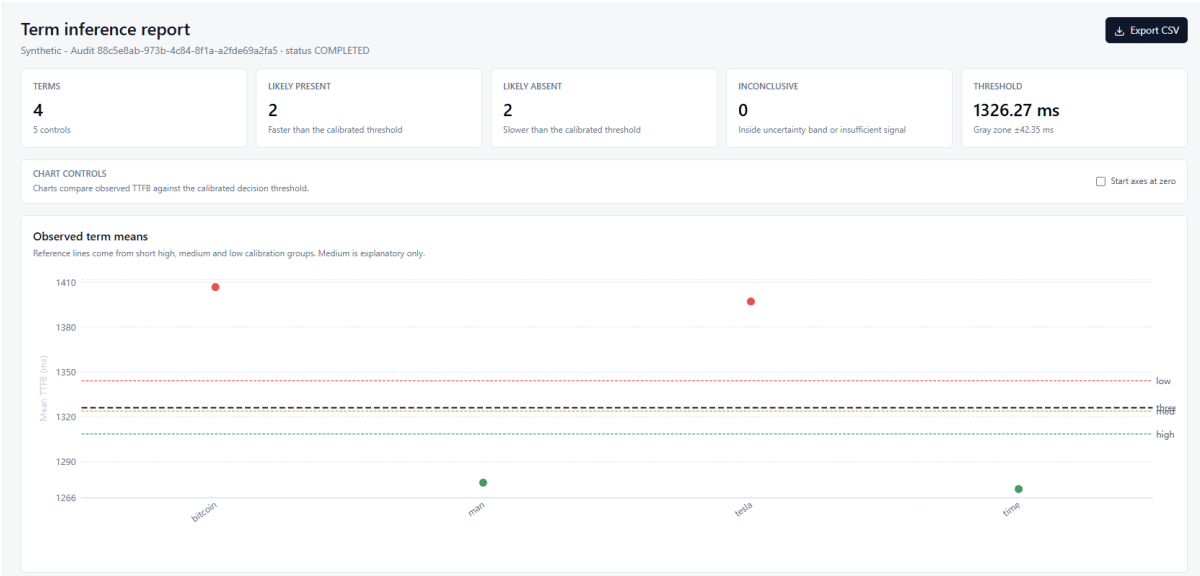


Fig. 31. Clasificación de términos en una ejecución estable de inferencia.

Term decisions

Term	Type	Classification	Mean TTFB	Std	Valid/total	Distance	Closest ref	Errors
bitcoin	term	likely_absent	1406.79 ms	50.87 ms	10/10	80.51 ms	low	0
man	term	likely_present	1275.92 ms	34.36 ms	10/10	-50.35 ms	high	0
tesla	term	likely_absent	1397.09 ms	44.06 ms	40/40	70.82 ms	low	0
time	term	likely_present	1271.63 ms	32.46 ms	10/10	-54.64 ms	high	0
malvexor	control	likely_absent	1437.66 ms	17.72 ms	8/8	111.39 ms	low	0
norithium	control	likely_absent	1435.56 ms	21.73 ms	8/8	109.29 ms	low	0
qrevanta	control	likely_absent	1431.01 ms	22.68 ms	8/8	104.74 ms	low	0
veltraxis	control	likely_absent	1423.96 ms	18.72 ms	8/8	97.69 ms	low	0
zenthorium	control	likely_absent	1424.49 ms	20.90 ms	13/13	98.22 ms	low	0

Fig. 32. Resultados tabulados de una ejecución estable de inferencia.

Sin embargo, también se realizaron varias ejecuciones adicionales en las que se introdujo un pequeño ruido sobre el sistema objetivo para comprobar cómo afectaba a este flujo. En estos casos se observó que la inferencia es bastante más frágil que la auditoría. Mientras que en las auditorías el patrón general seguía apareciendo incluso cuando había interferencias, en la inferencia de términos un pequeño desplazamiento global de las latencias podía alterar por completo la clasificación final. Esto ocurre porque las nuevas mediciones no se comparan contra una frontera construida en ese mismo instante, sino que se comparan contra una referencia calculada con anterioridad, por lo que cualquier cambio general en el sistema puede desplazar todos los términos hacia una zona distinta.

Este comportamiento puede verse en la Fig. 33 mostrada a continuación. En esa ejecución, las medias observadas de todos los términos quedaron claramente elevadas respecto a las fronteras calculadas a partir de la auditoría previa. Como consecuencia, todos los términos quedaron situados en la misma zona de decisión y la herramienta los clasificó como ausentes.

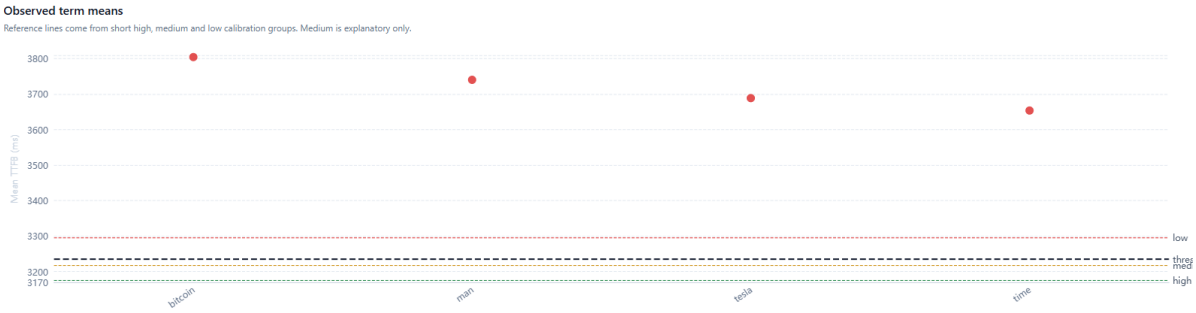


Fig. 33. Clasificación de términos en una ejecución de inferencia afectada por ruido.

La tabla de la Fig. 34 presentada a continuación confirma esta interpretación, pudiendo observar que los términos probados, incluyendo los términos de control, aparecen clasificados dentro del mismo grupo y con mediciones válidas suficientes para que el resultado no pueda atribuirse a una falta de rondas.

Term decisions

Term	Type	Classification	Mean TTFB	Std	Valid/total	Distance	Closest ref	Errors
bitcoin	term	likely_absent	3803.86 ms	135.31 ms	10/10	567.75 ms	low	0
man	term	likely_absent	3739.85 ms	160.68 ms	10/10	503.74 ms	low	0
tesla	term	likely_absent	3688.37 ms	115.69 ms	10/10	452.27 ms	low	0
time	term	likely_absent	3653.40 ms	108.02 ms	10/10	417.30 ms	low	0
malvexor	control	likely_absent	3796.38 ms	54.83 ms	2/2	560.27 ms	low	0
norithium	control	likely_absent	3601.49 ms	25.86 ms	2/2	365.38 ms	low	0
qrevanta	control	likely_absent	3906.37 ms	146.70 ms	2/2	670.27 ms	low	0
veltraxis	control	likely_absent	3813.33 ms	177.63 ms	2/2	577.23 ms	low	0
zenthorium	control	likely_absent	3382.78 ms	400.96 ms	7/7	146.68 ms	low	0

Fig. 34. Resultados tabulados de una ejecución de inferencia afectada por ruido.

La herramienta, por tanto, sigue funcionando correctamente desde el punto de vista técnico, ya que mide, registra y clasifica, pero la utilidad de esa clasificación depende mucho más de que el sistema permanezca estable entre la auditoría previa y la inferencia posterior.

La funcionalidad es útil y puede producir resultados interpretables cuando el sistema objetivo mantiene un comportamiento temporal cercano al observado durante la auditoría de calibración. Sin embargo, también es un flujo más delicado que la auditoría, ya que pequeñas variaciones globales del sistema pueden desplazar todas las mediciones y provocar

clasificaciones erróneas incluso cuando se ejecutan varias rondas. Por este motivo, la inferencia de términos debe entenderse como una funcionalidad dependiente de la estabilidad del target y de la calidad de la calibración previa, lo que encaja con las limitaciones ya identificadas en este trabajo.

6.5.4. VALIDACIÓN DEL ALCANCE DEFINIDO POR LOS REQUISITOS

Una vez comprobados los flujos principales de la herramienta, se revisó si el sistema desarrollado cubría el alcance definido en el capítulo de análisis de requisitos.

El primer punto validado fue que la herramienta permita preparar una auditoría sin modificar código ni ejecutar scripts manuales. Esto cubre una parte importante del alcance inicial, ya que el auditor puede iniciar sesión, configurar un sistema RAG externo, cargar un dataset con consultas clasificadas y dejar guardada la información necesaria para ejecutar pruebas posteriores. Además, se revisó la parte central del sistema, que es la ejecución de mediciones temporales, ya que los requisitos definidos al inicio del proyecto exigían que la herramienta pudiera lanzar peticiones contra un sistema externo, medir los tiempos de respuesta y conservar cada resultado de forma individual. Las pruebas realizadas muestran que este comportamiento se cumple, ya que cada consulta enviada genera un registro asociado a su sesión, con su Time-to-First-Byte, tiempo completo, estado HTTP, tamaño de respuesta y posible error.

Con esto también se validó que la ejecución de las pruebas quedase separada del uso normal de la interfaz, ya que era necesario hacer que las auditorías, al tardar mucho, no bloqueasen el resto del sistema. La ejecución mediante el worker permite que la interfaz actúe como capa de control y seguimiento, mientras que las mediciones se realizan en segundo plano. Con esto se cumple el requisito de reducir interferencias internas de la propia herramienta durante la recogida de datos.

El siguiente bloque validado fue el análisis de resultados, ya que la herramienta debe agrupar mediciones por frecuencia y longitud, calcular métricas estadísticas y generar visualizaciones para que el auditor pueda interpretar si existe separación temporal entre grupos. Esta validación comprobaría que el requisito de análisis y visualización de auditorías ha sido implementado con éxito en la herramienta. También en esta comprobación se verificó que los resultados pueden exportarse en CSV, lo que permite conservar las pruebas y analizarlas fuera de la aplicación si fuera necesario.

Todas estas comprobaciones que cubren los requisitos de las auditorías también se realizaron para verificar que los requisitos de la inferencia de términos estuvieran implementados correctamente, comprobando: exportación CSV, ejecución asíncrona, medición temporal, creación de inferencias y clasificación y análisis de resultados.

En cuanto a los requisitos no funcionales, la validación verificó todos los aspectos necesarios, muchos de ellos ya verificados en las pruebas de los requisitos funcionales, como el de seguridad y protección de acceso. En estas validaciones se comprobaron cosas como el funcionamiento de caja negra, ya que la herramienta solo necesita comunicarse con el sistema auditado mediante HTTP y no requiere acceso a su configuración interna. También se verificó la persistencia, porque los targets, datasets, sesiones, mediciones, errores, semillas y resultados deben quedar guardados para su consulta posterior. Por último, se comprobó la ejecución local controlada, validada mediante Docker Compose, que permite levantar frontend, backend, base de datos, cola y worker sin configurar cada servicio por separado.

Con esta revisión se puede considerar que el alcance definido por los requisitos queda cubierto. La herramienta permite preparar una auditoría, ejecutarla contra un sistema RAG externo, medir la latencia, analizar los resultados, exportarlos y realizar una inferencia posterior de términos.

6.6. LIMITACIONES OBSERVADAS

Durante el desarrollo experimental se han observado varias limitaciones que afectan tanto a la interpretación de los resultados como al alcance real del ataque. Estas limitaciones no invalidan la hipótesis, pero sí ayudan a entender en qué condiciones el ataque funciona mejor y en cuáles debe interpretarse con más cuidado.

La principal limitación del sistema es que las diferencias temporales se ven afectadas tanto por factores del dispositivo, como la carga de la máquina, el estado de la GPU, el tamaño del contexto recuperado y el comportamiento interno de ChromaDB, como por procesos del propio pipeline RAG, como la vectorización de las consultas, la recuperación de documentos, la construcción del prompt y la generación de la respuesta. Aunque los experimentos muestran una relación clara entre frecuencia semántica y latencia, este ruido generado por los sistemas hace que el tiempo medio estudiado se eleve mucho, en algunos casos variando su comportamiento, y que se necesiten más consultas para diferenciar bien los grupos estudiados. Esta limitación se presenta tanto en las auditorías como en las inferencias, pero afecta en mayor medida a las inferencias al necesitar más rondas para poder clasificar correctamente los términos, ya que las auditorías, al ser largas por naturaleza, ya envían suficientes consultas. Esta limitación va ligada a otra que es la necesidad de enviar bastantes mediciones para realizar una prueba completa, ya que la señal aparece al comparar grupos y observar tendencias repetidas.

Relacionada con el valor de k utilizado por el sistema atacado, encontramos una limitación más. La mayoría de los sistemas utilizan k similares a las probadas, pero en algunos casos, por la necesidad de respuestas cortas y rápidas, los sistemas podrían llegar a usar valores de k muy bajos. Es en estos sistemas en donde esta limitación aparece, ya que la diferencia temporal puede ser demasiado pequeña para distinguir bien los grupos. En la mayoría de sistemas, esta limitación no debería aparecer, ya que al usar valores de k en el rango normal esto no ocurre, pero es importante entender incluso las limitaciones del sistema que se pueden dar en casos aislados o extremos.

El uso de cachés por parte de algunos sistemas puede producir otra limitación, ya que el comportamiento de la prueba puede variar mucho dependiendo de la implementación de la misma, cambiando si se trata de una caché exacta, una caché semántica o una caché interna del proveedor. En algunos casos, la caché puede generar muchos más aciertos que en otros, pudiendo generar resultados no concluyentes por el problema explicado en el apartado 6.2.6, el cual explica que, aunque se muestren diferencias entre consultas, pueden deberse a un mal diseño de las pruebas. Esta limitación no debería aparecer demasiado si la prueba está bien diseñada, forzando que no se generen aciertos en caché al separar mucho las consultas similares.

También existe una limitación asociada al corpus y al entorno usado, ya que, por limitaciones de tiempo, la investigación se ha realizado principalmente sobre PG-19, ChromaDB y el modelo de embeddings all-MiniLM-L6-v2. Este entorno permite estudiar el fenómeno con un

corpus grande y variado, pero no demuestra automáticamente que todos los motores vectoriales, todos los modelos de embeddings y todos los dominios documentales se comporten exactamente igual. Como se ha explicado, el entorno utilizado es uno que no ayuda a la herramienta, por lo que en condiciones más favorables la herramienta debería rendir mejor, pero, por ejemplo, en un corpus empresarial más homogéneo o una base de datos vectorial distinta, se podrían generar comportamientos no estudiados en esta investigación.

Por último, es importante comentar que la inferencia de términos es la parte más sensible de la herramienta, ya que depende de que exista una auditoría previa de buena calidad y de que el target se mantenga estable entre la calibración y la prueba posterior. Si el sistema cambia su carga, su caché, su modelo o su configuración, las fronteras temporales pueden dejar de representar correctamente el comportamiento real del entorno, haciendo que la inferencia sea algo más orientativo que una prueba con resultados definitivos.

En conjunto, estas limitaciones ayudan a acotar el resultado obtenido. La investigación demuestra que existe una diferencia temporal medible en la recuperación vectorial y que esta puede mantenerse al añadir componentes propios de un sistema RAG real. También muestra que la herramienta desarrollada permite automatizar ese proceso contra un sistema externo funcional. Sin embargo, la efectividad del ataque depende de algunas configuraciones como la estabilidad del entorno atacado, el diseño del banco de consultas, el ruido introducido por el resto del pipeline o la implementación de caché, entre otras. Por este motivo, el canal lateral estudiado debe entenderse como una vulnerabilidad explotable bajo ciertas condiciones.

7. ASPECTOS ECONÓMICOS Y SOCIALES

Una vez descrito el diseño, la implementación y la evaluación del sistema, este capítulo analiza el proyecto desde una perspectiva no puramente técnica, ya que, aunque el objetivo principal del trabajo ha sido estudiar un canal lateral temporal en sistemas RAG y desarrollar una herramienta para auditarlo, también es necesario valorar el coste aproximado del desarrollo y su posible impacto social y ético. Este análisis es importante porque la herramienta desarrollada podría aplicarse en contextos reales de auditoría de sistemas de inteligencia artificial, por lo que su uso debe entenderse dentro de un marco responsable. El hecho de que permita estudiar fugas de información mediante latencias hace que tenga un valor defensivo claro, pero también obliga a considerar los riesgos asociados a un posible uso indebido [35], [36].

En este capítulo se presenta primero una estimación económica del proyecto considerando los recursos técnicos empleados y después se analiza el impacto social y ético del trabajo, prestando especial atención a la privacidad, la autorización de las pruebas y el carácter de doble uso de la herramienta.

