



Centro Universitario de la Defensa en la Escuela Naval Militar

TRABAJO FIN DE MÁSTER

Aplicación de Inteligencia Artificial en entornos aislados de Seguridad y Defensa: retos y soluciones

Máster Universitario en Dirección TIC para la Defensa

ALUMNO: Moisés Romero Sendino

DIRECTORES: Miguel Rodelgo Lacruz
Luis Álvarez Sabucedo

CURSO ACADÉMICO: 2025-2026

Universida_{de}Vigo



Centro Universitario de la Defensa en la Escuela Naval Militar

TRABAJO FIN DE MÁSTER

*Aplicación de Inteligencia Artificial en entornos aislados de
Seguridad y Defensa: retos y soluciones*

Máster Universitario en Dirección TIC para la Defensa
Especialidad de Sistemas y Tecnologías de Telecomunicación

UniversidadeVigo

RESUMEN

Este trabajo aborda el diseño, implementación y validación de una plataforma de inteligencia artificial orientada a entornos aislados en los que deben cumplirse requisitos estrictos de protección de la información, soberanía tecnológica y continuidad operativa en ausencia de conectividad externa.

La utilización de servicios cloud para el acceso a capacidades avanzadas de IA introduce riesgos relevantes en contextos con información sensible o clasificada, tales como la exposición de datos a terceros, la pérdida de control sobre los procesos de inferencia y la dependencia de la disponibilidad de infraestructuras externas. Frente a estas limitaciones, el presente trabajo propone una arquitectura alternativa basada en tecnologías de código abierto que permite la ejecución local de modelos de lenguaje de gran tamaño (LLM), sistemas de generación aumentada por recuperación (RAG), servicios de traducción automática y capacidades de generación de imágenes, sin requerir conectividad con redes externas.

La solución integra un motor de inferencia local para la ejecución de modelos LLM, una interfaz de usuario con soporte RAG apoyada en una base de datos vectorial, y servicios auxiliares especializados para traducción automática y generación de imágenes. La plataforma ha sido desplegada y evaluada tanto en una estación de trabajo con GPU dedicada NVIDIA RTX 4090 como en el superordenador FinisTerae III, este último orientado a tareas de entrenamiento y ajuste avanzado de modelos.

La validación se ha llevado a cabo mediante pruebas funcionales y de rendimiento en distintos escenarios de despliegue, confirmando la viabilidad técnica de operar capacidades de inteligencia artificial generativa en entornos completamente desconectados, con tiempos de respuesta compatibles con un uso interactivo y una calidad de salida adecuada para tareas de apoyo al análisis, al procesamiento documental y a aplicaciones relacionadas con la ciberseguridad.

PALABRAS CLAVE

Inteligencia artificial, LLM, Open WebUI, Ollama, Fine-tuning, MCP, RAG.

AGRADECIMIENTOS

En primer lugar, deseo expresar mi agradecimiento a mi familia por su comprensión, apoyo y paciencia durante el tiempo dedicado a la realización de este Máster DIRETIC. Asimismo, quiero agradecer a mis directores de TFM, Miguel Rodelgo Lacruz y Luis Álvarez Sabucedo, por su orientación, disponibilidad y aportaciones durante el desarrollo de este trabajo.

También quiero agradecer a los profesores del Centro Universitario de la Defensa que han impartido las asignaturas del máster, así como al Centro de Supercomputación de Galicia (CESGA) por facilitar el acceso al superordenador FinisTerrae III, imprescindible para las pruebas de entrenamiento de modelos.

Finalmente, quiero agradecer a mis compañeros del Máster, con quienes he compartido muchos buenos momentos y que han enriquecido esta experiencia académica.

CONTENIDO

1	Introducción y objetivos	7
1.1	Introducción	7
1.2	Objetivos	8
1.3	Estructura de la memoria	9
2	Estado del arte	11
2.1	Introducción a la inteligencia artificial y los modelos de lenguaje (LLM).....	11
2.2	Arquitectura Transformer y LLM	11
2.2.1	Evolución histórica de los LLM	12
2.2.2	Familias principales de LLM.....	13
2.2.3	Parámetros clave de los LLM	14
2.3	Modalidades de despliegue de LLM.....	15
2.3.1	LLM como servicio (SaaS).....	15
2.3.2	Cloud soberanos.....	15
2.3.3	On-premise con conectividad	15
2.3.4	On-premise air-gapped (offline)	16
2.4	IA Offline: Concepto y justificación.....	16
2.4.1	Requisitos de soberanía tecnológica en defensa	16
2.4.2	Necesidad de IA offline	17
2.5	Soluciones comerciales para IA soberana.....	17
2.5.1	Stargate, Stargate UK y OpenAI for countries	17
2.5.2	Palantir AIP	18
2.5.3	Scale AI y Defense Llama	18
2.5.4	OpenEuroLLM	18
2.6	Ecosistema Open Source para IA offline	18
2.6.1	Runtimes de inferencia local	19
2.6.2	Interfaces de usuario	19
2.6.3	Modelos Open Source destacados	20
2.6.4	Servicios auxiliares para ecosistema offline.....	21
2.7	Técnicas de personalización de LLM	22
2.7.1	Embeddings: concepto y modelos	22
2.7.2	Retrieval-Augmented Generation (RAG): arquitectura y flujo de procesamiento	23
2.7.3	Bases de datos vectoriales	24
2.7.4	Fine-tuning y técnicas PEFT	25
2.8	Agentes autónomos y herramientas	26

2.8.1 Arquitecturas de agentes LLM	26
2.8.2 Model Context Protocol (MCP).....	27
2.8.3 Arquitectura MCP.....	27
2.8.4 Ejemplos de servidores MCP.....	27
2.9 Retos para la adopción de IA offline en Defensa.....	28
2.9.1 Retos tecnológicos	28
2.9.2 Retos organizativos.....	30
2.9.3 Retos económicos	31
2.9.4 Retos normativos y de cumplimiento	31
2.9.5 Retos de seguridad específicos de IA	32
3 Diseño de la solución	35
3.1 Principios de diseño	35
3.2 Requisitos del sistema	36
3.3 Escenarios de despliegue de la solución	36
3.3.1 Escenario de despliegue monopuesto	36
3.3.2 Escenario de despliegue en grupo de trabajo.....	36
3.3.3 Arquitectura global/centralizada.....	37
3.4 Componentes principales de la solución.....	37
3.4.1 Componentes seleccionados	37
3.4.2 Modelos de lenguaje	38
3.5 Diseño de la arquitectura de grupo de trabajo.....	39
3.5.1 Arquitectura general	39
3.5.2 Flujo de interacción	39
3.5.3 Comunicación entre servicios.....	39
3.6 Diseño de la interfaz de usuario	39
3.7 Diseño del subsistema RAG	40
3.7.1 Pipeline de ingesta	40
3.7.2 Estrategia de fragmentación	40
3.7.3 Estrategia de recuperación	40
3.7.4 Organización en colecciones	40
3.8 Diseño de servicios auxiliares.....	41
3.8.1 Traducción	41
3.8.2 Generación de imágenes	41
3.8.3 Asistencia a programación.....	41
3.8.4 Integración MCP.....	41
3.9 Casos de uso incluidos en la solución.....	41