7.1. PRESUPUESTO DEL PROYECTO

El presupuesto del proyecto se ha elaborado considerando el coste que tendría desarrollar este trabajo en un entorno profesional. Por lo tanto, además de reflejar los gastos directos asumidos durante la realización del Trabajo de Fin de Grado, también se estima el valor económico del esfuerzo necesario para llevar a cabo la investigación, el diseño, la implementación, las pruebas y la documentación de la herramienta desarrollada. Para realizar esta estimación se han separado dos bloques principales. El primero corresponde al coste humano, que representa la mayor parte del presupuesto, ya que el proyecto requiere análisis técnico, experimentación,

desarrollo software y validación. El segundo recoge los costes asociados a infraestructura, herramientas y recursos necesarios para ejecutar las pruebas y levantar la aplicación.

Las cantidades mostradas son estimaciones orientativas y se expresan en euros. En el caso del coste humano, se han considerado perfiles técnicos equivalentes a los que participarían en un proyecto profesional de estas características.

El coste humano incluye las horas dedicadas a la investigación del problema, el diseño experimental, la ejecución de pruebas, el análisis de resultados, la construcción de la herramienta y la elaboración de la memoria. Dado que el proyecto combina investigación en ciberseguridad con desarrollo de software, se han considerado actividades con distintos niveles de especialización. En la Tabla V se muestra la distribución de horas y el coste de cada una según la actividad realizada.

Tabla V. ESTIMACIÓN DEL COSTE HUMANO DEL PROYECTO.

Actividad	Horas estimadas	Coste/hora	Coste estimado
Revisión del estado del arte y análisis de la brecha de investigación	70 h	30 €	2.100 €
Diseño experimental y definición del modelo de amenaza	50 h	30 €	1.500 €
Preparación del corpus, generación de consultas y ejecución experimental	300 h	30 €	9.000 €
Análisis estadístico e interpretación de resultados	100 h	30 €	3.000 €
Análisis de requisitos y diseño de la herramienta	60 h	35 €	2.100 €
Implementación del backend, worker y persistencia	120 h	35 €	4.200 €
Implementación del frontend e interfaz de usuario	85 h	35 €	2.975 €
Validación funcional y pruebas de la herramienta	45 h	30 €	1.350 €
Redacción, revisión y preparación de la memoria	230 h	25 €	5.750 €
Coordinación técnica y revisión global del proyecto	20 h	50 €	1.000 €
Total coste humano	**1.080 h**		**32.975 €**

La mayor carga de horas se concentra en la ejecución de la investigación, ya que esta parte consistió en un proceso progresivo con varias fases de experimentación. Fue necesario preparar el corpus, construir la base vectorial, generar y validar el banco de consultas, ejecutar pruebas repetidas con distintas configuraciones, analizar los resultados obtenidos y repetir parte de las ejecuciones cuando aparecían efectos de ruido. Esta dedicación explica que el coste humano esté especialmente vinculado a la fase experimental, ya que de ella dependía comprobar si la hipótesis del canal lateral temporal tenía una base medible antes de trasladarla al diseño y validación de la herramienta.

Además del coste humano, el proyecto requiere recursos técnicos para desarrollar, probar y ejecutar la herramienta. La mayor parte de las tecnologías empleadas son de código abierto, por lo que no se han imputado costes de licencia para el backend, frontend, base de datos, sistema de colas o librerías de análisis. Sí se han incluido costes asociados a infraestructura de cómputo, almacenamiento y uso del equipo local. La Tabla VI recoge la separación de costes estimados asociados a infraestructura y otros recursos.

Tabla VI. ESTIMACIÓN DE COSTES DE INFRAESTRUCTURA Y OTROS RECURSOS.

Concepto	Descripción	Coste estimado
Infraestructura cloud para experimentación	Uso de entorno cloud para ingesta del corpus, ejecución de pruebas y validación del RAG en RunPod.	160 €
Almacenamiento asociado a pruebas	Espacio necesario para bases vectoriales persistentes, datasets y resultados experimentales.	40 €
Equipo local amortizado	Uso proporcional de un equipo personal para desarrollo, documentación y ejecución local.	150 €
Repositorio y control de versiones	Uso de repositorio Git para mantener el código fuente del proyecto.	0 €
Licencias software	Uso de tecnologías y librerías sin coste de licencia comercial.	0 €
Total infraestructura y otros recursos		**350 €**

El uso de Docker Compose reduce el coste de instalación y despliegue, ya que permite levantar todos los servicios necesarios de forma local sin contratar una infraestructura permanente. También se ha evitado depender de herramientas comerciales de pago, lo que mantiene bajo el coste de licencias.

El presupuesto total, mostrado en la Tabla VII, se calcula sumando el coste humano, los costes de infraestructura y una partida de contingencia. Esta contingencia cubre posibles desviaciones en la ejecución de pruebas, ajustes de la herramienta, consumo adicional de infraestructura o revisiones finales no previstas.

Tabla VII. RESUMEN DEL PRESUPUESTO TOTAL DEL PROYECTO.

Concepto	Coste
Coste humano	32.975 €
Infraestructura y otros recursos	350 €
Subtotal	**33.325 €**
Contingencia y gastos indirectos, 10%	3.333 €
Total sin IVA	**36.658 €**
IVA, 21%	7.698 €
Total con IVA	**44.356 €**

El coste final estimado del proyecto asciende a 36.658 € sin IVA y a 44.356 € con IVA incluido. La mayor parte del presupuesto corresponde al coste humano, lo que es coherente con el

trabajo, ya que su valor principal está en el análisis técnico, la experimentación, la implementación de la herramienta y la interpretación de los resultados obtenidos. Este presupuesto refleja el coste profesional aproximado de desarrollar una herramienta de auditoría como la presentada en esta memoria, incluyendo la investigación previa necesaria para justificar su diseño y validar su utilidad.

7.2. ANÁLISIS DE IMPACTO SOCIAL Y ÉTICO

El proyecto desarrollado tiene un impacto social y ético relevante porque se sitúa en un punto delicado entre la mejora de la seguridad y la posible explotación de una vulnerabilidad. La herramienta se ha diseñado con una finalidad defensiva, permitiendo auditar sistemas RAG para detectar si podrían estar filtrando información mediante diferencias de latencia. Sin embargo, la misma técnica que permite a una organización evaluar sus propios sistemas también podría ser usada de forma ofensiva por un atacante para intentar inferir la presencia de información sensible en una base de conocimiento [35], [36].

Desde el punto de vista social, la investigación puede tener un efecto positivo si se aplica dentro de auditorías autorizadas. Cada vez más organizaciones incorporan sistemas RAG para consultar documentación interna, procedimientos, contratos, informes o datos técnicos. Estos sistemas suelen percibirse como seguros porque los documentos no viajan por internet ni están expuestos a terceros; además, el usuario solo interactúa con un chatbot y no accede directamente a los documentos. Sin embargo, este trabajo muestra que la seguridad de un RAG no depende únicamente de los permisos de lectura o de las respuestas generadas por el modelo, sino también de señales indirectas o canales laterales que pueden aparecer durante el proceso completo por el que pasa el sistema para responder al usuario. Por esto, detectar este tipo de fugas es importante, ya que ayuda a mejorar la protección de la información y a diseñar arquitecturas más seguras, realizando procedimientos de defensa que rompan estos canales sin interferir en la experiencia de usuario.

El principal riesgo ético aparece porque la herramienta automatiza un proceso de medición que podría ser usado para extraer información sin permiso. Un usuario malintencionado podría lanzar consultas repetidas contra un chatbot corporativo y observar si determinados términos o combinaciones de palabras generan patrones temporales diferentes. En un escenario real, esto podría afectar a documentos de recursos humanos, información financiera, datos legales, proyectos internos o cualquier otra información que no debería ser inferible por personas sin autorización. Aunque el contenido del documento no se revele directamente, saber que un documento existe o contiene alguna información concreta ya puede suponer una fuga de privacidad importante, por lo que el uso correcto de la herramienta debe quedar limitado a entornos propios, controlados y autorizados, ya que esta se plantea como un instrumento de evaluación defensiva. Su objetivo es permitir que un auditor compruebe si una arquitectura RAG presenta señales temporales explotables antes de que esas señales puedan ser usadas por un atacante real, por lo que el valor principal del proyecto está en anticipar un riesgo y ofrecer una forma de medirlo, y no en facilitar el acceso no autorizado a información privada.

Por otro lado, se debe tener en cuenta que la propia ejecución de auditorías genera datos sensibles, ya que los resultados de una prueba pueden incluir endpoints internos, headers, consultas diseñadas para comprobar la presencia de información y mediciones que revelan el comportamiento del sistema auditado. Por este motivo, estos datos deben tratarse como información confidencial y no como simples resultados técnicos, haciendo que en un uso real de la herramienta, los CSV exportados, las sesiones guardadas y los datasets de prueba se

almacenen en entornos controlados, se compartan solo con personal autorizado y se eliminen cuando ya no sean necesarios.

Desde el punto de vista de la privacidad, el trabajo pone de manifiesto un problema importante en la adopción de sistemas de inteligencia artificial generativa en empresas, ya que no basta con controlar qué responde el modelo, sino que también es necesario estudiar cómo responde. Las latencias, los tiempos de generación, los aciertos en caché o las diferencias en la recuperación documental pueden convertirse en señales útiles para un atacante, por lo que este enfoque amplía la forma tradicional de entender la seguridad en sistemas RAG, ya que obliga a considerar no solo el contenido visible, sino también el comportamiento observable desde el exterior. El proyecto también entra en el debate ético sobre la publicación y comunicación de vulnerabilidades. Mostrar una técnica de ataque puede aumentar el riesgo de que otros intenten reproducirla, pero ocultarla por completo impide que las organizaciones conozcan el problema y puedan protegerse. En este caso, el trabajo se orienta hacia una divulgación responsable: se explica el riesgo, se construye una herramienta de auditoría y se enmarca su uso dentro de pruebas autorizadas, lo que permite aportar conocimiento útil sin presentar la herramienta como un mecanismo para vulnerar sistemas ajenos [36].

En conjunto, el impacto social y ético del proyecto depende en gran medida del contexto en el que se use. Si se emplea sin autorización, podría convertirse en una herramienta ofensiva para inferir información sensible. Si se usa dentro de auditorías defensivas, puede ayudar a detectar un tipo de fuga difícil de observar con métodos tradicionales. Por tanto, el proyecto debe entenderse como una contribución a la seguridad de sistemas RAG, siempre que su aplicación se limite a entornos autorizados, con una interpretación prudente de los resultados y con medidas adecuadas para proteger los datos generados durante las pruebas.

8. CONCLUSIONES Y TRABAJO FUTURO

Este capítulo recoge el cierre del Trabajo de Fin de Grado. Después de haber estudiado el estado del arte, realizado la experimentación necesaria, diseñado la herramienta, implementado sus módulos principales y validado su funcionamiento, es necesario valorar qué se ha conseguido y qué partes quedan abiertas para seguir trabajando en el futuro.

El proyecto partía de la idea de comprobar si aparecían diferencias de latencia en la recuperación vectorial de los sistemas RAG y si estas podían llegar a revelar información sobre el contenido indexado. Por esto, primero se realizó una investigación experimental para observar si la señal temporal existía y si seguía apareciendo al añadir los componentes propios de un sistema más realista. Una vez encontrada esta diferencia temporal, se trabajó en el desarrollo de una herramienta que permite automatizar este tipo de auditorías mediante un enfoque de caja negra y llamadas HTTP.

Por esta división entre investigación y herramienta, las limitaciones de las que se habla en este capítulo se tratan de forma separada. Las conclusiones valoran el cumplimiento de los objetivos iniciales del proyecto, mientras que las líneas de trabajo futuras diferencian entre las posibles ampliaciones de la investigación y las mejoras que podrían incorporarse a la herramienta. Esta separación es importante porque la investigación se ha detenido en un punto en el que ya se podían obtener conclusiones, pero todavía quedaban muchas variantes por estudiar. En cambio, las mejoras de la herramienta dependen en gran medida de cómo evolucione esa investigación.

8.1. CONCLUSIONES Y CONSECUCIÓN DE OBJETIVOS

Este proyecto tiene un objetivo principal que engloba todo el desarrollo del mismo: estudiar la existencia de un canal lateral temporal en arquitecturas RAG y desarrollar una herramienta capaz de medirlo de forma automatizada. A partir del desarrollo realizado y de los resultados obtenidos en la fase de pruebas, se puede considerar que este objetivo se ha cumplido, ya que el proyecto ha permitido observar que la recuperación vectorial introduce diferencias temporales medibles entre consultas asociadas a información indexada y consultas asociadas a información ausente o poco relacionada con el corpus. Además, la herramienta desarrollada permite automatizar este tipo de auditorías contra un sistema autorizado del que solo se conoce el endpoint y el formato de la petición HTTP.

El primer objetivo específico consistía en analizar el estado del arte e identificar la brecha de investigación. Este objetivo se ha cumplido mediante la revisión de trabajos relacionados con privacidad en sistemas RAG, inferencia de membresía, ataques de canal lateral en modelos de lenguaje y vulnerabilidades asociadas a mecanismos de caché [7]-[15]. Esta revisión permitió situar el trabajo dentro de la literatura existente y observar que la latencia del componente de recuperación vectorial no había sido estudiada de forma específica como posible origen de una fuga temporal.

El segundo objetivo específico era diseñar y ejecutar una metodología experimental progresiva para estudiar la latencia en sistemas RAG. Este objetivo se ha cumplido mediante la definición del modelo de amenaza, la identificación de variables relevantes, la construcción del corpus y del banco de consultas, la validación inicial en un entorno reducido, el escalado a un corpus mayor, la evaluación de la recuperación vectorial aislada, el análisis de un pipeline RAG completo y el estudio del efecto de mecanismos de caché. Esta progresión permitió comprobar en qué condiciones la señal temporal era más clara y en qué escenarios quedaba más afectada por el ruido.

El tercer objetivo específico consistía en diseñar e implementar una herramienta web de auditoría para evaluar sistemas RAG externos desde una perspectiva de caja negra. Este objetivo también se ha cumplido, ya que la herramienta permite registrar targets, cargar datasets, crear auditorías, ejecutar mediciones en segundo plano, medir el Time-to-First-Byte, guardar resultados, generar perfiles de calibración, clasificar resultados e interpretar las mediciones mediante gráficas y métricas estadísticas. La separación entre interfaz, backend, base de datos, cola de trabajos y worker permitió convertir los scripts experimentales iniciales en un flujo más ordenado, persistente y repetible.

El cuarto objetivo específico era construir un entorno RAG externo que permitiese validar correctamente la herramienta desarrollada. Este objetivo se ha cumplido mediante la preparación de un sistema RAG accesible por HTTP, con una base documental controlada, un modelo de embeddings, una base vectorial y un modelo de lenguaje conectado al pipeline de respuesta. Gracias a este entorno fue posible ejecutar auditorías e inferencias desde la herramienta final y comprobar que los flujos principales funcionaban fuera de un script local, en un escenario más cercano al uso real de la aplicación.

El quinto objetivo específico consistía en identificar las limitaciones del enfoque propuesto y analizar su impacto desde una perspectiva defensiva, ética y social. Este objetivo se ha tratado tanto en el análisis de resultados como en los apartados finales de la memoria. Se han identificado limitaciones relacionadas con el ruido de red, la estabilidad del sistema auditado,

la dependencia de la calibración y la sensibilidad de la inferencia de términos. También se ha destacado que la herramienta debe usarse únicamente en entornos autorizados y con una finalidad defensiva.

En conjunto, el trabajo ha conseguido unir una parte de investigación con una parte de desarrollo de software. La investigación permitió detectar y estudiar la señal temporal, mientras que la herramienta permitió automatizar su medición y análisis, lo que transforma una hipótesis técnica en un sistema que puede ejecutarse, repetirse y evaluarse sobre distintos targets. Además, es importante mencionar que la herramienta desarrollada también cumple con el objetivo defensivo del proyecto, ya que su finalidad es ayudar a estudiar si una arquitectura propia puede filtrar información mediante tiempos de respuesta.

Con todos los objetivos del proyecto cubiertos, queda claro que se puede concluir que la hipótesis planteada es real en los escenarios estudiados y puede aparecer en mayor o menor medida en sistemas que incluyen recuperación vectorial. No se puede afirmar que todos los sistemas RAG presenten la misma intensidad de señal, pero sí que la recuperación vectorial introduce un comportamiento temporal que debe tenerse en cuenta dentro de su análisis de seguridad. Además, con todo el contexto estudiado para la realización del proyecto, se puede observar que los sistemas basados en inteligencia artificial generativa todavía están muy lejos de poder considerarse seguros por defecto, ya que son sistemas nuevos, complejos, formados por muchas capas distintas y dinámicos, donde cada componente añadido para mejorar la precisión o facilitar el uso puede convertirse también en una nueva superficie de ataque. Durante el estado del arte se han analizado ataques que explotan las respuestas del modelo, las probabilidades de los tokens, el presupuesto de generación, el tráfico de red, la KV-Cache y las cachés semánticas. Muchos de estos ataques no existían o no se conocían hace apenas unos años, lo que demuestra que el problema no está solo en fallos concretos de implementación, sino en que todavía no se entiende por completo cómo proteger arquitecturas tan recientes. En este trabajo se ha estudiado la recuperación vectorial, que es otro punto más de la superficie de ataque, entendiendo que el hecho de que una base de datos vectorial pueda revelar información a través de sus tiempos de respuesta es peligroso, ya que incluso los componentes que parecen puramente técnicos o internos pueden terminar filtrando información sensible, lo que lleva a la conclusión de que la seguridad de un sistema RAG no puede limitarse a la seguridad de los sistemas tradicionales como revisar permisos o validar usuarios. En estos sistemas, el no mostrar ningún dato confidencial en la respuesta no significa que haya alguna señal que esté dando pistas sobre lo que tiene almacenado, ya que las latencias, los aciertos en caché, el comportamiento del retriever o las diferencias entre consultas aparentemente similares pueden actuar como señales indirectas para un atacante. Por tanto, proteger estos sistemas requiere mirar más allá del contenido generado y estudiar también cómo se comporta toda la arquitectura cuando procesa una consulta [35], [36].

Por otro lado, es importante entender que este ataque no será el último que se encuentre, ya que la velocidad con la que están evolucionando los sistemas RAG y las optimizaciones de los mismos hace que aparezcan continuamente nuevas técnicas para atacarlos. Lo que hoy se ve como una optimización normal, mañana puede convertirse en una vulnerabilidad si permite distinguir estados internos del sistema. Por eso, las empresas que adopten este tipo de soluciones no deberían tratarlas como herramientas cerradas y definitivamente seguras, sino como sistemas vivos que necesitan revisión continua.

En este contexto, las auditorías de seguridad también deben evolucionar, ya que no basta con hacer pruebas tradicionales de autenticación, control de acceso, cifrado de datos o protección

de endpoints. En sistemas RAG es necesario analizar la privacidad desde una perspectiva más amplia, incluyendo canales laterales, tiempos de respuesta, funcionamiento del índice vectorial, comportamiento de las cachés y estabilidad del pipeline completo. La defensa de estos sistemas requiere estar al día de las investigaciones recientes, adaptar las medidas de protección, auditar de distintas formas los sistemas y asumir que los ataques pueden venir de señales que no contienen texto sensible de forma directa, pero que permiten inferirlo. Por todo esto, el trabajo realizado refuerza la idea de que la seguridad en inteligencia artificial generativa todavía está en una fase temprana, ya que la adopción empresarial de sistemas RAG avanza muy rápido, pero la comprensión de sus riesgos no avanza siempre al mismo ritmo. Este desfase puede provocar que muchas organizaciones integren documentación interna en sistemas aparentemente controlados sin conocer todos los canales por los que esa información puede filtrarse. Por tanto, la aportación de este proyecto se sitúa precisamente en el punto de mostrar que existe una vía adicional de fuga y proponer una herramienta para que pueda auditarse antes de que sea explotada de forma ofensiva.

8.2. LIMITACIONES DEL TRABAJO DESARROLLADO

Aunque el trabajo ha permitido demostrar la viabilidad del canal lateral temporal y desarrollar una herramienta capaz de automatizar su medición, el alcance del proyecto debe interpretarse dentro del contexto temporal establecido. El objetivo principal era comprobar si existía una señal temporal explotable en sistemas RAG y construir una aplicación que permitiera auditarla desde fuera. Por tanto, el resultado obtenido no debe entenderse como una solución cerrada para todos los sistemas RAG, sino como una primera aproximación técnica a un problema de seguridad que todavía tiene mucho margen de estudio. En este proyecto se ha abordado una primera fase de identificación y validación de una vulnerabilidad no estudiada. Este tipo de trabajo exige más esfuerzo inicial que una ampliación sobre una línea ya consolidada, porque obliga a definir el problema, construir el entorno experimental, diseñar las mediciones y comprobar si la señal observada tiene sentido. Por esto, el trabajo presenta varias limitaciones y deja abiertas futuras líneas de investigación, ya que no era posible estudiar todas las variantes del ataque dentro del alcance temporal del TFG.

Una primera limitación del trabajo es que la investigación se ha centrado en demostrar la existencia del fenómeno en un entorno controlado, lo que ha sido necesario para poder medir correctamente el comportamiento del sistema y separar las fases del ataque, pero también implica que las conclusiones deben trasladarse con cuidado a otros entornos. En un sistema real pueden existir más capas intermedias, políticas internas de caché, balanceadores, límites de tasa, variaciones de carga o configuraciones concretas que no se han reproducido completamente en los experimentos. El trabajo demuestra que la fuga temporal puede aparecer y que puede medirse, pero no pretende afirmar que todos los sistemas RAG vayan a presentar la misma intensidad de señal.

La automatización de las pruebas es algo que se ha buscado en este proyecto con la realización de la herramienta, pero es verdad que esta sigue teniendo una limitación, ya que, aunque la herramienta permita lanzar auditorías, guardar resultados, generar gráficas y realizar inferencias posteriores, sigue siendo necesario que el auditor entienda qué está probando. La calidad del banco de consultas, la elección de términos, la separación entre categorías y la interpretación de los resultados siguen dependiendo del criterio técnico de la persona que usa la aplicación. Esto es importante porque el ataque no se basa en una única petición, sino en tendencias temporales observadas a partir de muchas mediciones, por lo que la herramienta reduce el trabajo manual, pero no elimina la necesidad de diseñar bien la prueba.

Otra limitación del proyecto es que la validación de la herramienta se ha realizado con un RAG externo preparado para el trabajo. Esto es lo que ha permitido probar la aplicación en un escenario black-box más realista, ya que la herramienta se ha comunicado con una API desplegada en RunPod, pero este sigue siendo un entorno controlado por el propio proyecto. Para verificar profesionalmente la herramienta, lo mejor sería realizar estas pruebas en un entorno profesional de verdad, pero por motivos obvios este paso no se ha podido realizar en este trabajo. Por lo tanto, aunque se ha imitado un sistema profesional lo mejor posible, habría que estudiar la efectividad de la herramienta en su aplicación contra múltiples sistemas RAG reales de empresas distintas.

Por otro lado, la inferencia de términos debe tratarse como una funcionalidad menos concluyente dentro de la herramienta. Su objetivo es ir un paso más allá de la auditoría general y probar si, a partir de una calibración previa, se puede clasificar el comportamiento temporal de términos concretos. Sin embargo, esta parte depende mucho de la calidad de las consultas y de la estabilidad del sistema auditado. Por ese motivo, sus resultados deben interpretarse como señales orientativas y no como afirmaciones absolutas sobre la presencia de un término en la base documental, lo que no invalida la funcionalidad, pero sí marca claramente su uso correcto.

Por último, el trabajo queda limitado por el tiempo disponible. Se ha llegado hasta el punto en el que se demuestra el ataque, se estudian varios escenarios experimentales y se desarrolla una herramienta funcional para medirlo. Sin embargo, muchas decisiones podrían ampliarse con más tiempo probando más corpus, más motores vectoriales, más modelos, más configuraciones de caché, más defensas y más entornos reales, lo que se explica más detalladamente en el siguiente apartado. Por todo esto queda claro que el valor principal del proyecto está en haber identificado, medido y automatizado una vía de fuga temporal en sistemas RAG, dejando una base clara para investigaciones posteriores.

8.3. LÍNEAS DE TRABAJO FUTURAS

El trabajo realizado demuestra que es posible observar diferencias temporales en sistemas RAG y que esas diferencias pueden actuar como canal lateral para inferir información sobre el contenido de una base documental. Aun así, este estudio debe entenderse como un punto de partida, ya que la investigación simplemente ha permitido comprobar el ataque en un entorno controlado, con un corpus concreto, una base vectorial determinada y varios escenarios de ejecución, pero todavía quedan muchas líneas abiertas para estudiar hasta qué punto este comportamiento se mantiene en otros sistemas y bajo otras configuraciones. Por esto, todas las elecciones realizadas para desarrollar el ataque, como pueden ser el corpus o la base de datos vectorial, generan nuevas líneas de trabajo experimentales que nos ayudarán a entender los puntos claves del ataque y cómo de aplicable es este en entornos con diferentes tecnologías o tipos de datos vectorizados.

La primera línea futura importante para investigar sería la ampliación de la experimentación a otros corpus. En este trabajo se ha usado principalmente PG-19, que está formado por textos literarios en inglés, lo cual nos ha permitido trabajar con una base documental grande y con términos que pueden clasificarse por frecuencia semántica, pero no representa todos los usos reales de un sistema RAG. Sería interesante repetir los experimentos con documentación técnica, manuales internos, contratos, artículos científicos, tickets de soporte o bases de conocimiento empresariales. Cada tipo de corpus puede producir distribuciones vectoriales distintas, por ejemplo, un corpus muy repetitivo puede hacer que muchas consultas parezcan

presentes aunque no lo estén directamente y un corpus técnico, en cambio, puede tener términos muy específicos que faciliten más la inferencia. Comparar estos escenarios ayudaría a saber si el canal lateral depende mucho del dominio o si aparece de forma estable en sistemas RAG variados, pudiendo clasificar en qué tipo de empresas estas auditorías se pueden aplicar con mayor facilidad.

De la misma manera, sería importante el estudio de otras bases de datos vectoriales y, aún más importante, el estudio de diferentes algoritmos de recuperación documental. En este proyecto se ha trabajado la mayor parte del tiempo con la tecnología de ChromaDB, ya que es una de las bases vectoriales más utilizadas actualmente, pero una buena investigación a realizar sería hacer estas mismas pruebas a sistemas externos que utilicen bases de datos diferentes. En este caso, esta investigación no solo nos ayudaría a observar cuán importante es la implementación que tenga una base de datos u otra, comprobando las distintas implementaciones de cachés o los diferentes tiempos de vectorización de consultas, sino que también nos ayudaría a estudiar lo más importante del ataque, que son los algoritmos de recuperación. La mayoría de bases de datos vectoriales utilizan el algoritmo HNSW por lo comentado en apartados anteriores, pero cada una lo implementa de distintas formas, realizando cambios en las tecnologías de filtrado, en la escalabilidad de la RAM, en la infraestructura de los clústeres, en los algoritmos de modificación de datos y en muchas otras partes del algoritmo. Por tanto, es interesante probar otras bases de datos vectoriales que también utilicen este algoritmo, como son Milvus y Pinecone, observando si la implementación del mismo es importante para la efectividad del ataque. Además de estas pruebas sobre el algoritmo HNSW, que mejorarían el conocimiento de si este algoritmo es atacable con sus distintas implementaciones, también sería importante probar otros algoritmos menos utilizados, como los comentados en el apartado de hipótesis y modelo de amenaza.

De la misma manera que en el estudio de las bases de datos, todas las variables aplicadas en el pipeline de un sistema RAG generan nuevas líneas de trabajo interesantes, pudiendo estudiar cosas como el modelo de embeddings, la fragmentación de documentos para la indexación, el LLM utilizado y el sistema de caché implementado.

En cuanto al modelo de embeddings, en la investigación se ha usado all-MiniLM-L6-v2, que es un modelo ligero y común para crear representaciones semánticas. Este modelo nos ha ayudado a poder realizar más pruebas al ser más pequeño, pero sería interesante probar a realizar las mismas pruebas con sistemas que utilicen modelos más grandes, como puede ser el text-embedding-3-small de OpenAI [37], que tiene 1536 dimensiones en vez de 384, 8192 tokens en vez de 256, está adaptado a diferentes lenguajes en vez de únicamente al inglés y tiene mayor comprensión sobre matices semánticos importantes. Este tipo de modelos no se ha utilizado para este proyecto, ya que en la mayoría de ocasiones conllevan un pago por uso, además de un tiempo de ejecución algo mayor al tener que usarlos por API. En esta línea de trabajo se podrían observar cambios en la distribución de distancias entre consultas y documentos, lo que afectaría a la recuperación, además de poder observar cambios en los tiempos que tarda el sistema en buscar y construir el contexto. Probar modelos de embeddings distintos permitiría entender si el ataque depende de un modelo concreto o si la señal temporal se mantiene aunque cambie la representación semántica.

Por otro lado, la forma de fragmentar el corpus abre otra gran ventana de estudio, ya que hay muchas formas de realizar esta fragmentación y algunos sistemas incluso utilizan estrategias híbridas que combinan varias de estas. En este trabajo se trabajó con chunks de tamaño fijo por la sencillez de su implementación, pero algunos sistemas RAG implementan fragmentaciones

recursivas en las que el sistema fragmenta el texto según sus separadores del lenguaje natural. Con esta separación, el texto es partido en párrafos si estos son suficientemente pequeños como para caber en un fragmento, dándole más consistencia a la separación. Por otro lado, en las fragmentaciones también hay varias configuraciones que se podrían estudiar, como son el tamaño del chunk, que cambia la cantidad de documentos almacenados, o la cantidad de solapamiento por chunk, que genera más fragmentos cuanto mayor sea este valor al repetir información. Por tanto, una futura investigación debería estudiar cómo cambia el canal lateral al variar la formación de la base de datos documental, cambiando el tamaño de chunk, el solapamiento y la estrategia de fragmentación.

Como hemos comentado, el modelo de lenguaje utilizado es importante para el ataque, ya que con modelos grandes se ha encontrado que la efectividad del ataque decae por su alto coste de generación, pero esto genera otra línea de trabajo futura. En este caso se ha concluido esto por una necesidad de agilizar las pruebas, pero es verdad que no se han investigado a fondo los modelos de lenguaje y es posible que, modificando configuraciones de estos como la temperatura, que los hace menos aleatorios cuanto más baja es, o el top-p, que pone un umbral mínimo de probabilidad para la siguiente palabra generada, se consiga que los modelos más grandes se estabilicen, pudiendo observar de forma consistente el ataque en ellos. Además de estas configuraciones, también sería interesante probar la modificación de parámetros como max tokens y min tokens para acotar el tamaño de respuesta y medir de forma sistemática cómo afectan al canal lateral. Esto permitiría distinguir mejor qué parte del tiempo procede de la recuperación y qué parte procede de la generación, y nos ayudaría a comprender si en sistemas con algunas configuraciones específicas estas auditorías son más útiles que en otros.

En cuanto al pipeline de un sistema RAG, la última configuración que genera una línea de trabajo es la aplicación de la caché. En este trabajo se observó que determinados mecanismos de caché pueden modificar el canal lateral, porque el diseño de las pruebas hace que algunas consultas tengan más probabilidad de producir aciertos. Sin embargo, no se ha estudiado la forma óptima de construir unas pruebas para que los resultados obtenidos muestren la representación real de los datos en la base atacada. Está claro que estas pruebas se deben hacer generando consultas diferentes y bien separadas unas de otras, pero sería interesante investigar un sistema que mida si el conjunto de consultas generado podría generar muchos aciertos o no, observando la similitud entre ellas y dando al usuario una forma de saber si sus pruebas son fiables en sistemas con caché. Junto con esto, sería interesante realizar un sistema de generación de consultas que ya tenga esto en cuenta, implementando el uso de agentes de inteligencia artificial y automatizando aún más el proceso de auditorías. Con estos agentes se podrían diseñar bancos de consultas más especializados por dominio, realizando consultas más eficaces y mejorando la efectividad del ataque.

Otra línea de trabajo futura especialmente interesante sería estudiar qué defensas hacen que los sistemas no filtren información por este tipo de canal. Está claro que una defensa sencilla de aplicar sería introducir un retraso en la recuperación de documentos que sea aleatorio, pero esto puede interferir en la usabilidad del sistema, haciéndolo más lento. Por esto es interesante estudiar distintos tipos de defensas y entender cuáles reducen de mejor manera el ataque sin afectar al usuario.

En cuanto a la herramienta, sus líneas futuras dependen directamente de la investigación, ya que la aplicación se ha diseñado para automatizar el ataque estudiado, por lo que cualquier mejora experimental puede acabar reflejándose en nuevas funciones. Si en futuras pruebas se descubre que cierto tipo de corpus, configuración de ingesta de datos, base vectorial o

configuración de caché produce señales más claras modificando la prueba de manera determinada, la herramienta podría incorporar perfiles de auditoría específicos para esos casos. Actualmente no podemos estudiar estas mejoras al no conocer cómo evolucionará la investigación, pero sí podemos hablar de distintos módulos que se podrían implementar en la herramienta actual que serían útiles para el ataque desarrollado.