3.9.1 Análisis y procesamiento de documentación.....	41
3.9.2 Apoyo a la ciberseguridad	41
3.9.3 Formación y capacitación	42
3.10 Limitaciones y decisiones de diseño	42
4 Implementación y resultados.....	45
4.1 Entornos de implementación.....	45
4.1.1 Estación de trabajo local (sin GPU dedicada)	45
4.1.2 Estación de trabajo local (con GPU dedicada)	46
4.1.3 Infraestructura de supercomputación (CESGA).....	46
4.2 Implementación del puesto individual aislado (sin GPU)	46
4.2.1 LM Studio	47
4.2.2 Modelos desplegados y rendimiento	48
4.2.3 Instalación de RAG.....	48
4.2.4 Resultados obtenidos	49
4.3 Implementación del entorno de grupo de trabajo (con GPU)	49
4.3.1 Despliegue del motor de inferencia (Ollama).....	49
4.3.2 Despliegue de la interfaz de usuario (Open WebUI).....	52
4.3.3 Modelos desplegados.....	54
4.3.4 Implementación del sistema RAG	54
4.3.5 Servicios auxiliares	59
4.3.6 Integración con agentes MCP	62
4.3.7 Resultados obtenidos	64
4.4 Infraestructura de supercomputación (CESGA)	67
4.5 Fine-tuning del modelo	69
4.5.1 Dataset de entrenamiento.....	69
4.5.2 Descarga y configuración del modelo base	69
4.5.3 Fusión del adaptador LoRA con el modelo base	71
4.5.4 Registro en Ollama	71
4.5.5 Integración en Open WebUI.....	72
4.6 RAG versus Fine-tuning: Criterios de selección.....	72
4.6.1 RAG como técnica preferente.....	72
4.6.2 Fine-tuning como técnica preferente	73
4.6.3 RAG y Fine-tuning como técnica preferente.....	73
4.7 Retos identificados y soluciones implementadas.....	73
4.7.1 Retos tecnológicos	73
4.7.2 Retos organizativos.....	74

4.7.3 Retos económicos	75
4.7.4 Retos normativos y de cumplimiento	75
4.7.5 Retos de seguridad específicos de IA	75
5 Conclusiones	77
5.1 Consecución de objetivos.....	77
5.2 Lecciones aprendidas	78
5.3 Limitaciones identificadas	79
5.4 Líneas de trabajo futuro	79
5.5 Consideraciones finales.....	80
6 Bibliografía.....	81
Anexo I: Script de descarga del corpus documental	85
Anexo II: LibreTranslate en Open WebUI.....	91
Anexo III: Script de auditoría de Open WebUI	94
Anexo IV: Fine-tuning de Mistral	102
Anexo V: Evaluación del modelo adaptado	105
Anexo VI: Integración de LoRA con Mistral.....	106
Anexo VII: Acrónimos	107

ÍNDICE DE FIGURAS

Figura 2-1 Flujo de procesamiento RAG (Fuente: Elaboración propia)	24
Figura 3-1 Arquitectura general de la solución de grupo de trabajo (Fuente: Elaboración propia).....	39
Figura 3-2 Funcionamiento RAG (Fuente: Elaboración propia).....	40
Figura 4-1 Funcionamiento de LM Studio (Fuente: Elaboración propia).....	47
Figura 4-2 AnythingLLM en entorno individual (Fuente: Elaboración propia)	48
Figura 4-3 Interfaz de Open WebUI (Fuente: Elaboración propia)	53
Figura 4-4 Ejemplo de fuente RAG en Open WebUI (Fuente: Elaboración propia)	56
Figura 4-5 Respuesta en Open WebUI (Fuente: Elaboración propia).....	57
Figura 4-6 Respuesta en Open WebUI (Fuente: Elaboración propia).....	58
Figura 4-7 Interfaz LibreTranslate (Fuente: Elaboración propia)	59
Figura 4-8 Integración de LibreTranslate en Open WebUI (Fuente: Elaboración propia)	60
Figura 4-9 Integración de StableDiffusion en Open WebUI (Fuente: Elaboración propia).....	61
Figura 4-10 Stable Diffusion en Open WebUI (Fuente: Elaboración propia).....	61
Figura 4-11 Interfaz de Continue en Visual Studio Code (Fuente: Elaboración propia)	62
Figura 4-12 Diagrama de bloques del plugin de Ghidra (Fuente: Elaboración propia)	62
Figura 4-13 Configuración de Ghidra (Fuente: Elaboración propia)	63
Figura 4-14 Integración de Ghidra en Open WebUI (Fuente: Elaboración propia).....	63
Figura 4-15 Integración Ghidra-MCP en Open WebUI (Fuente: Elaboración propia).....	64
Figura 4-16 Salida del modelo mistral-ciber (Fuente: Elaboración propia).....	72

ÍNDICE DE TABLAS

Tabla 2-1 Listado de LLM destacados (Fuente: Elaboración propia).....	13
Tabla 2-2 Modelos de embeddings para uso offline (Fuente: Elaboración propia)	22
Tabla 2-3 Comparativa de bases de datos vectoriales (Fuente: Elaboración propia).....	25
Tabla 3-1 Especificaciones arquitectura individual (Fuente: Elaboración propia)	36
Tabla 3-2 Especificaciones de la arquitectura de grupo de trabajo (Fuente: Elaboración propia)...	37
Tabla 3-3 Modelos LLM seleccionados para entorno individual (Fuente: Elaboración propia)	38
Tabla 3-4 Modelos LLM para entorno de grupo de trabajo (Fuente: Elaboración propia).....	38
Tabla 4-1 Estación de trabajo local sin GPU (Fuente: Elaboración propia)	45
Tabla 4-2 Estación de trabajo local con GPU (Fuente: Elaboración propia)	46
Tabla 4-3 Modelos desplegados en entorno individual (Fuente: Elaboración propia).....	48
Tabla 4-4 Modelos evaluados en entorno de grupo de trabajo (Fuente: Elaboración propia)	54
Tabla 4-5 Consulta sobre normativa específica (Fuente: Elaboración propia)	56
Tabla 4-6 Consulta de información no indexada (Fuente: Elaboración propia)	57
Tabla 4-7 Consulta en inglés sobre corpus científico (Fuente: Elaboración propia)	58
Tabla 4-8 Resultados pruebas funcionales (Fuente: Elaboración propia).....	64
Tabla 4-9 Resultados pruebas de integración (Fuente: Elaboración propia).....	65