Una mejora clara de la herramienta sería añadir un módulo específico para detectar patrones compatibles con caché. Si el sistema externo utiliza algún tipo de caché, la herramienta puede mostrar resultados variables dependiendo del diseño de la prueba, y el auditor puede no tener conocimiento sobre la implementación de este sistema de caché. Por esto, sería útil diseñar una prueba que lance ciertas consultas controladas en varias rondas y observe si las repeticiones o consultas semánticamente cercanas producen caídas bruscas de latencia. Si aparece una reducción sistemática entre la primera y la segunda ronda, la herramienta podría mostrar una advertencia indicando que el comportamiento es compatible con algún mecanismo de caché, lo que permitiría interpretar mejor los resultados de la auditoría.

Otra línea de trabajo futura sobre la herramienta sería mejorar la inferencia de términos, ya que la versión actual permite clasificar términos como probablemente presentes, probablemente ausentes o no concluyentes, pero todavía depende mucho de la estabilidad del sistema atacado. En el futuro se podría introducir la utilización de modelos de clasificación que interpreten los tiempos resultantes de las consultas enviadas de la prueba y clasifiquen los términos a partir de los datos de la prueba de auditoría. El sistema actual aplica un sistema de fronteras con cálculos matemáticos clásicos, pero en este tipo de tareas de clasificación los modelos predictivos suelen reflejar buenos resultados, por lo que sería una gran mejora para la inferencia aplicarlos.

Por último, una mejora útil para ampliar la compatibilidad con distintos tipos de API sería mejorar la configuración de los targets. En este momento, la herramienta actual trabaja con targets HTTP con llamadas tipo POST, ya que son las más utilizadas para este tipo de sistemas, pero se podrían añadir configuraciones de otro tipo de llamadas, como el método GET. También se podría mejorar el soporte para autenticación avanzada, certificados, proxies corporativos o límites de tasa, lo que haría que la herramienta fuera aplicable a más entornos reales.

El proyecto se ha detenido en el punto en el que el ataque ya había sido demostrado y se había construido una herramienta capaz de automatizarlo, pero el espacio de investigación sigue siendo amplio, ya que quedan por estudiar más corpus, más modelos, más bases vectoriales, más arquitecturas RAG, entre otras cosas. Por tanto, este TFG puede entenderse como una primera base experimental y técnica sobre la que continuar investigando la seguridad temporal de sistemas RAG.

9. BIBLIOGRAFÍA

[1] W. X. Zhao *et al.*, "A survey of large language models," *arXiv preprint arXiv:2303.18223*, 2023. [En línea]. Disponible en: https://arxiv.org/abs/2303.18223

[2] P. Lewis *et al.*, "Retrieval-augmented generation for knowledge-intensive NLP tasks," en *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 33, 2020, pp. 9459-9474. [En línea]. Disponible en: https://arxiv.org/abs/2005.11401

[3] Y. Gao *et al.*, "Retrieval-augmented generation for large language models: A survey," *arXiv preprint arXiv:2312.10997*, 2023. [En línea]. Disponible en: https://arxiv.org/abs/2312.10997

[4] P. C. Kocher, "Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems," en *Advances in Cryptology - CRYPTO '96*, N. Koblitz, ed. Berlín, Heidelberg: Springer, 1996, pp. 104-113. [En línea]. Disponible en: https://link.springer.com/chapter/10.1007/3-540-68697-5_9

[5] Q. Ge, Y. Yarom, D. Cock, y G. Heiser, "A survey of microarchitectural timing attacks and countermeasures on contemporary hardware," *Journal of Cryptographic Engineering*, vol. 8, no. 1, pp. 1-27, abr. 2018. [En línea]. Disponible en: https://eprint.iacr.org/2016/613

[6] Y. A. Malkov y D. A. Yashunin, "Efficient and robust approximate nearest neighbor search using Hierarchical Navigable Small World graphs," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 42, no. 4, pp. 824-836, abr. 2020. [En línea]. Disponible en: https://doi.org/10.1109/TPAMI.2018.2889473

[7] R. Shokri, M. Stronati, C. Song, y V. Shmatikov, "Membership inference attacks against machine learning models," en *Proc. IEEE Symposium on Security and Privacy (SP)*, San Jose, CA, EE.UU., 2017, pp. 3-18. [En línea]. Disponible en: https://doi.ieeecomputersociety.org/10.1109/SP.2017.41

[8] M. Anderson, G. Amit, y A. Goldsteen, "Is my data in your retrieval database? Membership inference attacks against retrieval augmented generation," *arXiv preprint arXiv:2405.20446*, 2024. [En línea]. Disponible en: https://arxiv.org/abs/2405.20446

[9] H. Li, J. He, G. Wang, D. Feng, Z. Li, y M. Zhang, "BudgetLeak: Membership inference attacks on RAG systems via the generation budget side channel," *arXiv preprint arXiv:2511.12043*, 2025. [En línea]. Disponible en: https://arxiv.org/abs/2511.12043

[10] N. Carlini y M. Nasr, "Remote timing attacks on efficient language model inference," *arXiv preprint arXiv:2410.17175*, 2024. [En línea]. Disponible en: https://arxiv.org/abs/2410.17175

[11] G. McDonald y J. Bar Or, "Whisper Leak: A side-channel attack on large language models," *arXiv preprint arXiv:2511.03675*, 2025. [En línea]. Disponible en: https://arxiv.org/abs/2511.03675

[12] W. Kwon *et al.*, "Efficient memory management for large language model serving with PagedAttention," en *Proc. 29th ACM Symposium on Operating Systems Principles*

(SOSP), Coblenza, Alemania, 2023, pp. 611-626. [En línea]. Disponible en: https://doi.org/10.1145/3600006.3613165

[13] F. Bang, "GPTCache: An open-source semantic cache for LLM applications enabling faster answers and cost savings," en *Proc. 3rd Workshop for Natural Language Processing Open Source Software (NLP-OSS)*, Singapur, 2023, pp. 212-218. [En línea]. Disponible en: https://aclanthology.org/2023.nlposs-1.24/

[14] X. Zheng *et al.*, "InputSnatch: Stealing input in LLM services via timing side-channel attacks," *arXiv preprint arXiv:2411.18191*, 2024. [En línea]. Disponible en: https://arxiv.org/abs/2411.18191

[15] G. Wu *et al.*, "I know what you asked: Prompt leakage via KV-cache sharing in multi-tenant LLM serving," en *Proc. Network and Distributed System Security Symposium (NDSS)*, San Diego, CA, EE.UU., 2025. [En línea]. Disponible en: https://www.ndss-symposium.org/ndss-paper/i-know-what-you-asked-prompt-leakage-via-kv-cache-sharing-in-multi-tenant-llm-serving/

[16] A. Vaswani *et al.*, "Attention is all you need," en *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 30, 2017, pp. 5998-6008. [En línea]. Disponible en: https://arxiv.org/abs/1706.03762

[17] E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.3," *Internet Engineering Task Force (IETF)*, RFC 8446, ago. 2018. [En línea]. Disponible en: https://www.rfc-editor.org/info/rfc8446/

[18] Chroma, "Configure collections: HNSW index configuration," *Chroma Documentation*, consultado: ago. 2026. [En línea]. Disponible en: https://docs.trychroma.com/docs/collections/configure

[19] ISO/IEC/IEEE, "Systems and Software Engineering - Life Cycle Processes - Requirements Engineering," *ISO/IEC/IEEE Standard 29148:2018*, nov. 2018. [En línea]. Disponible en: https://doi.org/10.1109/IEEESTD.2018.8559686

[20] A. Cockburn, *Writing Effective Use Cases*. Boston, MA, EE.UU.: Addison-Wesley, 2001.

[21] R. Fielding, M. Nottingham, y J. Reschke, "HTTP Semantics," *RFC 9110*, *Internet Engineering Task Force (IETF)*, jun. 2022. [En línea]. Disponible en: https://www.rfc-editor.org/info/rfc9110

[22] B. L. Welch, "The Generalization of Student's Problem When Several Different Population Variances Are Involved," *Biometrika*, vol. 34, no. 1/2, pp. 28-35, 1947.

[23] T. Bray, "The JavaScript Object Notation (JSON) Data Interchange Format," *RFC 8259*, *Internet Engineering Task Force (IETF)*, dic. 2017. [En línea]. Disponible en: https://www.rfc-editor.org/info/rfc8259

[24] Y. Shafranovich, "Common Format and MIME Type for Comma-Separated Values (CSV) Files," *RFC 4180*, *Internet Engineering Task Force (IETF)*, oct. 2005. [En línea]. Disponible en: https://www.rfc-editor.org/info/rfc4180

[25] C. R. Harris *et al.*, "Array programming with NumPy," *Nature*, vol. 585, no. 7825, pp. 357-362, sep. 2020. [En línea]. Disponible en: https://doi.org/10.1038/s41586-020-2649-2

[26] P. Virtanen *et al.*, "SciPy 1.0: Fundamental algorithms for scientific computing in Python," *Nature Methods*, vol. 17, no. 3, pp. 261-272, mar. 2020. [En línea]. Disponible en: https://doi.org/10.1038/s41592-019-0686-2

[27] S. Verde Llorente, "Auditoria_RAG_Timing," *GitHub repository*, 2026. [En línea]. Disponible en: https://github.com/100495899/TFG

[28] N. Reimers y I. Gurevych, "Sentence-BERT: Sentence embeddings using Siamese BERT-networks," en *Proc. 2019 Conference on Empirical Methods in Natural Language Processing (EMNLP-IJCNLP)*, Hong Kong, China, 2019, pp. 3982-3992. [En línea]. Disponible en: https://arxiv.org/abs/1908.10084

[29] J. W. Rae, A. Potapenko, S. M. Jayakumar, y T. P. Lillicrap, "Compressive Transformers for long-range sequence modelling," en *Proc. International Conference on Learning Representations (ICLR)*, 2020. [En línea]. Disponible en: https://arxiv.org/abs/1911.05507

[30] S. Bird, E. Klein, y E. Loper, *Natural Language Processing with Python*. Sebastopol, CA, EE.UU.: O'Reilly Media, 2009.

[31] W. Wang *et al.*, "MiniLM: Deep self-attention distillation for task-agnostic compression of pre-trained transformers," en *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 33, 2020, pp. 5776-5788. [En línea]. Disponible en: https://arxiv.org/abs/2002.10957

[32] L. McInnes, J. Healy, y J. Melville, "UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction," *arXiv preprint arXiv:1802.03426*, 2018. [En línea]. Disponible en: https://arxiv.org/abs/1802.03426

[33] A. Yang *et al.*, "Qwen2.5 technical report," *arXiv preprint arXiv:2412.15115*, 2024. [En línea]. Disponible en: https://arxiv.org/abs/2412.15115

[34] L. B. Allal *et al.*, "SmolLM2: When Smol goes big – Data-centric training of a small language model," *arXiv preprint arXiv:2502.02737*, 2025. [En línea]. Disponible en: https://arxiv.org/abs/2502.02737

[35] OWASP Foundation, "OWASP Top 10 for Large Language Model Applications 2025," *OWASP Gen AI Security Project*, 2025. [En línea]. Disponible en: https://owasp.org/www-project-top-10-for-large-language-model-applications/

[36] National Institute of Standards and Technology (NIST), "Artificial Intelligence Risk Management Framework (AI RMF 1.0)," *NIST AI 100-1*, Gaithersburg, MD, EE. UU., ene. 2023. [En línea]. Disponible en: https://nvlpubs.nist.gov/nistpubs/ai/NIST.AI.100-1.pdf

[37] OpenAI, "text-embedding-3-small Model," *OpenAI API Documentation*, consultado: ago. 2026. [En línea]. Disponible en: https://developers.openai.com/api/docs/models/text-embedding-3-small

ANEXO A. VISUALIZACIÓN DE LA INTERFAZ DE USUARIO

En este anexo se muestran las principales pantallas de la herramienta desarrollada. Estas capturas complementan el diseño descrito en el apartado 4.6 y la implementación explicada en el apartado 5.7, permitiendo observar cómo se materializan visualmente los flujos de autenticación, gestión de targets, carga de datasets, creación de auditorías, consulta de resultados e inferencia de términos.

A.1. PANTALLA DE AUTENTICACIÓN

La Fig. 35 muestra la pantalla inicial de autenticación de la herramienta, la cual permite que el auditor introduzca sus credenciales antes de acceder a las pantallas internas. Esta autenticación permite proteger información sensible almacenada en la aplicación, como endpoints, headers, datasets cargados y resultados de auditorías anteriores.

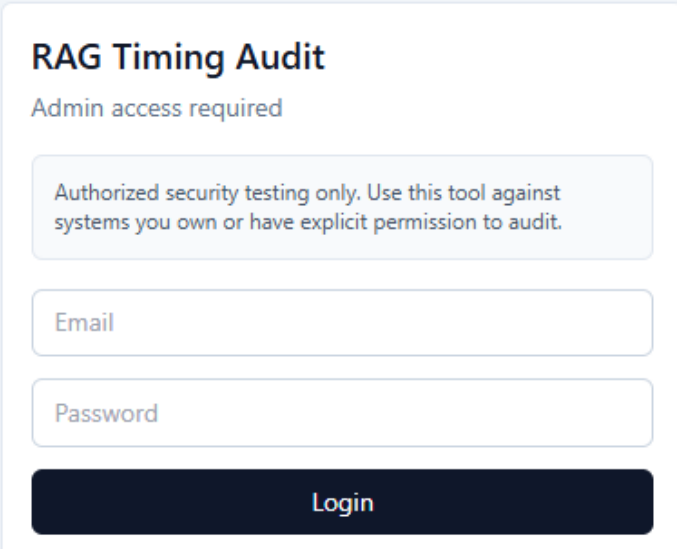


Fig. 35. Pantalla de autenticación

A.2. PANEL PRINCIPAL DE AUDITORÍAS

La Fig. 36 muestra el panel principal de auditorías, que es una de las pantallas centrales de la aplicación. Desde esta vista, el auditor puede consultar las auditorías creadas, revisar su estado, observar el progreso de ejecución y acceder a los resultados de aquellas pruebas que ya han finalizado.

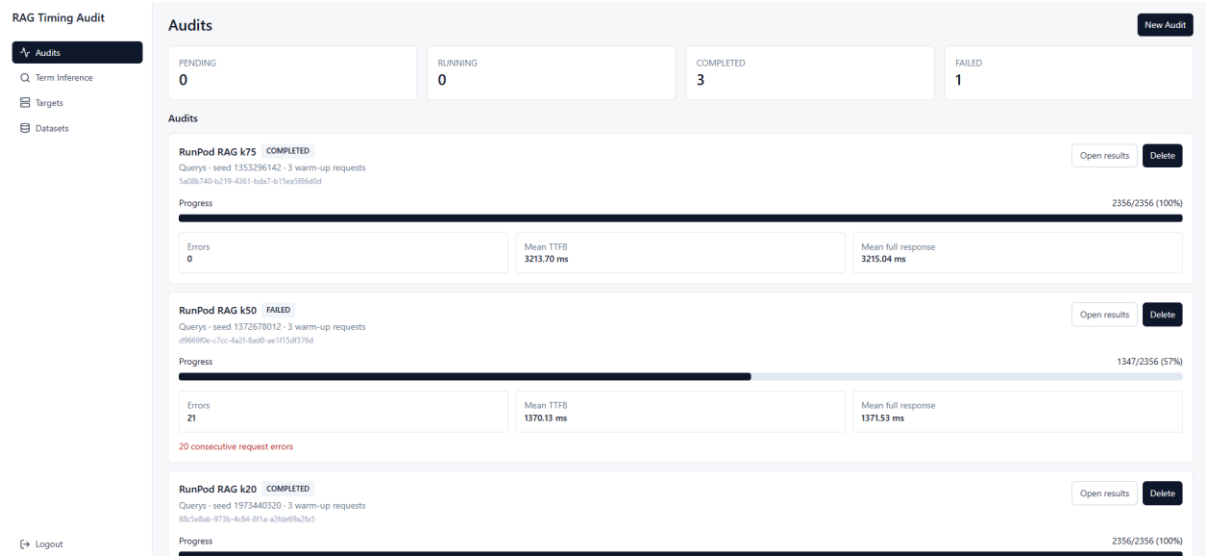


Fig. 36. Panel principal de auditorías.

A.3. GESTIÓN DE TARGETS

La Fig. 37 muestra la pantalla de gestión de targets. Desde esta vista, el auditor puede registrar un nuevo sistema RAG externo indicando los datos necesarios para comunicarse con él, como la URL del endpoint, los headers, el timeout, la configuración TLS y la plantilla del body. Esta pantalla implementa el flujo descrito en el apartado 4.3, donde se explicó que el target representa el sistema externo que se quiere auditar mediante peticiones HTTP. En esta imagen también se muestra la funcionalidad de prueba de conexión de un target, la cual se ejecuta mediante el botón de prueba del target y muestra un mensaje de confirmación o error debajo de configured targets. Esta acción permite comprobar si el sistema externo responde correctamente antes de utilizarlo en una auditoría.

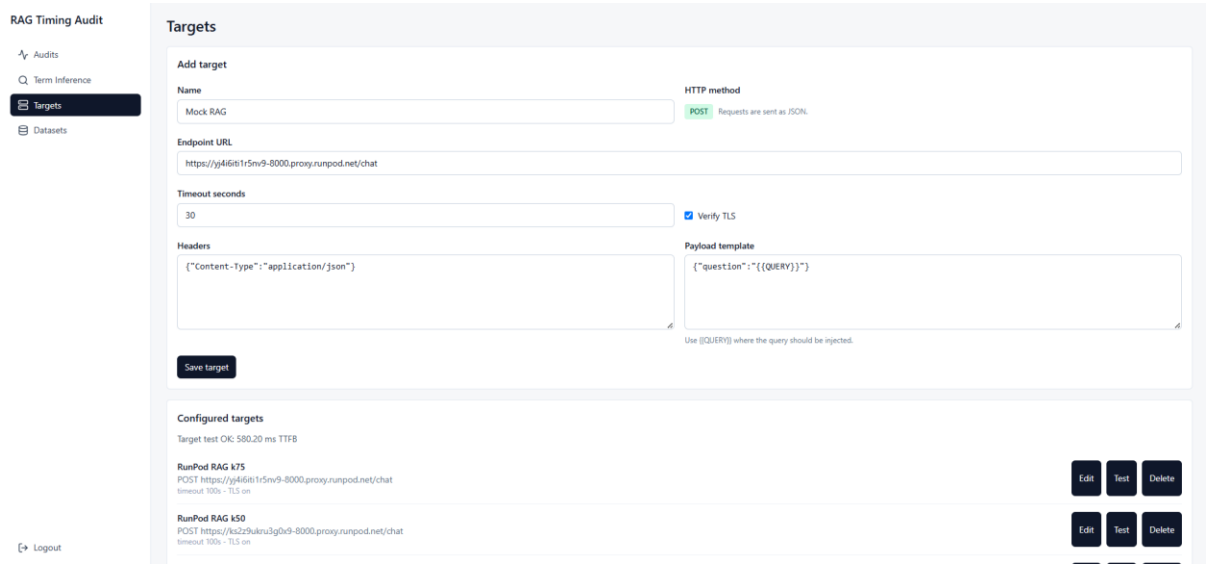


Fig. 37. Pantalla de gestión y creación de targets.

La Fig. 38 muestra el formulario de edición de un target ya registrado. Aunque visualmente es similar al formulario de creación, su finalidad es distinta, ya que permite modificar la configuración de un sistema RAG externo previamente almacenado. Esta opción resulta útil cuando cambia el endpoint, se actualizan headers de autenticación, se modifica el timeout o se necesita ajustar la plantilla del body sin tener que crear un nuevo target desde cero.

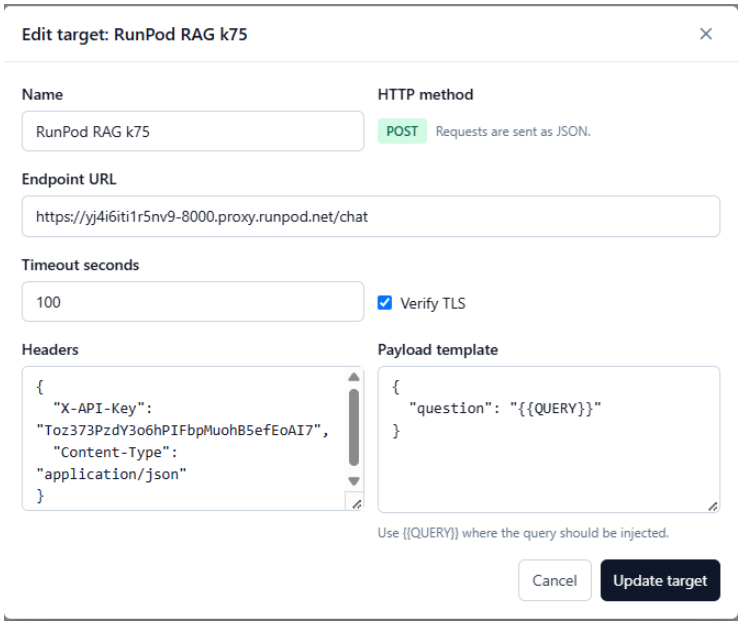


Fig. 38. Formulario de edición de un target existente.

A.4. CARGA Y VALIDACIÓN DE DATASETS

La Fig. 39 muestra la pantalla de carga de datasets. Desde esta vista, el auditor puede subir un archivo JSON con consultas clasificadas por frecuencia y longitud. Esta estructura es necesaria porque la herramienta utiliza esas etiquetas para comparar grupos durante el análisis estadístico, tal como se explicó en los apartados de diseño funcional y medición temporal.

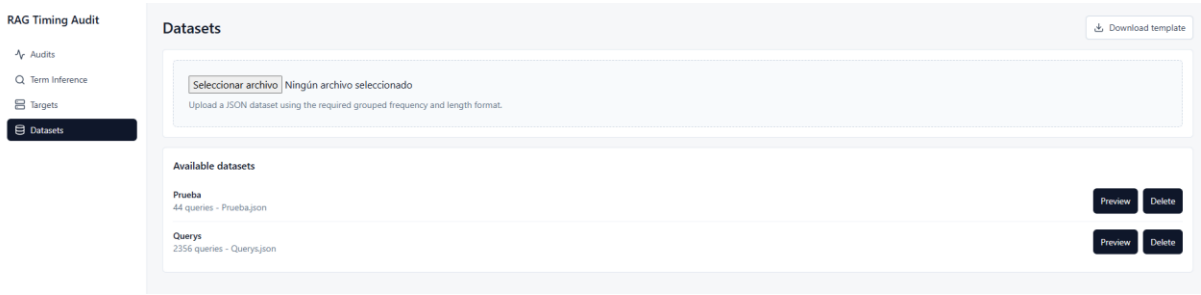


Fig. 39. Pantalla de carga de datasets.

La Fig. 40 muestra la vista previa del dataset una vez cargado y validado. En esta pantalla se presentan estadísticas básicas del archivo, como el número total de consultas y su distribución por grupos de frecuencia y longitud. Esta funcionalidad ayuda al auditor a comprobar que el dataset ha sido interpretado correctamente antes de usarlo en una auditoría, reduciendo el riesgo de ejecutar pruebas con archivos incompletos o mal estructurados.

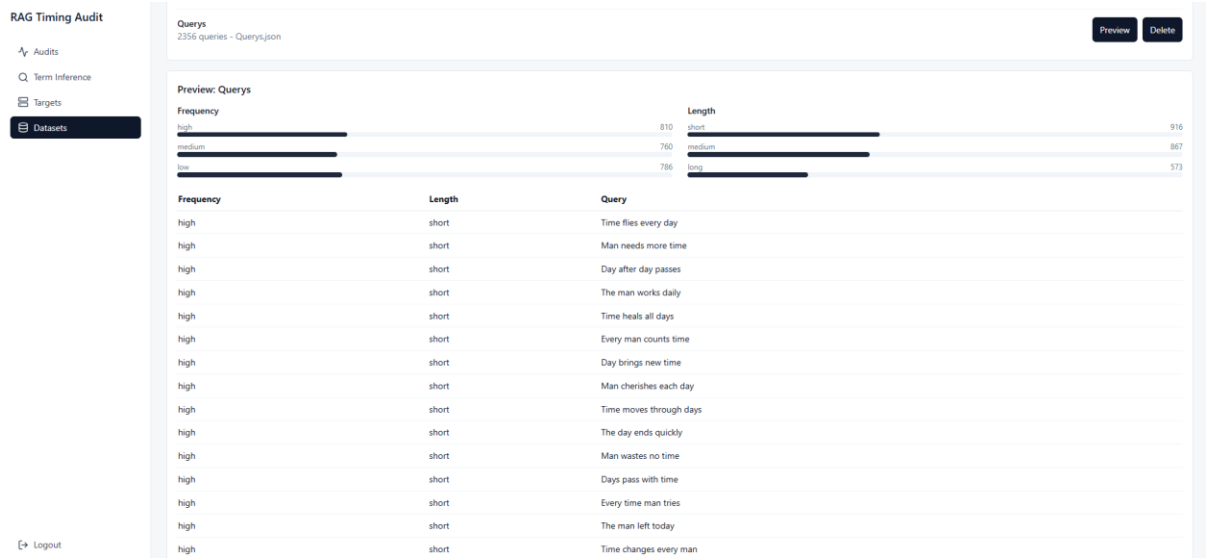


Fig. 40. Vista previa y estadísticas de un dataset cargado.

A.5. CREACIÓN DE UNA AUDITORÍA

La Fig. 41 muestra el formulario de creación de auditorías. En esta pantalla, el auditor selecciona un target, un dataset y los parámetros de ejecución necesarios, como el número de peticiones de calentamiento y, opcionalmente, la semilla aleatoria. Este formulario representa la entrada al flujo de auditoría descrito en el apartado 4.3.1, donde se explicó que la prueba queda registrada en la base de datos y posteriormente es encolada para su ejecución por el worker.

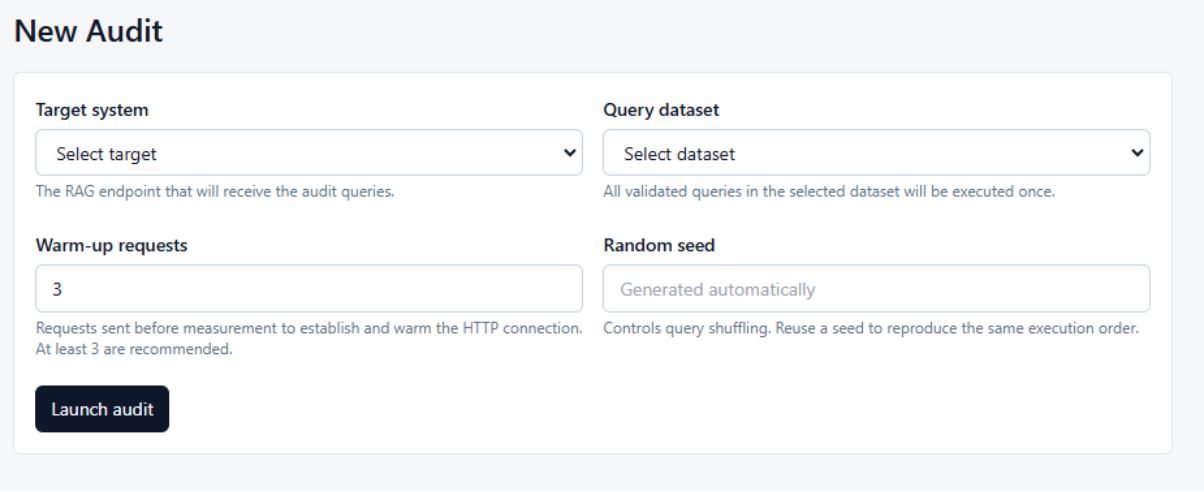


Fig. 41. Formulario de creación de una auditoría.

A.6. RESULTADOS DE AUDITORÍA

La Fig. 42 muestra la vista general de resultados de una auditoría. En esta pantalla se presentan las métricas agregadas y las gráficas principales calculadas a partir de las mediciones guardadas. Como se explicó en los apartados 4.4 y 5.6, estos resultados permiten comparar grupos de consultas según su frecuencia y longitud, utilizando métricas como la media, la mediana, los percentiles y el p-value.

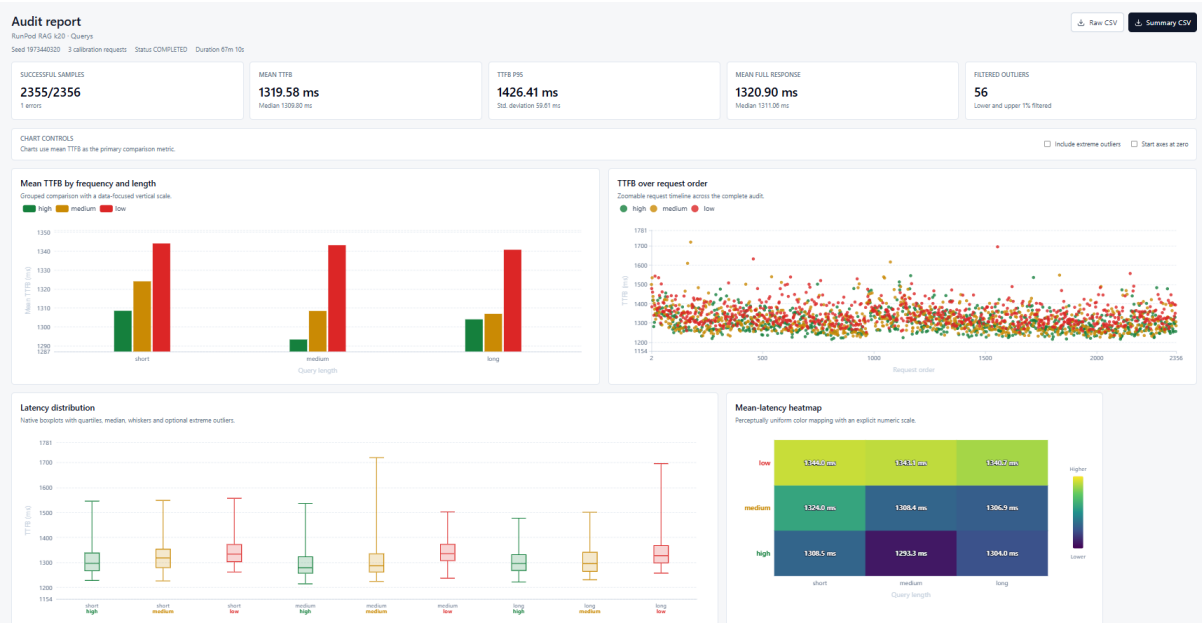
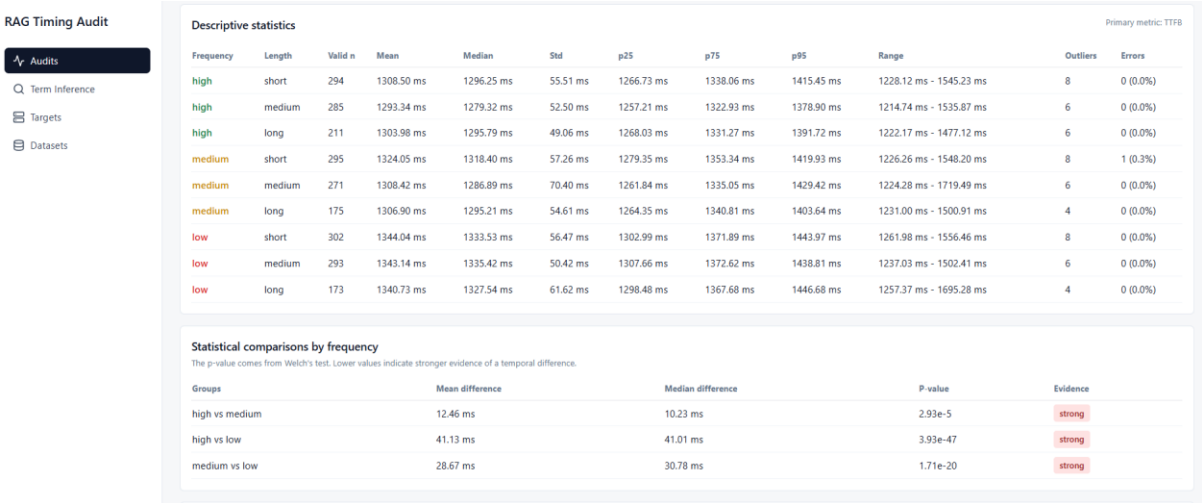


Fig. 42. Vista general de resultados de una auditoría.

La Fig. 43 muestra las comparaciones estadísticas calculadas por la herramienta. Esta vista permite analizar si las diferencias temporales entre grupos presentan suficiente peso estadístico o si podrían deberse al ruido normal del sistema auditado. Esta parte de la interfaz es importante porque evita que el auditor tenga que interpretar únicamente valores individuales de latencia.