1 INTRODUCCIÓN Y OBJETIVOS

1.1 Introducción

La inteligencia artificial (IA) ha experimentado en la última década un crecimiento significativo tanto en capacidad técnica como en adopción. En apenas una década, ha pasado de ser una disciplina técnica asociada a ámbitos especializados a integrarse de forma progresiva en entornos profesionales y en usos cotidianos a nivel individual. La aparición de sistemas de inteligencia artificial generativa a finales de 2022, con ChatGPT, desarrollado por OpenAI, como solución más relevante, aceleró este proceso de forma notable. El ritmo de adopción fue especialmente rápido, alcanzando cien millones de usuarios en dos meses [1], y evidenció que los modelos de lenguaje habían alcanzado un nivel de madurez suficiente para su uso generalizado. Esta expansión ha impulsado tanto el desarrollo de modelos cada vez más capaces como la aparición de herramientas y plataformas que facilitan su integración en flujos de trabajo existentes, situando la IA generativa en el centro de las estrategias de transformación digital de un número creciente de organizaciones.

La disponibilidad reciente de modelos de aprendizaje profundo, y en particular de modelos de lenguaje de gran tamaño (LLM), ha extendido su aplicabilidad a prácticamente todos los ámbitos, incluyendo asistencia a usuarios, generación y síntesis de documentación, análisis técnico, extracción de información, automatización de actividades repetitivas, entre otros. Como consecuencia, numerosos procesos organizativos y operativos están incorporando estas capacidades, lo que incrementa las exigencias técnicas asociadas al tratamiento de información, a la gestión del conocimiento y a la eficiencia operativa.

Este avance, sin embargo, se ha materializado principalmente a través de servicios en la nube, lo que facilita el despliegue y el acceso a capacidades avanzadas, pero introduce dependencias externas que no siempre son compatibles con entornos que requieren un control estricto sobre la información y sobre la tecnología utilizada. En escenarios de seguridad y defensa, donde la soberanía del dato y la operación en ausencia de conectividad externa son requisitos habituales, el reto consiste en trasladar de forma controlada parte de estas capacidades a entornos desconectados (*air-gapped*).

Este trabajo responde precisamente a esa necesidad de disponer de capacidades avanzadas de inteligencia artificial en entornos donde no existe conectividad externa y donde el tratamiento de la información exige mantener bajo control propio tanto el dato, como el software y la infraestructura. En estos escenarios, la dependencia de servicios en la nube no resulta viable, por lo que el objetivo ha sido diseñar e implementar una alternativa técnicamente viable y operativa, basada en ejecución local y apoyada en procedimientos reproducibles. Resulta necesario disponer de servicios que sean, en la medida de lo posible, independientes de terceros, en aras de avanzar hacia la soberanía digital y la alta disponibilidad de servicios estratégicos. La rápida evolución de los modelos y del ecosistema asociado

hace recomendable un enfoque práctico, orientado a implementar y validar un prototipo funcional que pueda servir como base para su adaptación y evolución futuras.

La solución desarrollada se estructura en dos escenarios operativos complementarios. Por un lado, un puesto individual de alto nivel de aislamiento, sin GPU dedicada, orientado a la simplicidad operativa y al uso de modelos compactos ejecutables íntegramente en CPU, para consultas y asistencia básica en condiciones desconectadas. Por otro lado, un entorno de grupo de trabajo con un servidor central dotado de GPU, diseñado para proporcionar capacidades compartidas a varios usuarios mediante una interfaz web, concentrando la carga de inferencia en el nodo de cómputo y manteniendo el funcionamiento interno del sistema sin dependencias externas. En ambos casos, se ha priorizado una arquitectura modular y documentada, con procedimientos de despliegue claros, reproducibles, para facilitar su adaptación a distintos niveles de hardware y a diferentes restricciones operativas.

En este documento se presentan, en primer lugar, el contexto y los objetivos del trabajo, junto con los fundamentos técnicos necesarios para justificar las decisiones adoptadas. A continuación, se describe la arquitectura propuesta y la implementación de los entornos definidos, incluyendo la integración del motor de inferencia y el subsistema de generación aumentada por recuperación (RAG). Por último, se exponen las pruebas realizadas, se analizan las limitaciones relevantes de cara a un despliegue en producción y se proponen líneas de evolución orientadas a reforzar la seguridad, la escalabilidad y la aplicabilidad operativa en entornos aún más restrictivos.

Este documento recoge información disponible hasta finales de 2025, fecha de cierre de este trabajo. Dada la rápida velocidad de este dominio, se recomienda a futuros lectores del documento la revisión de posibles avances posteriores a la información citada.

1.2 Objetivos

El objetivo principal de este trabajo consiste en diseñar, implementar y validar una plataforma de inteligencia artificial en modalidad offline, adaptada a los requisitos específicos de entornos desconectados de Internet, que permita disponer de capacidades funcionales de IA generativa con control local sobre los modelos, los datos y la ejecución.

Este objetivo general se descompone en los siguientes objetivos específicos:

- OE1. Analizar el estado del arte de los modelos LLM y las tecnologías asociadas, identificando aquellas que resultan aptas para su despliegue en entornos completamente desconectados, atendiendo a requisitos funcionales, de hardware y de madurez tecnológica.
- OE2. Identificar y estructurar los principales retos para la adopción de IA generativa en entornos aislados, incluyendo las limitaciones de cómputo, la integración de conocimiento interno verificable, la trazabilidad y los riesgos específicos de seguridad asociados a los LLM.
- OE3. Diseñar una arquitectura modular y extensible que integre las principales capacidades de la IA generativa: inferencia de modelos de lenguaje, generación aumentada por recuperación (RAG), traducción automática multilingüe, generación de imágenes y asistencia a la programación.
- OE4. Implementar un prototipo funcional sobre hardware de disponibilidad general, validando su operación en condiciones de aislamiento completo y documentando los procedimientos de despliegue y configuración.
- OE5. Evaluar el rendimiento y la utilidad de los resultados obtenidos mediante pruebas representativas, cuantificando capacidades y limitaciones del sistema en distintos escenarios de ejecución (monopuesto sin GPU y entorno de grupo de trabajo con GPU).
- OE6. Documentar las limitaciones identificadas y proponer líneas de trabajo futuro que permitan ampliar las capacidades de la plataforma y mejorar su aplicabilidad en escenarios operativos reales.

1.3 Estructura de la memoria

La memoria se estructura en seis capítulos, que abordan de forma progresiva los diferentes aspectos del trabajo realizado:

Capítulo 1. Introducción y objetivos. Presenta el contexto del trabajo, la motivación que lo fundamenta, los objetivos perseguidos, el alcance definido y la estructura del documento.