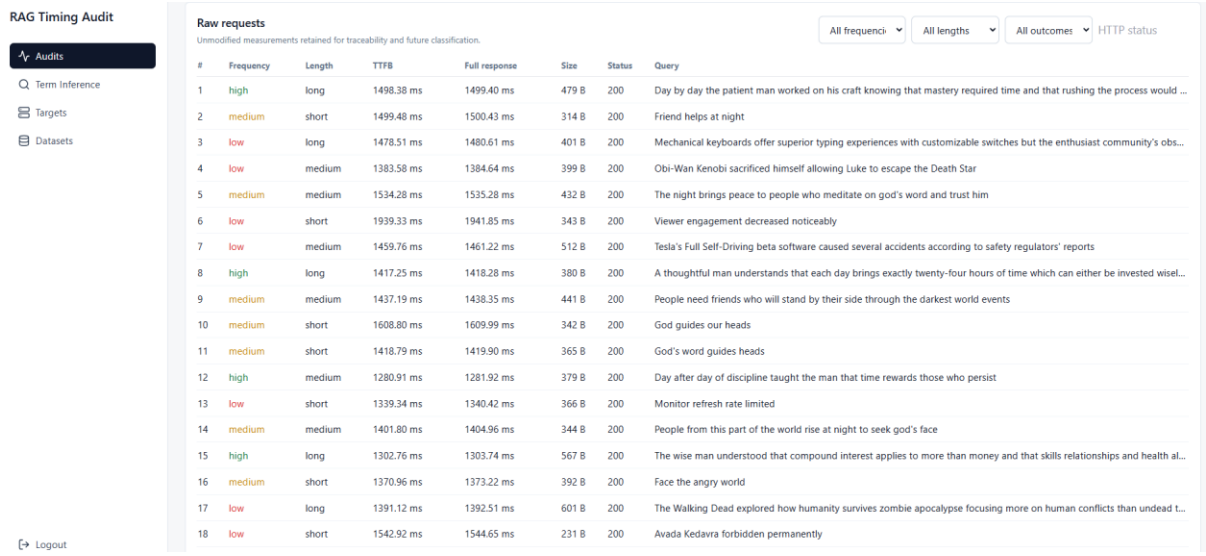


Frequency	Length	Valid n	Mean	Median	Std	p25	p75	p95	Range	Outliers	Errors
high	short	294	1308.50 ms	1296.25 ms	55.51 ms	1266.73 ms	1338.06 ms	1415.45 ms	1228.12 ms - 1545.23 ms	8	0 (0.0%)
high	medium	285	1293.34 ms	1279.32 ms	52.50 ms	1257.21 ms	1322.93 ms	1378.90 ms	1214.74 ms - 1535.87 ms	6	0 (0.0%)
high	long	211	1303.98 ms	1295.79 ms	49.06 ms	1268.03 ms	1331.27 ms	1391.72 ms	1222.17 ms - 1477.12 ms	6	0 (0.0%)
medium	short	295	1324.05 ms	1318.40 ms	57.26 ms	1279.35 ms	1353.34 ms	1419.93 ms	1226.26 ms - 1548.20 ms	8	1 (0.3%)
medium	medium	271	1308.42 ms	1286.89 ms	70.40 ms	1261.84 ms	1335.05 ms	1429.42 ms	1224.28 ms - 1719.49 ms	6	0 (0.0%)
medium	long	175	1306.90 ms	1295.21 ms	54.61 ms	1264.35 ms	1340.81 ms	1403.64 ms	1231.00 ms - 1500.91 ms	4	0 (0.0%)
low	short	302	1344.04 ms	1333.53 ms	56.47 ms	1302.99 ms	1371.89 ms	1443.97 ms	1261.98 ms - 1556.46 ms	8	0 (0.0%)
low	medium	293	1343.14 ms	1335.42 ms	50.42 ms	1307.66 ms	1372.62 ms	1438.81 ms	1237.03 ms - 1502.41 ms	6	0 (0.0%)
low	long	173	1340.73 ms	1327.54 ms	61.62 ms	1298.48 ms	1367.68 ms	1446.68 ms	1257.37 ms - 1695.28 ms	4	0 (0.0%)

Groups	Mean difference	Median difference	P-value	Evidence
high vs medium	12.46 ms	10.23 ms	2.93e-5	strong
high vs low	41.13 ms	41.01 ms	3.93e-47	strong
medium vs low	28.67 ms	30.78 ms	1.71e-20	strong

Fig. 43. Comparaciones estadísticas por frecuencia en una auditoría.

La Fig. 44 muestra la tabla paginada con filtros de mediciones individuales generadas durante una auditoría. En esta vista se pueden consultar las consultas enviadas, sus etiquetas, los tiempos medidos, el código HTTP, el tamaño de respuesta y los posibles errores. Esta información permite revisar el detalle de la ejecución y mantiene la trazabilidad de los resultados agregados mostrados en las gráficas.



#	Frequency	Length	TTFB	Full response	Size	Status	Query
1	high	long	1498.38 ms	1499.40 ms	479 B	200	Day by day the patient man worked on his craft knowing that mastery required time and that rushing the process would ...
2	medium	short	1499.48 ms	1500.43 ms	314 B	200	Friend helps at night
3	low	long	1478.51 ms	1480.61 ms	401 B	200	Mechanical keyboards offer superior typing experiences with customizable switches but the enthusiast community's obs...
4	low	medium	1383.58 ms	1384.64 ms	399 B	200	Obi-Wan Kenobi sacrificed himself allowing Luke to escape the Death Star
5	medium	medium	1534.28 ms	1535.28 ms	432 B	200	The night brings peace to people who meditate on god's word and trust him
6	low	short	1939.33 ms	1941.85 ms	343 B	200	Viewer engagement decreased noticeably
7	low	medium	1459.76 ms	1461.22 ms	512 B	200	Tesla's Full Self-Driving beta software caused several accidents according to safety regulators' reports
8	high	long	1417.25 ms	1418.28 ms	380 B	200	A thoughtful man understands that each day brings exactly twenty-four hours of time which can either be invested wisel...
9	medium	medium	1437.19 ms	1438.35 ms	441 B	200	People need friends who will stand by their side through the darkest world events
10	medium	short	1608.80 ms	1609.99 ms	342 B	200	God guides our heads
11	medium	short	1418.79 ms	1419.90 ms	365 B	200	God's word guides heads
12	high	medium	1280.91 ms	1281.92 ms	379 B	200	Day after day of discipline taught the man that time rewards those who persist
13	low	short	1339.34 ms	1340.42 ms	366 B	200	Monitor refresh rate limited
14	medium	medium	1401.80 ms	1404.96 ms	344 B	200	People from this part of the world rise at night to seek god's face
15	high	long	1302.76 ms	1303.74 ms	567 B	200	The wise man understood that compound interest applies to more than money and that skills relationships and health al...
16	medium	short	1370.96 ms	1373.22 ms	392 B	200	Face the angry world
17	low	long	1391.12 ms	1392.51 ms	601 B	200	The Walking Dead explored how humanity survives zombie apocalypse focusing more on human conflicts than undead t...
18	low	short	1542.92 ms	1544.65 ms	231 B	200	Avada Kedavra forbidden permanently

Fig. 44. Tabla de mediciones individuales de una auditoría.

A.7. PANEL DE INFERENCIA DE TÉRMINOS

La Fig. 45 muestra el panel principal de inferencias de términos. Esta pantalla permite consultar las pruebas de inferencia creadas, revisar su estado, acceder a sus resultados y crear nuevas ejecuciones. Como se explicó en el diseño funcional de la herramienta, la inferencia de términos actúa como un segundo flujo principal posterior a la auditoría inicial, ya que necesita una referencia temporal previa para poder comparar nuevos términos.

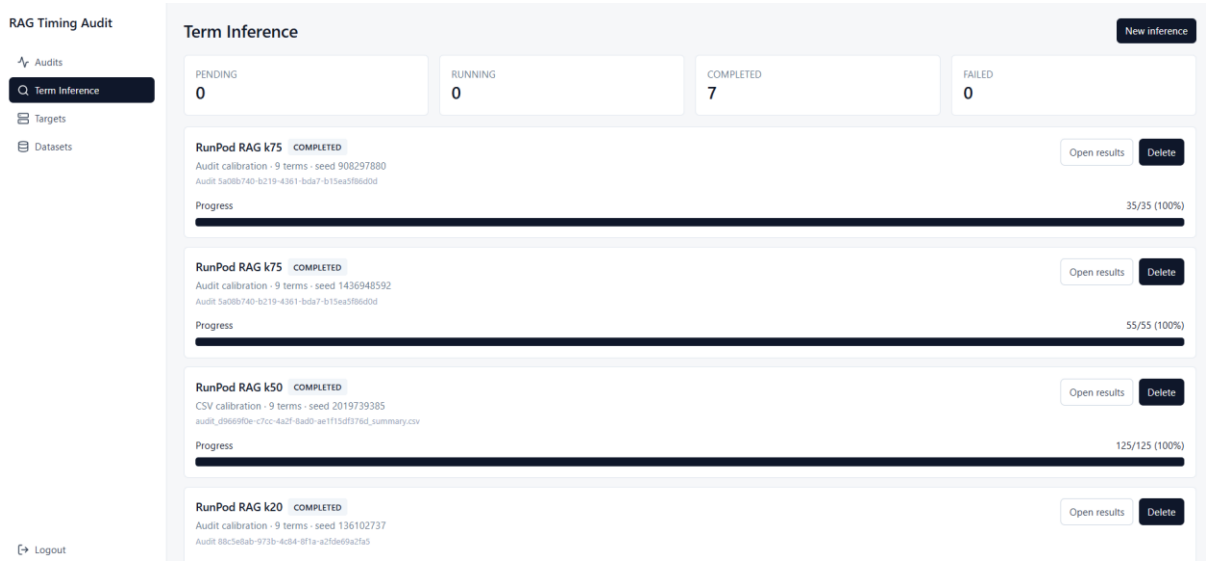


Fig. 45. Panel principal de inferencias de términos.

A.8. CREACIÓN DE INFERENCIA DE TÉRMINOS

La Fig. 46 muestra el formulario de creación de una inferencia de términos usando una auditoría previa como fuente de calibración. En este caso, el auditor selecciona el target, la auditoría de referencia y los términos o consultas que quiere evaluar. Esta configuración permite comparar nuevas mediciones con el perfil temporal obtenido previamente en el mismo sistema. Además, en esta imagen también se muestra el modo de términos individuales en la inferencia. En este modo, el auditor introduce los términos que quiere estudiar y la herramienta genera consultas asociadas para enviarlas al sistema externo. Esta opción está pensada para realizar pruebas

rápidas centradas en términos concretos sin tener que construir manualmente todas las consultas.

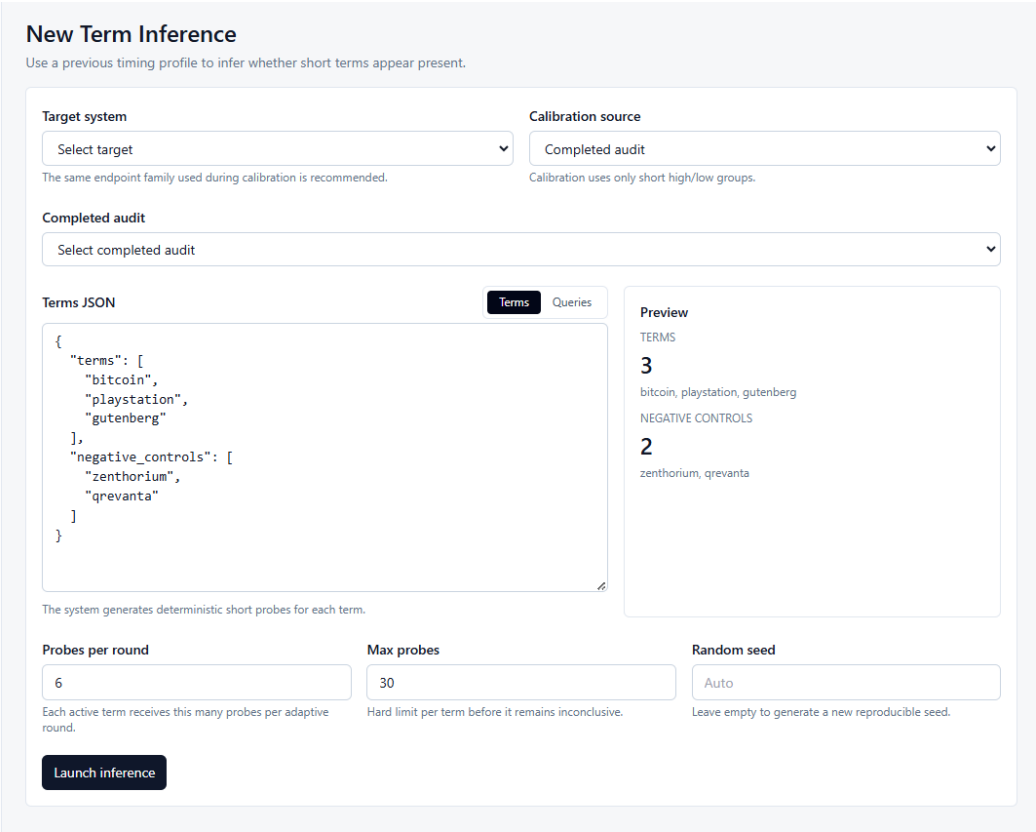


Fig. 46. Formulario de creación de inferencia de términos con auditoría previa y términos individuales.

La Fig. 47 muestra la opción de crear una inferencia de términos a partir de un archivo CSV externo. Esta alternativa permite cargar resultados de una auditoría previa no necesariamente almacenada en la aplicación, siempre que el archivo contenga las columnas y valores necesarios para construir el perfil de calibración. En esta imagen también se muestra el modo de queries del sistema de inferencia que ofrece mayor control al auditor, ya que permite definir directamente las frases que se enviarán al target durante la inferencia. Su uso es útil cuando se quiere adaptar la prueba a un contexto específico o evitar consultas genéricas que no representen bien el caso de estudio.

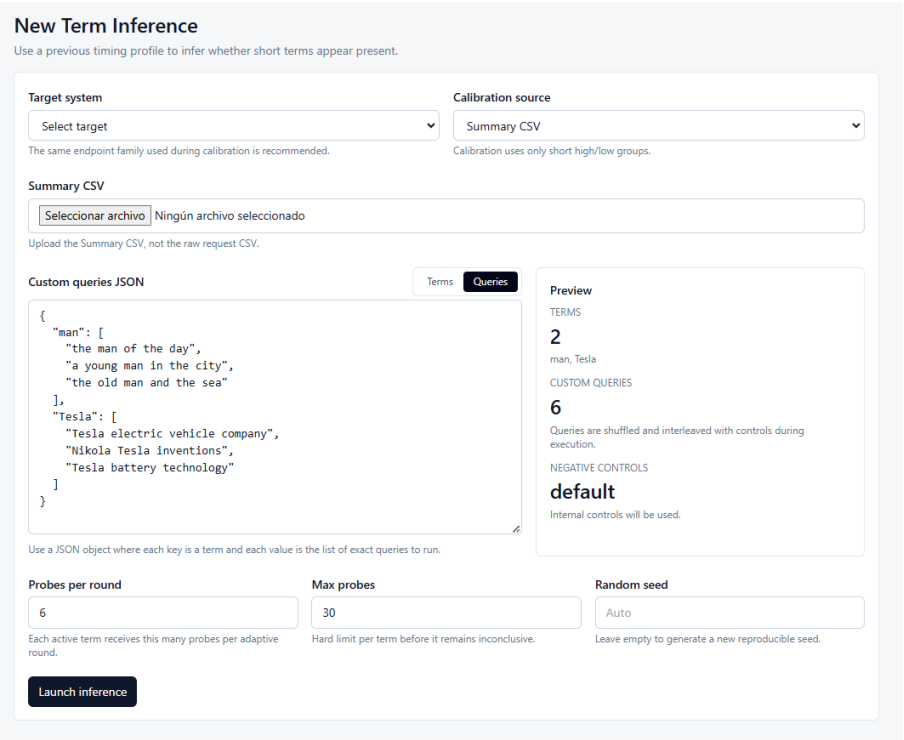


Fig. 47. Formulario de creación de inferencia de términos con calibración mediante CSV y modo de consultas.

A.9. RESULTADOS DE INFERENCIA DE TÉRMINOS

La Fig. 48 muestra la gráfica de clasificación de términos frente a las fronteras temporales de calibración. En ella se representan los valores de referencia de alta y baja presencia, la zona gris y la media observada para cada término evaluado. Esta visualización permite entender de forma directa por qué un término se ha clasificado de una manera concreta.

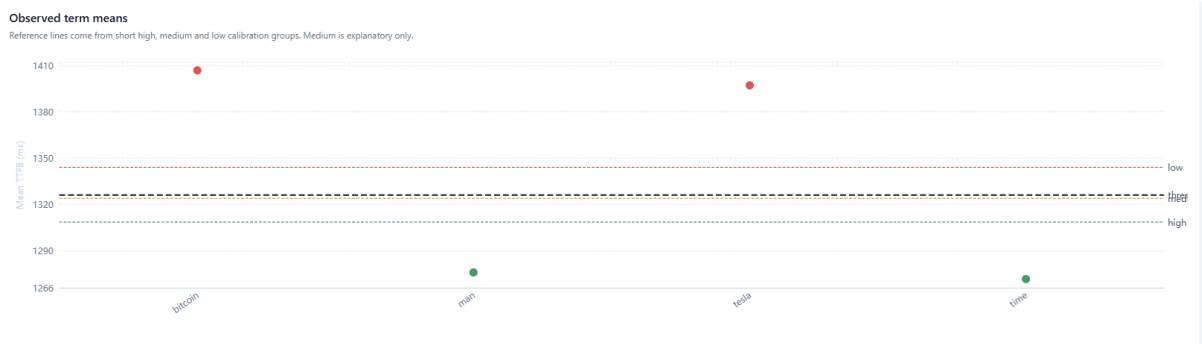


Fig. 48. Clasificación de términos frente a las fronteras de calibración.