Capítulo 2. Estado del arte. Analiza los fundamentos teóricos de los modelos de lenguaje, la arquitectura Transformer, las técnicas de adaptación (RAG y *fine-tuning*), las herramientas disponibles para despliegue offline, las soluciones comerciales de referencia y las consideraciones de seguridad aplicables. Este capítulo aborda los objetivos OE1 y OE2.

Capítulo 3. Diseño de la solución. Describe la arquitectura propuesta, incluyendo los diferentes niveles de despliegue, los casos de uso identificados, los componentes del sistema y las decisiones de diseño adoptadas. Este capítulo aborda el objetivo OE3.

Capítulo 4. Implementación y resultados. Detalla el proceso de despliegue de cada componente, la infraestructura utilizada, las configuraciones aplicadas, los procedimientos de integración y la evaluación de las pruebas realizadas, incluyendo los resultados obtenidos y las métricas de rendimiento. Este capítulo aborda los objetivos OE4 y OE5.

Capítulo 5. Conclusiones y líneas futuras. Sintetiza las aportaciones del trabajo, las conclusiones extraídas, las limitaciones identificadas y las líneas de investigación futura. Este capítulo aborda el objetivo OE6.

La memoria se complementa con una sección de bibliografía que recoge las referencias utilizadas y un conjunto de anexos que incluyen los listados del código desarrollado.

2 ESTADO DEL ARTE

El despliegue de capacidades de inteligencia artificial en entornos offline representa un desafío tecnológico complejo. La necesidad de operar en redes aisladas, sin conectividad externa, impone restricciones únicas que han impulsado el desarrollo de arquitecturas específicas y soluciones orientadas a la soberanía tecnológica. Este capítulo examina el estado actual de las tecnologías que permiten implementar sistemas de IA offline, desde los fundamentos técnicos de los modelos de lenguaje hasta los retos para su implantación en entornos desconectados con importantes restricciones de seguridad.

2.1 Introducción a la inteligencia artificial y los modelos de lenguaje (LLM)

La Real Academia Española (RAE) define la Inteligencia Artificial (IA) como «disciplina científica que se ocupa de crear programas informáticos que ejecutan operaciones comparables a las que realiza la mente humana, como el aprendizaje o el razonamiento lógico».

Desde sus orígenes conceptuales en la década de 1950, cuando Alan Turing propuso su célebre test de Turing [2] para evaluar la capacidad de las máquinas de exhibir comportamiento inteligente, hasta los actuales sistemas de aprendizaje profundo, la IA ha experimentado múltiples etapas de desarrollo, cada una caracterizada por avances significativos y, ocasionalmente, por períodos de estancamiento.

El concepto de IA fue formalmente establecido en la conferencia de Dartmouth de 1956 [3], donde John McCarthy acuñó el término. Los primeros sistemas expertos de los años ochenta demostraron el potencial de codificar conocimiento humano en reglas programáticas, aunque su carácter rígido limitaba su aplicabilidad. El resurgimiento del aprendizaje automático en los años noventa, impulsado por el aumento de capacidad computacional y la disponibilidad de datos, sentó las bases para las redes neuronales profundas que dominan el panorama actual.

Un punto de inflexión se produjo en 2012 con AlexNet [4], al demostrar la superioridad de las redes neuronales convolucionales en tareas de visión por computador. Sin embargo, el avance más significativo para el procesamiento del lenguaje natural se produjo en 2017 con la introducción de la arquitectura Transformer, por investigadores de Google [5]. Esta arquitectura eliminó las limitaciones de procesamiento secuencial inherentes a las redes recurrentes, permitiendo el entrenamiento altamente paralelo que caracteriza a los modelos actuales.

2.2 Arquitectura Transformer y LLM

Los modelos LLM se fundamentan en la arquitectura Transformer, que constituye actualmente la base dominante para el procesamiento del lenguaje natural. Su funcionamiento se basa en el mecanismo de atención *multi-head attention*, que permite al modelo ponderar la relevancia de cada elemento de una secuencia respecto a todos los demás elementos simultáneamente. A diferencia de las arquitecturas recurrentes, donde la información debe propagarse secuencialmente a través de estados ocultos, la

atención multicabeza establece conexiones directas entre cualquier par de posiciones en la secuencia. Esto elimina la dependencia secuencial de los modelos recurrentes y permite paralelizar el entrenamiento, procesando simultáneamente todos los tokens de la secuencia.

Sobre esta base, los LLM son redes neuronales entrenadas con cantidades masivas de datos para predecir el siguiente token a partir del contexto previo. A través de este objetivo aparentemente sencillo, los LLM desarrollan representaciones profundas del lenguaje, lo que les permite ofrecer capacidades que van desde la generación de texto coherente hasta el razonamiento complejo y la resolución de problemas.

2.2.1 Evolución histórica de los LLM

A lo largo de la última década, los modelos LLM han pasado de ser prototipos de investigación a convertirse en una tecnología central en múltiples ámbitos, desde la búsqueda de información hasta la asistencia en tareas complejas. Su evolución ha estado marcada por hitos concretos que han modificado la forma de entrenar, escalar y alinear estos modelos con las necesidades humanas. En este contexto, BERT (2018) [6], desarrollado por Google, supuso un punto de inflexión al introducir el preentrenamiento bidireccional: a diferencia de modelos anteriores que procesaban el texto únicamente de izquierda a derecha, BERT utiliza la tarea de *masked language modeling*, en la que se ocultan tokens aleatorios y el modelo debe predecirlos utilizando el contexto de ambos lados, lo que mejoró de forma notable el rendimiento en tareas de comprensión del lenguaje.

Posteriormente, GPT-2 y GPT-3 (2019–2020), desarrollados por OpenAI, demostraron que escalar de manera masiva el tamaño del modelo y el volumen de datos de entrenamiento produce mejoras cualitativas en las capacidades, hasta el punto de que GPT-3, con aproximadamente 175.000 millones de parámetros, exhibió capacidades emergentes como el *few-shot learning*, al resolver tareas nuevas a partir de un número reducido de ejemplos incluidos directamente en el prompt, sin requerir ajustes adicionales [7].

El lanzamiento de ChatGPT en 2022 constituyó un hito en la adopción de los LLM por parte del público general, al integrar modelos de la familia GPT con técnicas de *instruction tuning* y *Reinforcement Learning from Human Feedback* (RLHF). Esta combinación permitió alinear el comportamiento del modelo con instrucciones expresadas en lenguaje natural y mejorar su capacidad para sostener interacciones conversacionales coherentes, útiles y con mecanismos de salvaguarda operativa orientadas a reducir respuestas inapropiadas.

En paralelo, entre 2023 y 2025 se produjo el auge de los modelos abiertos, encabezado por la familia LLaMA de Meta, que posteriormente se consolidó con LLaMA 3, LLaMA 3.1 y LLaMA 4, ofreciendo modelos de alta calidad con pesos abiertos en rangos de parámetros que van desde unos pocos miles de millones hasta centenares de miles de millones. Esta decisión facilitó el desarrollo comunitario, la experimentación académica y la aparición de variantes optimizadas para diferentes casos de uso.

Junto a LLaMA surgieron modelos abiertos de referencia como Mistral, Qwen o Phi-3; esta última familia, impulsada por Microsoft, demostró que modelos relativamente pequeños (del orden de 3–4 mil millones de parámetros), adecuadamente entrenados, pueden igualar o superar a modelos de mayor tamaño en tareas de lenguaje, programación y matemáticas, lo que impulsó la adopción de LLM en dispositivos *edge* y en entornos con recursos limitados, especialmente relevantes para escenarios offline.

Finalmente, la evolución de la familia GPT se orientó hacia modelos multimodales, con integración nativa de distintas modalidades de entrada y salida. En este marco, GPT-4o (2024) incorporó capacidades para razonar sobre texto, visión y audio en tiempo casi real, ampliando el espectro de interacción más allá del texto. Posteriormente, GPT-5 (2025) consolidó esta línea como modelo de propósito general con mayores capacidades y rendimiento, reforzando la convergencia entre interacción multimodal y razonamiento integrado. En paralelo, emergieron otras alternativas como Gemini (Google) y Claude (Anthropic), también centradas en el procesamiento multimodal, el soporte de contextos extensos y el refuerzo de mecanismos de seguridad.

2.2.2 Familias principales de LLM

El ecosistema de LLM se puede clasificar, de forma general, en dos grandes categorías: modelos propietarios, accesibles mediante API comerciales, y modelos abiertos cuyos pesos están disponibles para descarga y ejecución local controlada.

En el panorama reciente de la IA generativa, la evolución de los modelos LLM se orienta principalmente a mejorar el razonamiento y la planificación en tareas de múltiples pasos, integrar capacidades multimodales (texto, imagen y audio) de forma nativa, y a la utilización de arquitecturas basadas en agentes para implementar flujos de trabajo, en los que el sistema puede ejecutar secuencias de acciones con un mayor grado de autonomía.

En esta línea, modelos de la familia GPT de OpenAI, integran capacidades de razonamiento integrado y multimodalidad nativa, lo que reduce la necesidad de orquestación externa para abordar tareas complejas. Por su parte, la familia Claude 4.5 (Sonnet 4.5 y Opus 4.5) se describe como especialmente enfocada en tareas de programación [8] y en la construcción de agentes [8], con un énfasis explícito en prácticas de seguridad y transparencia asociadas al modelo. Finalmente, Gemini 3 incorpora un modo de razonamiento iterativo (“*Deep Think*”) basado en rondas sucesivas de razonamiento para explorar hipótesis, lo que refuerza su orientación a problemas de mayor complejidad cognitiva y en escenarios donde es necesario descomponer, evaluar y revisar las soluciones de manera gradual [9].

A continuación se presenta una tabla que incluye algunas de las principales familias de LLM disponibles en el ecosistema actual, indicando su proveedor, usos habituales y principales características técnicas.

Modelo / familia	Proveedor	Usos habituales	Observaciones técnicas
GPT-5 (ChatGPT)	OpenAI	Asistencia general, análisis, programación, generación de texto	Modelo generalista de altas prestaciones, uso transversal
Claude 4 / 4.5	Anthropic	Redacción, análisis de texto extenso, programación	Respuestas naturales y contextos largos. Destaca en tareas complejas.
Gemini 3	Google	Multilingüismo, análisis, tareas multimodales	Integración con ecosistema Google. Razonamiento y casos de uso avanzados.
Microsoft Copilot (LLM subyacente GPT)	Microsoft	Productividad, asistencia ofimática, desarrollo	Integración nativa en entornos corporativos
Mistral	Mistral AI	Asistencia general, programación, análisis, despliegues on-premise y offline	Familia de modelos abiertos optimizados para eficiencia y ejecución local
DeepSeek	DeepSeek AI	Programación, razonamiento técnico, análisis estructurado, investigación	Modelos abiertos con énfasis en razonamiento y tareas de código
Grok	xAI	Asistencia general, razonamiento	Integrado en plataforma X

Tabla 2-1 Listado de LLM destacados (Fuente: Elaboración propia)

En los últimos años, los modelos abiertos han alcanzado una madurez que les permite competir de forma realista con los modelos propietarios, contribuyendo a facilitar el acceso a la IA más avanzada. Meta LLaMA 4 se considera uno de los modelos base abiertos de referencia, incorporando capacidades de razonamiento en arquitecturas de gran escala, con variantes que alcanzan decenas de miles de millones de parámetros y, en algunos casos, configuraciones *Mixture of Experts (MoE)* [10], donde solo se activa un subconjunto de parámetros por token, lo que lo hace adecuado como base para investigación y para adaptaciones orientadas a despliegue.

En el ámbito de tareas especializadas, destacan familias de modelos orientadas explícitamente a programación, razonamiento y despliegue eficiente. Qwen3-Coder se ha diseñado para generación y comprensión de código a nivel de repositorio, soportando flujos de trabajo iterativos habituales en ingeniería de software, como depuración guiada, revisión de cambios y refactorización estructurada. Por su parte, DeepSeek-R1, desarrollado en el ecosistema tecnológico chino, adopta enfoques de escalado eficientes, incluyendo también variantes con arquitecturas MoE, con el objetivo de mejorar el rendimiento en tareas de razonamiento, aspecto relevante en escenarios con restricciones de GPU o presupuesto. Finalmente, los modelos de Mistral, desarrollados en el ámbito europeo, se orientan a un compromiso práctico entre calidad, latencia y eficiencia, con variantes que incorporan capacidades multimodales y que resultan adecuadas para despliegues, tanto centralizados como locales, en contextos donde la protección de los datos, el cumplimiento normativo y la autonomía tecnológica constituyen requisitos estratégicos.

En entornos de Defensa, los modelos abiertos presentan ventajas críticas: permiten auditoría completa del código y los pesos, eliminan dependencias de proveedores externos, y posibilitan el despliegue en redes aisladas.

2.2.3 Parámetros clave de los LLM

Para analizar las implicaciones prácticas de estos sistemas y los requisitos de hardware asociados a su despliegue, resulta conveniente considerar tres dimensiones técnicas fundamentales.

La primera es el número de parámetros, que sirve como un indicador aproximado de la capacidad del modelo para representar conocimiento, modelar estructuras lingüísticas y generalizar en tareas complejas. En el estado actual de la tecnología conviven modelos compactos, con del orden de cientos de millones de parámetros, orientados a ejecución en dispositivos con recursos limitados o escenarios de *edge computing*, con modelos de gran escala —frecuentemente denominados *frontier models*— que alcanzan cientos de miles de millones de parámetros totales o activos, destinados a tareas de razonamiento intensivo y a infraestructuras de cómputo de alta capacidad.