La Fig. 49 muestra la evolución de la inferencia a lo largo de sus rondas de ejecución. Esta vista permite observar cuándo se clasifica cada término y si alguno necesita continuar a rondas posteriores por encontrarse cerca de la zona gris. Esta información resulta útil para entender la estabilidad de la clasificación y la sensibilidad de la prueba frente al ruido.

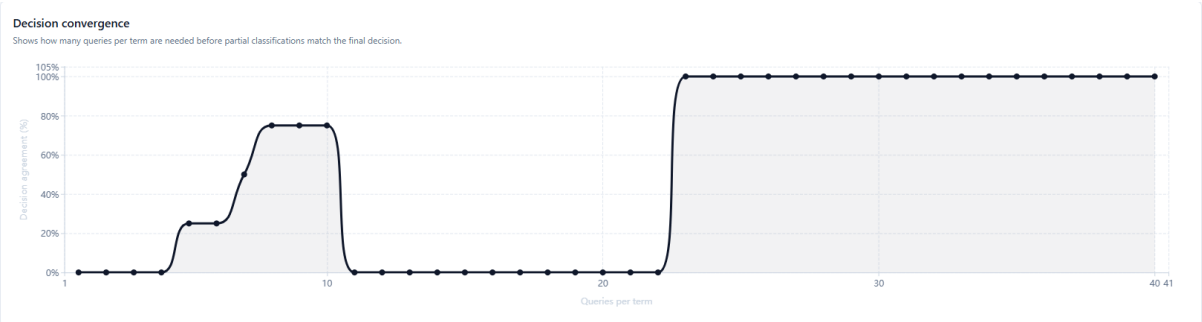


Fig. 49. Evolución de la clasificación por rondas durante una inferencia.

La Fig. 50 muestra la tabla de resultados de una inferencia de términos. En esta tabla se recogen valores como la clasificación final, la media observada del Time-to-First-Byte, la desviación típica, el número de mediciones válidas, la distancia respecto al umbral y la referencia temporal más cercana. Esta información complementa las gráficas y permite revisar los resultados de forma más detallada.

Term decisions

Term	Type	Classification	Mean TTFB	Std	Valid/total	Distance	Closest ref	Errors
bitcoin	term	likely_absent	1406.79 ms	50.87 ms	10/10	80.51 ms	low	0
man	term	likely_present	1275.92 ms	34.36 ms	10/10	-50.35 ms	high	0
tesla	term	likely_absent	1397.09 ms	44.06 ms	40/40	70.82 ms	low	0
time	term	likely_present	1271.63 ms	32.46 ms	10/10	-54.64 ms	high	0
malvexor	control	likely_absent	1437.66 ms	17.72 ms	8/8	111.39 ms	low	0
norithium	control	likely_absent	1435.56 ms	21.73 ms	8/8	109.29 ms	low	0
qrevanta	control	likely_absent	1431.01 ms	22.68 ms	8/8	104.74 ms	low	0
veltraxis	control	likely_absent	1423.96 ms	18.72 ms	8/8	97.69 ms	low	0
zenthorium	control	likely_absent	1424.49 ms	20.90 ms	13/13	98.22 ms	low	0

Fig. 50. Tabla de resultados de una inferencia de términos.

La Fig. 51 muestra las mediciones individuales en una tabla paginada con filtros generadas durante una inferencia de términos. Cada registro queda asociado a una consulta concreta, una ronda de ejecución y el término evaluado. Esta información permite revisar el comportamiento de la prueba con mayor detalle y detectar posibles errores o valores anómalos.

Probe timeline

Chronological sequence of term probes and interleaved negative controls stored during this inference.

All types | Term | Round | All outcom

#	Round	Term	Type	TTFB	Full response	Size	Status	Query
1	0	zenthorium	control	1410.50 ms	1650.15 ms	1837 B	200	Information about zenthorium.
2	0	zenthorium	control	1428.60 ms	1679.55 ms	1874 B	200	Context for zenthorium.
3	0	zenthorium	control	1446.70 ms	1708.95 ms	1911 B	200	Meaning of zenthorium.
4	0	zenthorium	control	1464.80 ms	1737.35 ms	1948 B	200	What is zenthorium?
5	0	zenthorium	control	1406.90 ms	1690.75 ms	1985 B	200	Explain zenthorium.
6	1	tesla	term	1339.50 ms	1637.65 ms	2022 B	200	Tesla leads electric future
7	1	bitcoin	term	1329.95 ms	1638.40 ms	2059 B	200	Bitcoin changes everything
8	1	man	term	1323.75 ms	1559.50 ms	2096 B	200	Man lived many days
9	1	bitcoin	term	1331.75 ms	1578.80 ms	2133 B	200	Never sell your bitcoin
10	1	zenthorium	control	1400.40 ms	1657.75 ms	2170 B	200	About zenthorium.
11	1	tesla	term	1328.00 ms	1596.65 ms	2207 B	200	Tesla stock is rising
12	1	time	term	1322.95 ms	1602.90 ms	2244 B	200	The time arrived suddenly
13	1	man	term	1341.25 ms	1632.50 ms	2281 B	200	Man needs more time
14	1	time	term	1318.55 ms	1620.10 ms	2318 B	200	Time shapes the man
15	1	qrevanta	control	1440.90 ms	1669.75 ms	2355 B	200	What is qrevanta?
16	1	man	term	1315.15 ms	1555.30 ms	2392 B	200	The man left today
17	1	tesla	term	1318.40 ms	1568.85 ms	2429 B	200	Tesla makes great cars

Fig. 51. Mediciones individuales registradas durante una inferencia de términos.

A.10. CONFIRMACIÓN DE ACCIONES DESTRUCTIVAS

La Fig. 52 muestra el panel de confirmación que aparece cuando el usuario intenta borrar un elemento de la herramienta. Este componente se utiliza en distintas partes de la aplicación, como targets, datasets, auditorías o inferencias, y su texto cambia dependiendo del tipo de elemento que se vaya a eliminar. Su objetivo es evitar borrados accidentales y reforzar la seguridad de las acciones no reversibles dentro de la interfaz.

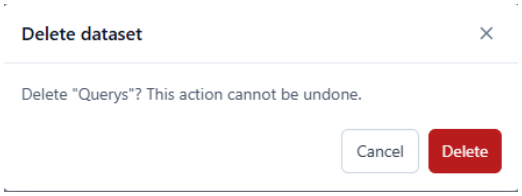


Fig. 52. Panel de confirmación para acciones de eliminación.

ANEXO B. MANUAL DE INSTALACIÓN Y EJECUCIÓN

Este anexo recoge los pasos necesarios para instalar y ejecutar la herramienta Auditoria_RAG_Timing en un entorno local controlado. Su objetivo es complementar la implementación descrita en el capítulo 5 con una guía práctica que permita levantar la aplicación y comprobar que sus servicios principales funcionan correctamente. La herramienta se ejecuta mediante Docker Compose, por lo que no es necesario instalar manualmente las dependencias internas del backend, frontend, base de datos o sistema de colas. El entorno queda formado por varios servicios coordinados: la interfaz, la API backend, PostgreSQL, Redis y el worker encargado de ejecutar auditorías e inferencias en segundo plano.

B.1. REQUISITOS PREVIOS

Antes de iniciar la instalación, el usuario debe disponer de Docker y Docker Compose en la máquina donde se vaya a ejecutar la herramienta. También será necesario contar con un navegador web para acceder a la interfaz y, si el código se obtiene desde un repositorio remoto, tener Git instalado. Por otro lado, si se va a auditar un sistema RAG externo, la máquina también debe tener conectividad hacia el endpoint configurado como target. En caso contrario, la herramienta puede levantarse igualmente y probarse contra un entorno controlado de validación, pero no funcionará cuando se ejecuten pruebas contra el target externo real.

B.2. OBTENCIÓN DEL CÓDIGO FUENTE

El código fuente puede obtenerse desde el repositorio del proyecto o desde una carpeta entregada junto con la memoria. Si se descarga desde un repositorio Git, el proceso general sería el siguiente:

```
git clone https://github.com/100495899/TFG.git
cd Auditoria_RAG_Timing
```

Si el proyecto se entrega como carpeta comprimida, basta con descomprimirla y situarse en la carpeta principal de la herramienta:

```
cd Auditoria_RAG_Timing
```

En la raíz del proyecto debe encontrarse el archivo docker-compose.yml, junto con las carpetas correspondientes al backend y al frontend.

B.3. CONFIGURACIÓN DEL ENTORNO

La configuración de la herramienta se realiza mediante variables de entorno. Para ello, la entrega debe incluir una plantilla sin secretos reales, a partir de la cual se genera el archivo de configuración local:

```
cp .env.example .env
```

El archivo .env contiene los valores necesarios para conectar los distintos servicios. Entre ellos se incluyen la configuración de PostgreSQL, la conexión a Redis, la clave usada para las sesiones, las credenciales iniciales del usuario administrador, los límites de ejecución de

auditorías y la URL del backend utilizada por el frontend. Antes de arrancar la aplicación, el usuario debe revisar este archivo y adaptar los valores necesarios a su entorno local. No deben incluirse claves reales, contraseñas de producción ni tokens privados en archivos versionados. Para una ejecución local se pueden usar valores de prueba, siempre que sean coherentes entre los distintos servicios. Por ejemplo, la URL de base de datos debe apuntar al servicio PostgreSQL definido en Docker Compose y la URL de Redis debe apuntar al servicio Redis usado por el worker.

B.4. ARRANQUE DE LA HERRAMIENTA

Una vez configurado el entorno, la herramienta se levanta desde la carpeta principal del proyecto con el siguiente comando:

```
docker compose up --build
```

Este comando construye las imágenes necesarias y arranca todos los servicios definidos en el proyecto.

El proyecto utiliza PostgreSQL como base de datos principal y Alembic para gestionar las migraciones. En la ejecución normal con Docker Compose, las migraciones se aplican durante el arranque del backend, por lo que no es necesario hacer nada manualmente para iniciar la aplicación.

El sistema también crea el usuario administrador inicial a partir de las variables configuradas en el archivo .env, siempre que no exista ya un usuario creado en la base de datos.

Durante el arranque, Docker Compose coordina los servicios necesarios para que la aplicación pueda funcionar de forma conjunta. Una vez que todos los contenedores están levantados, la herramienta queda preparada para su acceso desde el navegador.

B.5. ACCESO A LA INTERFAZ WEB

Cuando los servicios están levantados, el usuario puede acceder a la herramienta desde el navegador. En la configuración local, la interfaz queda disponible en: http://localhost:5173

El backend queda expuesto en: http://localhost:8000

También puede comprobarse el estado básico del backend mediante la ruta de salud configurada en la API: http://localhost:8000/health

Tras acceder a la interfaz, el usuario debe iniciar sesión con las credenciales del administrador definidas en el archivo .env. Una vez autenticado, podrá usar todas las funciones del sistema.

B.6. PARADA Y REINICIO DEL ENTORNO

Para detener la herramienta sin borrar los datos persistentes se utiliza:

```
docker compose down
```

Este comando detiene los contenedores, pero conserva los volúmenes asociados a la base de datos y a los archivos cargados. Por tanto, al volver a levantar el entorno, los datos almacenados seguirán disponibles.

Para arrancar de nuevo la herramienta, se puede ejecutar:

```
docker compose up
```

Si se han realizado cambios en el código o en las dependencias, conviene reconstruir las imágenes:

```
docker compose up --build
```

Si se desea eliminar completamente el entorno, incluyendo base de datos, datasets, auditorías, inferencias y resultados guardados, puede usarse:

```
docker compose down -v
```

Este último comando debe utilizarse con cuidado, ya que borra los volúmenes persistentes y elimina la información almacenada por la herramienta.

B.7. PROBLEMAS FRECUENTES

Si Docker no está iniciado, el comando de arranque no podrá conectarse al servicio de Docker. En ese caso, debe abrirse Docker Desktop o iniciarse el daemon correspondiente antes de ejecutar Docker Compose.

Si alguno de los puertos configurados ya está ocupado, uno o varios servicios no podrán levantarse correctamente. En ese caso, se debe liberar el puerto o modificar el mapeo correspondiente en el archivo docker-compose.yml.

Si el backend no consigue conectarse a PostgreSQL, conviene revisar que las variables del archivo .env coinciden con los servicios definidos en Docker Compose y que el contenedor de base de datos ha arrancado correctamente.

Si el worker no ejecuta auditorías o inferencias, debe comprobarse que Redis está disponible y que la URL de conexión configurada para la cola es correcta. También es útil revisar los logs del worker para verificar si los trabajos se están recibiendo y procesando.

Si el frontend no puede comunicarse con el backend, se debe revisar la URL base de la API configurada para la interfaz y la configuración CORS del backend. En ejecución local, ambos valores deben ser coherentes con los puertos expuestos por Docker Compose.

Si un target externo no responde, el problema puede estar en la URL, los headers, la plantilla del body, el timeout o la conectividad desde el contenedor del worker.

Si un dataset no se carga correctamente, debe comprobarse que el archivo es un JSON válido y que respeta la estructura esperada por la herramienta. Los formatos concretos de entrada y salida se recogen en el Anexo C.

ANEXO C. FORMATOS DE ENTRADA Y SALIDA

Este anexo recoge los principales formatos de entrada y salida utilizados por la herramienta Auditoria_RAG_Timing. Su objetivo es complementar la explicación funcional e implementación desarrollada en la memoria, mostrando de forma práctica cómo deben estructurarse los datos que introduce el auditor y qué información genera la herramienta durante las auditorías e inferencias. En este anexo se muestran únicamente los formatos que resultan relevantes para el uso de la herramienta que son configuración de targets, plantilla del body HTTP, datasets de auditoría, entradas para inferencia de términos y archivos CSV exportados.

C.1. CONFIGURACIÓN DE UN TARGET

Para que la herramienta pueda enviar consultas a un target específico, el auditor debe configurar los datos necesarios para construir una petición HTTP válida. Los campos principales de un target son:

- Nombre.
- URL del endpoint.
- Timeout de respuesta.
- Configuración de verificación TLS.
- Plantilla JSON de los headers de la petición.
- Plantilla JSON del cuerpo de la petición.

En este caso, todos los campos son claros, pero es bueno explicar que los campos que usan una plantilla JSON deben tener un JSON construido correctamente. Un ejemplo de los headers de un sistema sería el siguiente:

```
{
   "Content-Type": "application/json",
   "Authorization": "Bearer <TOKEN>"
}
```

Por otro lado, el cuerpo de la petición es importante que además de ser un JSON bien construido contenga algún campo con la variable {{QUERY}} como su valor. No todos los sistemas RAG reciben la pregunta del usuario con el mismo nombre de campo. Algunos endpoints pueden esperar un campo llamado question, otros query, otros message o incluso una estructura JSON más compleja. Por este motivo, la herramienta permite configurar una plantilla flexible para el body de la petición.

Un ejemplo sencillo sería:

```
{
  "question": "{{QUERY}}"
}
```

En este caso, si la herramienta debe enviar la consulta What is Bitcoin?, el body final enviado al target será:

```
{
  "question": "What is Bitcoin?"
}
```

También se puede usar una estructura más compleja si el sistema externo lo requiere:

```
{
 "input": {
   "message": "{{QUERY}}",
   "language": "en"
 },
 "options": {
   "stream": true
 }
}
```

Durante la ejecución, la herramienta conserva el resto de campos de la plantilla y sustituye únicamente el marcador {{QUERY}}. Esto permite adaptar la herramienta a distintos endpoints sin modificar el código.