La segunda dimensión relevante es la ventana de contexto, definida como el número máximo de tokens que el modelo puede procesar dentro de una misma inferencia manteniendo disponible esa información para la generación. La evolución en este parámetro ha sido muy destacada en las últimas generaciones: Llama 3.1 incorporó ventanas de contexto de 128.000 tokens, mientras que Llama 4 Scout amplía este límite hasta aproximadamente 10 millones de tokens [11]. Este aumento permite pasar de interacciones basadas en documentos individuales a escenarios donde puede incorporarse, en una sola sesión, un volumen muy superior de contexto, como colecciones extensas de informes técnicos, repositorios de código o bases documentales amplias. No obstante, el aumento del contexto también impone mayores requisitos de cómputo y memoria para el procesamiento de la secuencia, por lo que su aprovechamiento efectivo depende de la infraestructura disponible y del diseño del flujo de consulta.

Finalmente, para posibilitar la ejecución de estos modelos en hardware no especializado, resulta necesario recurrir a mecanismos de cuantización. Esta técnica reduce la precisión numérica de los pesos del modelo, disminuyendo los requisitos de memoria y el volumen de transferencia de datos durante la inferencia, a cambio de una degradación acotada de la calidad. Como referencia, un modelo de aproximadamente 7 mil millones de parámetros en precisión FP16 requiere del orden de 14 GB de VRAM, sin considerar el consumo adicional asociado al *runtime* y a la memoria cache derivada del

tamaño de la ventana de contexto. Mediante cuantización a 4 bits, el almacenamiento de pesos puede reducirse a valores cercanos a 3,5–4 GB, habilitando la inferencia local en GPUs de consumo o estaciones de trabajo.

2.3 Modalidades de despliegue de LLM

Las modalidades de despliegue de modelos de lenguaje abarcan desde servicios completamente gestionados en la nube hasta implementaciones locales orientadas a maximizar el control y la soberanía tecnológica. Cada alternativa presenta diferencias relevantes en términos de control sobre los datos y el procesamiento, garantías de privacidad, costes operativos, rendimiento y grado de personalización. En consecuencia, la selección del modelo de despliegue debe alinearse con los requisitos operativos, de seguridad y de cumplimiento normativo aplicables en cada entorno. Este apartado revisa las opciones más habituales y analiza sus implicaciones técnicas, de seguridad y de conformidad normativa.

2.3.1 LLM como servicio (SaaS)

El modelo Software como Servicio representa la forma más accesible de consumir capacidades basadas en LLM, al ofrecer acceso a modelos alojados por el proveedor mediante interfaces web o APIs. Este enfoque reduce la necesidad de infraestructura propia y simplifica el ciclo de vida operativo (aprovisionamiento, escalado, mantenimiento y actualizaciones), permitiendo incorporar capacidades avanzadas con un esfuerzo de integración relativamente bajo.

No obstante, este modelo presenta limitaciones significativas en entornos con requisitos estrictos de seguridad. El procesamiento de solicitudes y, en algunos casos, de datos asociados se realiza en infraestructuras de terceros, lo que introduce riesgos de confidencialidad y reduce el control sobre la localización, custodia y tratamiento del dato. Asimismo, la dependencia de conectividad externa lo hace incompatible con redes aisladas o clasificadas. A ello se suma la dependencia operativa del proveedor respecto a cambios de servicio, disponibilidad, políticas de uso y condiciones contractuales, que pueden afectar a la continuidad y a la trazabilidad del sistema en escenarios críticos.

2.3.2 Cloud soberanos

El concepto de cloud soberano surge como respuesta directa a las preocupaciones sobre soberanía tecnológica y control de los datos cuando se utilizan infraestructuras externas. En términos generales, la soberanía del dato implica que el tratamiento y almacenamiento de la información quedan sujetos al marco legal de una geografía determinada y a los mecanismos de control y gobernanza aplicables. En este enfoque, plataformas como Azure OpenAI permiten desplegar modelos en regiones geográficas concretas, bajo marcos regulatorios locales. En este mismo ámbito se destaca el proyecto Stargate UK, anunciado en septiembre de 2025, que proyecta una infraestructura prevista de decenas de miles de GPUs.

2.3.3 On-premise con conectividad

El despliegue *on-premise* con conectividad controlada combina el uso de infraestructura propia con acceso restringido a redes externas para funciones específicas, como la actualización de software, la descarga de modelos y dependencias, o la integración puntual con servicios externos no críticos. Este enfoque resulta apropiado para organizaciones que requieren un elevado grado de control sobre los datos y el procesamiento, pero que pueden asumir una superficie de exposición limitada, definida y sujeta a mecanismos estrictos de supervisión y filtrado.

Desde el punto de vista técnico, suelen requerirse GPUs de alto rendimiento (por ejemplo, NVIDIA A100, H100 o H200), sistemas de orquestación como Kubernetes con soporte para GPU, mecanismos de autenticación y segmentación de red, así como personal especializado capaz de gestionar tanto la plataforma de hardware como la capa de software y seguridad.

Se destaca el equilibrio intermedio que ofrece entre flexibilidad operativa y soberanía tecnológica, permitiendo incorporar capacidades avanzadas de IA sin delegar completamente el control del sistema ni del ciclo de vida de los datos a terceros.

2.3.4 *On-premise air-gapped (offline)*

El despliegue *on-premise air-gapped* representa el máximo nivel de aislamiento operativo, ya que los sistemas de IA funcionan completamente desconectados de cualquier red externa. Esta modalidad se emplea cuando existen requisitos estrictos de compartimentación, confidencialidad y control del procesamiento, como en el tratamiento de información clasificada o de alta sensibilidad en ámbitos de Defensa, inteligencia y determinados entornos industriales críticos.

Su adopción implica contar con una infraestructura autónoma completa (cómputo, almacenamiento, redes internas y sistemas de respaldo), así como con procedimientos estrictos para la gestión del ciclo de vida del software. Las actualizaciones, nuevos modelos o parches de seguridad deben introducirse mediante medios físicos controlados, con verificación de integridad y trazabilidad de extremo a extremo. Además, resulta imprescindible disponer de repositorios locales de dependencias y paquetes, y de políticas claras de auditoría y control de cambios que aseguren que el entorno mantiene su condición *air-gapped* en todo momento.

2.4 IA Offline: Concepto y justificación

Se entiende por inteligencia artificial offline todo sistema de IA diseñado para operar de manera autónoma sin conectividad a redes externas, ejecutando los procesos de inferencia (y, cuando aplique, entrenamiento o adaptación) exclusivamente sobre infraestructura local y empleando modelos, datos y dependencias disponibles en el propio entorno. En este marco, la ausencia de conectividad no se trata como un estado excepcional o degradado, sino como condición de operación prevista y sostenida, incorporada de forma explícita en el diseño arquitectónico, la gestión del ciclo de vida del software y los procedimientos de explotación, incluyendo la provisión local de repositorios, actualizaciones controladas y mecanismos de verificación de integridad.

2.4.1 *Requisitos de soberanía tecnológica en defensa*

La soberanía tecnológica en el ámbito de la defensa se justifica por la necesidad de asegurar la operación y mantenimiento de capacidades críticas sin dependencias no controladas de terceros, especialmente cuando estos están sujetos a jurisdicciones ajenas o a condicionantes externos que puedan afectar a la disponibilidad, integridad o continuidad del servicio. En sistemas basados en IA, este principio se materializa en un conjunto de requisitos que abarcan el ciclo de vida completo de la solución.

En primer lugar, resulta necesario garantizar el control sobre el software, incluyendo la posibilidad de auditoría técnica del código, su mantenimiento y su modificación bajo procedimientos propios o mediante proveedores de confianza. En segundo lugar, debe asegurarse el control sobre los datos y el proceso de entrenamiento, lo que incluye la trazabilidad del origen de los datos, evaluación de sesgos y riesgos, y capacidad de reproducir o ajustar el proceso para mitigar vulnerabilidades o dependencias no deseadas. En tercer lugar, se requiere control sobre la infraestructura de ejecución, de modo que el hardware, las plataformas de virtualización y los entornos de operación permanezcan bajo un marco de custodia y jurisdicción definido, con garantías físicas y lógicas acordes al nivel de sensibilidad tratado. Finalmente, la soberanía exige capacidad de evolución autónoma, entendida como la facultad de actualizar, reentrenar, reconfigurar y desplegar la solución sin depender operativamente de la intervención directa del proveedor del modelo o de servicios externos, preservando la continuidad de misión y la verificabilidad de los cambios.

Estos requisitos condicionan la idoneidad de las modalidades de despliegue y constituyen el marco de referencia para justificar el uso de arquitecturas offline en escenarios con restricciones de seguridad elevadas.

2.4.2 Necesidad de IA offline

A partir de los requisitos anteriores, la IA offline se plantea como una modalidad de despliegue orientada a maximizar el control sobre datos, procesamiento e infraestructura, especialmente en entornos donde la conectividad no puede asumirse como disponible o confiable.

En múltiples escenarios tácticos y estratégicos, las comunicaciones pueden ser limitadas, degradadas o inexistentes por condiciones del entorno, por restricciones deliberadas o por acciones hostiles. Bajo estas circunstancias, la dependencia de servicios remotos introduce un riesgo directo para la continuidad de misión; la ejecución local permite mantener capacidades de análisis, consulta y apoyo a la decisión aun ante pérdida de enlace, denegación de servicio o degradación intencionada de las comunicaciones.

Adicionalmente, la operación offline reduce la exposición asociada a la transferencia de información fuera del perímetro controlado. En despliegues basados en servicios externos, el tratamiento de entradas y salidas implica, por diseño, que datos potencialmente sensibles atraviesen infraestructuras de terceros, dificultando la custodia y la trazabilidad completa del procesamiento. La ejecución local permite aplicar controles consistentes con entornos de alta exigencia: segmentación, control de accesos, registro de actividad, verificación de integridad y procedimientos reproducibles para la introducción de actualizaciones, modelos y dependencias.

Por último, la IA offline facilita una gestión autónoma del ciclo de vida del sistema, al permitir gobernar versiones, aplicar parches bajo ventanas operativas definidas y realizar ajustes con datos propios cuando proceda. Además, permite contextualizar las respuestas utilizando documentación interna mediante técnicas como RAG, asegurando que la información generada sea precisa y relevante para la organización. En conjunto, esta modalidad responde de forma directa a los requisitos de soberanía tecnológica y continuidad operativa definidos en el apartado anterior.

Antes de analizar el ecosistema de código abierto que fundamenta la solución propuesta, se presentan a continuación las alternativas comerciales existentes, ya que permiten contextualizar las capacidades disponibles.

2.5 Soluciones comerciales para IA soberana

Ante la necesidad de IA Offline, han surgido iniciativas comerciales que no solo proporcionan infraestructuras de cómputo avanzadas, sino que incorporan mecanismos de gobernanza diseñados para ofrecer mayor control sobre los datos, los modelos y los procesos operativos. Los apartados siguientes introducen algunas de las iniciativas de IA soberana existentes:

2.5.1 Stargate, Stargate UK y OpenAI for countries

Los proyectos Stargate y Stargate UK son ejemplos de arquitecturas para infraestructuras de IA a escala estatal [12]. Stargate articula varios centros de datos distribuidos, con nodos equipados con aceleradores NVIDIA de la familia Blackwell [13] y plataformas cloud como Oracle Cloud Infrastructure y Azure, diseñados para soportar cargas intensivas de entrenamiento e inferencia de modelos.

Stargate UK [14] extiende este enfoque al contexto europeo mediante un campus específico en el Reino Unido, orientado a proporcionar capacidad soberana de cómputo de IA bajo jurisdicción nacional, pero apoyándose en el stack tecnológico de OpenAI, Microsoft y NVIDIA.

OpenAI for countries [12], lanzado en mayo de 2025, extiende el modelo Stargate a naciones aliadas mediante cinco pilares: centros de datos locales, ChatGPT personalizado, colaboración en desarrollo de modelos, fondos para startups, e inversión cruzada.

En conjunto, estos proyectos ilustran un modelo híbrido en el que la infraestructura física y parte de la gobernanza permanecen ancladas en el ámbito estatal, mientras que las capas de modelo, software y operación se articulan en colaboración con grandes proveedores comerciales de tecnología.

2.5.2 Palantir AIP

Palantir AIP [15] ofrece una arquitectura que integra de forma unificada el gobierno del dato, la ejecución de modelos y el despliegue operativo en infraestructuras controladas por el cliente. Su diseño se articula sobre tres componentes principales: Ontology, que actúa como capa semántica común para estructurar y relacionar información operativa; Foundry, que proporciona capacidades avanzadas de gestión del dato, auditoría y cumplimiento; y Apollo, responsable del despliegue autónomo del software en entornos heterogéneos, incluidos sistemas clasificados y redes totalmente aisladas [16].

2.5.3 Scale AI y Defense Llama

En el ámbito de soluciones de IA orientadas a defensa, Scale AI desarrolla la plataforma Donovan, una plataforma dirigida a entornos gubernamentales que proporciona una interfaz de alto nivel para configurar flujos de trabajo asistidos por IA y su integración con fuentes de información y sistemas internos mediante conectores. Su propósito operativo es facilitar el uso de modelos fundacionales en procesos de análisis y gestión de información dentro de marcos de control establecidos por la organización (control de accesos, trazabilidad y gobernanza de datos), incluyendo escenarios de despliegue en redes con restricciones.