C.2. FORMATO DEL DATASET DE AUDITORÍA

El dataset de auditoría contiene las consultas que se enviarán al sistema RAG externo durante una prueba. Estas consultas deben estar clasificadas por frecuencia y longitud, ya que la herramienta utiliza estas etiquetas para comparar grupos durante el análisis estadístico. Por esto, la herramienta permite descargar una plantilla con la estructura JSON ya creada para que el usuario solo tenga que rellenarla con consultas. La plantilla entregada al usuario contiene esto:

```
{
 "alta_frecuencia": {
   "corta": [],
   "media": [],
   "larga": []
 },
 "media_frecuencia": {
   "corta": [],
   "media": [],
   "larga": []
 },
 "baja_frecuencia": {
   "corta": [],
   "media": [],
   "larga": []
 }
}
```

El usuario debe rellenar las listas con consultas. Un ejemplo reducido de un dataset final sería el siguiente:

```
{
 "alta_frecuencia": {
   "corta": [
     "Time flies every day",
```

"Man needs more time"
],
"media": [
"Every man needs time to reflect on the days that have passed by quickly",
"The day started early when the man realized time was running out for completion"
],
"larga": [
"The wise old man sat quietly reflecting on how many days had passed since he first understood that time was the most precious commodity any man could possess in his entire lifetime of experiences",
"Day after day the young man worked tirelessly in the fields wondering if there would ever come a time when he could rest and enjoy the fruits of his labor without worrying about tomorrow's sunrise"
]
},
"media_frecuencia": {
"corta": [
"People face the world",
"Friend by my side"
],
"media": [
"People around the world gather at night to hear god's word spoken clearly",
"My friend stood by my side when I needed to face the difficult world"
],
"larga": [
"In this vast and complex world people search for meaning through god's word while friends stand by their side offering support and comfort during the darkest hours of the night when doubts and fears threaten to overwhelm",
"Every night when the world grows quiet and people rest their weary heads true friends gather together to study god's word and discuss how his divine plan affects every part of their lives and the lives of those they love"
]
},
"baja_frecuencia": {
"corta": [
"Bitcoin mining requires electricity",
"Ethereum blockchain validates transactions"
],
"media": [
"Bitcoin cryptocurrency reached an all-time high before crashing dramatically due to regulatory concerns",
"Elon Musk's Tesla company announced new electric vehicles with advanced autopilot features yesterday"
],
"larga": [
"Bitcoin cryptocurrency mining operations consume more electricity than entire countries while blockchain technology promises to revolutionize financial systems through decentralized ledgers that eliminate traditional banking intermediaries and enable peer-to-peer transactions globally",

"Tesla's autopilot system uses multiple cameras and neural networks to navigate highways but still requires driver attention despite Elon Musk's promises of full self-driving capability coming soon for years without delivering"
  ]
 }
}

C.3. ENTRADA PARA INFERENCIA DE TÉRMINOS

En las auditorías no hay ningún parámetro que necesite un formato de entrada especial, pero en la inferencia de términos sí, ya que en esta hay otro campo con un formato JSON específico. En este campo, la herramienta permite dos formas de entrada: términos simples o consultas manuales agrupadas. En ambos, cuando se selecciona la herramienta, ofrece el JSON vacío con la estructura ya hecha, pero es bueno mostrar un ejemplo de cada una.

En el modo basado en términos, el auditor introduce una lista de términos que quiere evaluar. También puede incluir controles negativos, que son términos que no deberían estar presentes en la base de conocimiento y que ayudan a comprobar la estabilidad de la prueba. Un ejemplo sería:

```
{
 "terms": [
  "bitcoin",
  "playstation",
  "electric vehicle"
 ],
 "negative_controls": [
  "zenthorium",
  "novalith"
 ]
}
```

El segundo modo de inferencia permite introducir directamente las consultas que se enviarán al sistema, agrupadas por término. Este formato ofrece más control al auditor, ya que permite definir frases concretas y adaptadas al contexto de la prueba.

Un ejemplo sería:

```
{
 "bitcoin": [
  "What is Bitcoin?",
  "Explain how Bitcoin transactions work",
  "Describe the role of Bitcoin in digital payments"
 ],
 "playstation": [
  "What is PlayStation?",
  "Explain the main features of PlayStation consoles",
  "Describe the role of PlayStation in the video game industry"
 ],
 "zenthorium": [
  "What is zenthorium?",
```

```
    "Explain the properties of zenthorium",
    "Describe the use of zenthorium in synthetic systems"
  ]
}
```

Además, en la creación de inferencias se puede seleccionar utilizar un CSV como referencia de calibración. El archivo debe permitir identificar, como mínimo, los grupos de frecuencia alta y baja, el grupo de longitud corta de esos grupos de frecuencia, la media de esos grupos y el std de esos grupos. Un ejemplo de un archivo mínimo válido para la herramienta sería:

```
frequency,length,mean_ttfb_ms,std_ttfb_ms
high,short,125.40,12.20
low,short,190.80,20.40
```

C.4. SALIDA DE RESULTADOS DE AUDITORÍA

Una vez finalizada una auditoría, la herramienta permite consultar los resultados desde la interfaz y exportarlos en formato CSV. Esta exportación conserva las mediciones individuales generadas durante la prueba, lo que permite revisar los datos fuera de la aplicación o analizarlos posteriormente con herramientas externas. El archivo CSV de auditoría contiene muchas columnas porque cada medición necesita conservar información de contexto, tiempos, respuesta HTTP y posibles errores. Para facilitar su comprensión, en la Tabla VIII se puede observar la división en grupos de los campos.

Tabla VIII. GRUPOS DE CAMPOS INCLUIDOS EN LA EXPORTACIÓN CSV DE AUDITORÍAS.

Grupo	Campos principales	Descripción
Identificación de la auditoría	session_id, target_name, dataset_name, audit_status, random_seed	Permiten identificar la auditoría, el target evaluado, el dataset usado y la semilla de ejecución.
Información temporal de la sesión	audit_started_at, audit_completed_at, audit_duration_seconds	Indican cuándo empezó y terminó la auditoría y cuánto duró la ejecución completa.
Información de la consulta	request_index, query_text, query_length_chars, frequency_tag, length_tag	Describen cada consulta enviada al target y las etiquetas usadas para agruparla en el análisis.
Métricas temporales	latency_ms, ttfb_ms, full_response_ms	Recogen los tiempos medidos durante la petición. El campo principal es ttfb_ms.
Respuesta HTTP	status_code, response_size_bytes	Permiten revisar si el target respondió correctamente y el tamaño de la respuesta recibida.

Errores y valores extremos	is_error, error_type, error_message, is_extreme_outlier	Permiten distinguir peticiones fallidas o valores que han sido marcados como extremos durante el análisis.
Marca temporal individual	timestamp	Indica el momento concreto en el que se registró la medición.

Un ejemplo reducido de una exportación de auditoría sería el siguiente:

```
session_id,query_text,frequency_tag,length_tag,ttfb_ms,full_response_ms,status_code,is_error
5,"What is time?",high,short,125.40,480.10,200,false
5,"What is zenthorium?",low,short,190.20,550.80,200,false
```

El CSV real contiene más columnas que este ejemplo, pero la lógica es la misma: cada fila representa una petición enviada al target durante la auditoría. Esta estructura permite conservar la trazabilidad completa de la prueba y relacionar los resultados agregados de la interfaz con las mediciones individuales.

La herramienta también puede generar una salida agregada por grupos. En este caso, las filas no representan peticiones individuales, sino combinaciones de frecuencia y longitud. Estos resúmenes incluyen métricas como número de muestras, media, mediana, desviación típica, percentiles, valores mínimos y máximos, valores extremos y tasa de error. Su objetivo es facilitar la comparación entre grupos sin tener que revisar manualmente todas las peticiones. Este resumen, aunque incluya más datos de los necesarios, se puede usar como referencia en la inferencia de términos.

C.5. SALIDA DE RESULTADOS DE INFERENCIA DE TÉRMINOS

La exportación de una inferencia de términos recoge la clasificación calculada para cada término evaluado. A diferencia del CSV de auditoría, este archivo no está centrado en cada consulta individual, sino en el resultado final obtenido para cada término. En la Tabla IX se ve como los campos principales pueden agruparse.

Tabla IX. GRUPOS DE CAMPOS INCLUIDOS EN EL CSV DE INFERENCIA DE TÉRMINOS.

Grupo	Campos principales	Descripción
Identificación de la inferencia	session_id, target_id, source_type, source_label	Permiten identificar la sesión de inferencia, el target evaluado y la fuente de calibración usada.
Información del término	term, is_control	Indican el término evaluado y si se trata de un control negativo.
Clasificación	classification, closest_reference	Muestran la decisión calculada por la herramienta y el grupo temporal más cercano.

Métricas temporales	observed_mean_ttfb_ms, observed_std_ttfb_ms, distance_to_threshold_ms	Recogen la media observada, la desviación típica y la distancia respecto a la frontera temporal de decisión.
Calidad de la medición	valid_measurements, total_measurements, error_count	Permiten saber cuántas mediciones válidas se han usado y si se han producido errores durante la prueba.

Un ejemplo reducido de exportación sería:

```
session_id,term,classification,observed_mean_ttfb_ms,valid_measurements,closest_reference,error_count
9,time,likely_present,128.45,18,high,0 9,playstation,inconclusive,158.20,16,medium,2
```

ANEXO D. DECLARACIÓN DE USO DE INTELIGENCIA ARTIFICIAL GENERATIVA

He usado IAG en mi TFG

Marca lo que corresponda:

SI	NO

Si has marcado SI, completa las siguientes 3 partes de este documento:

Parte 1: declaración sobre comportamiento legal, ético y responsable

Ten presente que el uso de IAG conlleva unos riesgos y puede generar una serie de consecuencias académicas graves: la evaluación de tu TFG puede verse comprometida si el uso de la IAG comporta la utilización de datos de carácter confidencial, materiales protegidos por derechos de autoría, o datos de carácter personal, y se hace sin cumplir las condiciones exigidas en cada caso (autorización de los interesados, autorización de los titulares, seguimiento de las instrucciones de la Universidad).

Pregunta	
1. En mi interacción con herramientas de IAG he facilitado **datos de carácter confidencial** contando siempre con la debida autorización de los interesados. La confidencialidad abarca toda información que una persona u organización desea proteger por razones legales, comerciales, de privacidad o estratégicas (como patentes o secretos comerciales).	
SÍ, he usado estos datos con la autorización de los interesados	**NO, no he usado datos de carácter confidencial**
2. En mi interacción con herramientas de IAG he facilitado **materiales protegidos por derechos de autoría** contando siempre con la autorización de los respectivos titulares.	
SÍ, he usado estos materiales con autorización de los titulares de derechos de autor; o bien sin ella porque se ajustan a una de las excepciones o límites que permite la ley: • **obra en dominio público** • **obra licenciada (licencias Creative Commons)** • **uso de fragmentos con fines de investigación (derecho de cita)**	NO, no he usado materiales protegidos por derechos de autoría

3. En mi interacción con herramientas de IAG he facilitado **datos de carácter personal** con la debida autorización de los interesados.	
SÍ, he usado estos datos con autorización de los interesados y conforme a las instrucciones contenidas en la guía aprobada por la Universidad	**NO, no he usado datos de carácter personal**
4. Mi utilización de la herramienta de IAG ha **respetado sus términos de uso**, así como los principios éticos esenciales, no orientándola de manera maliciosa a obtener un resultado inapropiado para el trabajo presentado, es decir, que produzca una impresión o conocimiento contrario a la realidad de los resultados obtenidos, que suplante mi propio trabajo o que pueda resultar en un perjuicio para las personas.	
SI	NO

Parte 2: declaración de uso técnico

Declaro haber hecho uso de los sistemas de IAG ChatGPT y Codex de OpenAI **para:**

Documentación y redacción:

- *Soporte a la reflexión en relación con el desarrollo del trabajo: proceso iterativo de análisis de alternativas y enfoques utilizando la IAG*

He utilizado IAG como apoyo para reflexionar sobre la organización general del TFG y la forma de presentar el trabajo. Por ejemplo, he solicitado ayuda para ordenar la memoria en capítulos y subapartados coherentes, diferenciando entre estado del arte, diseño, implementación, pruebas, resultados, conclusiones y anexos.

- *Revisión o reescritura de párrafos redactados previamente*

He utilizado IAG como apoyo para plantear la estructura inicial de algunos párrafos y ordenar las ideas principales que quería transmitir. A partir de esas propuestas, he reescrito y mejorado el texto manualmente, adaptándolo al contenido real del trabajo y a mi forma de explicar el proyecto. Además, he realizado una revisión manual posterior para comprobar que el apartado mantuviera coherencia con la memoria.

- *Búsqueda de información o respuesta a preguntas concretas*

He utilizado IAG para resolver dudas técnicas concretas relacionadas con cosas como el funcionamiento interno de los sistemas RAG o la forma de utilizar un sistema de colas con Redis. Estas respuestas se usaron como apoyo para entender conceptos y ordenar explicaciones.

- *Búsqueda de bibliografía*

No aplica. No he utilizado IAG para este fin.

- *Resumen de bibliografía consultada*

He utilizado IAG para apoyar la comprensión de bibliografía y documentación técnica consultada. Por ejemplo, he pedido explicaciones de ideas principales de trabajos relacionados para entenderlas mejor. El contenido incorporado a la memoria fue contrastado y redactado finalmente por mí.

- *Traducción de textos consultados*

No aplica. No he utilizado IAG para este fin.

Desarrollar contenido específico

Se ha hecho uso de IAG como herramienta de soporte para el desarrollo del contenido específico del TFG, incluyendo:

- *Asistencia en el desarrollo de líneas de código (programación)*

He utilizado herramientas de IAG como apoyo durante el desarrollo de código en algunos escenarios, como entender errores técnicos producidos durante la instalación y configuración de Redis o implementar mejor el trabajo con procesos asíncronos de algunos módulos del backend. La IAG sirvió como apoyo para razonar sobre este tipo de problemas y plantear posibles soluciones.

- *Generación de esquemas, imágenes, audios o vídeos*

He utilizado IAG de forma puntual para la elaboración de una figura incluida en la memoria. En concreto, la Fig. 7 fue generada inicialmente con ayuda de ChatGPT como apoyo para representar visualmente las carpetas del proyecto. A partir de esa versión inicial, la figura fue modificada y ajustada manualmente por mí para realizar distintos arreglos.

- *Procesos de optimización*

He utilizado IAG como apoyo en procesos de depuración y mejora técnica. Por ejemplo, he pedido ayuda para interpretar errores de ejecución, revisar problemas de conexión entre servicios y razonar sobre el procesamiento asíncrono. Las soluciones aportadas por la IA se implementaron de forma manual, entendiendo el problema encontrado y la solución desarrollada.

- *Tratamiento de datos: recogida, análisis, cruce de datos…*

No aplica. No he utilizado IAG para este fin.

- *Inspiración de ideas en el proceso creativo*

No aplica. No he utilizado IAG para este fin.

- *Otros usos vinculados a la generación de puntos concretos del desarrollo específico del trabajo*

No aplica. No he utilizado IAG para otros usos distintos a los declarados anteriormente.

Parte 3: reflexión sobre utilidad

Aporta una valoración personal (formato libre) sobre las fortalezas y debilidades que has identificado en el uso de herramientas de IAG en el desarrollo de tu trabajo. Menciona si te ha servido en el proceso de aprendizaje, o en el desarrollo o en la extracción de conclusiones de tu trabajo.

El uso de herramientas de Inteligencia Artificial Generativa me ha resultado útil durante el desarrollo del TFG, sobre todo como apoyo para ordenar ideas, resolver dudas técnicas, sugerir estructuras iniciales para la redacción y revisar la claridad de algunos apartados. En un trabajo que combina investigación experimental y desarrollo de software, la IAG ha servido como una herramienta de acompañamiento para plantear alternativas y mejorar la forma de explicar decisiones técnicas.

Una de las principales fortalezas que he identificado es su capacidad para ayudar a estructurar información compleja. En varios momentos del trabajo había que conectar conceptos de recuperación vectorial, latencia, sistemas RAG, caché, estadística y arquitectura de software. La IAG ha ayudado a plantear formas de ordenar esas ideas y convertirlas en apartados más comprensibles. También ha sido útil para revisar redacción y detectar frases poco claras. La IAG también ha sido útil durante el desarrollo técnico para resolver dudas sobre tecnologías concretas o interpretar errores. Las herramientas de IAG utilizadas no han sustituido el trabajo técnico, sino que han actuado como apoyo durante el proceso.

Por otro lado, las herramientas de IAG han generado algunos problemas al generar explicaciones no comprobadas o proponer soluciones que no encajan exactamente con el proyecto. Por este motivo, ha sido necesario revisar sus respuestas, contrastarlas con la implementación real y descartar propuestas que no se ajustaban al alcance del trabajo.

En conjunto, considero que el uso de IAG ha sido especialmente útil porque me ha permitido trabajar con mayor rapidez y dedicar más tiempo a las partes más importantes del proyecto. En un trabajo como este, donde se combinan investigación, experimentación, desarrollo software y redacción técnica, la IA ha servido como apoyo para avanzar de forma más ágil y llegar a un resultado más completo y sólido. Aun así, su valor depende de que haya una persona capaz de dirigir el proceso, tomar decisiones y entender técnicamente lo que está haciendo. Esto ha sido especialmente claro en la fase experimental, ya que la IAG podía ayudar a ordenar ideas o mejorar la forma de explicar los resultados, pero no podía sustituir el diseño de las pruebas, la ejecución de los experimentos ni la interpretación crítica de las mediciones obtenidas.