Sobre esta línea se incluye Defense Llama, una familia de modelos derivados de LLaMA 3 mediante ajuste supervisado y técnicas de alineamiento, entrenada con corpus específicos del dominio, como doctrina militar, derecho internacional humanitario y políticas relevantes, con el objetivo de adecuar el comportamiento del modelo a casos de uso de Defensa y a criterios de uso responsable. Este tipo de especialización se orienta, principalmente, a tareas de apoyo al análisis, síntesis de información, elaboración de borradores y asistencia en planificación, bajo supervisión humana y con controles de uso definidos por la organización.

En este contexto se enmarca el Proyecto Thunderforge (marzo de 2025), programa del Departamento de Defensa de Estados Unidos para el despliegue de agentes de IA en planificación militar, en el que participan Scale AI, Anduril y Microsoft [17].

2.5.4 OpenEuroLLM

OpenEuroLLM es un proyecto europeo orientado al desarrollo de modelos LLM alineados con los principios de soberanía digital, transparencia y cumplimiento normativo de la Unión Europea. La iniciativa tiene como objetivo la creación de modelos LLM multilingües entrenados sobre datos representativos del contexto europeo, garantizando el control sobre los datos de entrenamiento, la trazabilidad de los procesos y la adecuación a marcos regulatorios como el RGPD y el AI Act. El proyecto está diseñado como arquitectura modular, permitiendo el despliegue de los modelos tanto en infraestructuras cloud europeas como en entornos on-premise o aislados, con especial énfasis en casos de uso del sector público, defensa, industria crítica y administración electrónica [18].

2.6 Ecosistema Open Source para IA offline

La consolidación de modelos abiertos entre 2023 y 2025 ha favorecido el desarrollo de un ecosistema de código abierto complementario (motores de inferencia, interfaces de usuario y herramientas auxiliares) que permite desplegar y operar modelos de lenguaje de forma local, reduciendo la dependencia de servicios en la nube. La evolución de estos componentes ha dado lugar a un ecosistema técnicamente coherente para escenarios offline, con soporte para ejecución eficiente, integración de funcionalidades avanzadas como RAG y operación reproducible en infraestructuras propias.

Este ecosistema presenta características relevantes para entornos con requisitos de seguridad elevados. En primer lugar, la compatibilidad con diferentes interfaces, API HTTP/REST, OpenAPI, APIs propietarias, etc. facilita la interoperabilidad entre componentes y simplifica la transición desde despliegues basados en servicios externos. En segundo lugar, el soporte para cuantización y para formatos de modelos orientados a inferencia local permite ajustar el consumo de memoria y cómputo a

las restricciones del hardware disponible, desde estaciones de trabajo con GPU de consumo hasta servidores con aceleradores de alto rendimiento. Por último, la disponibilidad del código fuente posibilita la auditoría técnica, control del ciclo de vida de dependencias y adaptación del software a requisitos específicos de explotación, integración y seguridad.

2.6.1 *Runtimes de inferencia local*

Los runtimes de inferencia local constituyen la base operativa de la arquitectura de IA offline, al habilitar la ejecución local y optimizada de modelos de lenguaje sin dependencia de servicios externos. A continuación se describen las soluciones más relevantes:

Ollama: Plataforma que simplifica la ejecución de LLM locales. Proporciona funcionalidades para la descarga, gestión y ejecución con comandos simples. Ofrece soporte nativo para GGUF, API REST compatible con OpenAI, aceleración GPU multiplataforma, y biblioteca con más de 100 modelos [19].

llama.cpp: Implementación en C++ de inferencia LLM, optimizada para CPU. Es fundamento de muchas otras herramientas. Ofrece control granular del rendimiento, soporte de cuantización y capacidades para integraciones personalizadas [20].

LM Studio: Aplicación de escritorio multiplataforma con interfaz gráfica intuitiva. Permite buscar, descargar y ejecutar modelos desde Hugging Face, incluye servidor local compatible con API OpenAI y soporte MCP como host facilitando workflows avanzados [21].

LocalAI: Alternativa *drop-in* a la API de OpenAI que ejecuta modelos localmente. Soporta generación de texto, embeddings, transcripción de audio y generación de imágenes [22].

vLLM: Se orienta a entornos de producción que requieren alta eficiencia, escalabilidad y control sobre la latencia. Su diseño se basa en PagedAttention, un mecanismo de gestión de memoria que permite servir múltiples consultas simultáneamente sin incurrir en penalizaciones significativas de rendimiento [23].

2.6.2 *Interfaces de usuario*

Las interfaces de usuario locales permiten aprovechar las capacidades de los modelos desplegados en infraestructura propia, facilitando su uso por analistas y operadores sin necesidad de herramientas externas. A continuación, se presentan algunas de las plataformas más empleadas en entornos de IA offline por su versatilidad, integración y facilidad de operación.

Open WebUI: Open WebUI es una interfaz web diseñada para ofrecer una experiencia de chat similar a ChatGPT sobre modelos ejecutados en infraestructura local, con integración nativa con Ollama y compatibilidad con APIs tipo OpenAI. Destaca por su módulo RAG avanzado, con soporte para múltiples bases de datos vectoriales y múltiples proveedores de búsqueda web, y por funcionalidades orientadas al uso organizativo, como gestión de usuarios y roles, permisos RBAC granulares, historial de conversaciones persistente y personalización detallada de prompts y plantillas [24].

AnythingLLM: AnythingLLM es una plataforma integral orientada a la implementación de sistemas RAG en entornos empresariales, con un foco específico en el embedding y la gestión estructurada de documentos. Proporciona espacios de trabajo aislados para distintos proyectos, capacidades de procesamiento automático de la documentación, integración con múltiples LLM y agentes capaces de combinar búsqueda RAG, navegación web y generación de gráficos. Este conjunto de funcionalidades permite construir asistentes especializados sobre repositorios documentales internos de manera controlada, reproducible y alineada con los requisitos de gobernanza de la información de la organización [25].

Text Generation WebUI (Oobabooga): Text Generation WebUI es una interfaz orientada a usuarios avanzados que prioriza la flexibilidad y la experimentación con modelos de lenguaje. Soporta múltiples *backends* de inferencia, lo que permite seleccionar en cada caso la pila técnica más adecuada,

e incorpora desde la propia interfaz capacidades de *fine-tuning* mediante adaptadores LoRA, facilitando la adaptación de modelos a dominios específicos sin necesidad de desarrollar código adicional [26].

2.6.3 Modelos Open Source destacados

Durante el período 2024–2025 se ha observado una evolución significativa en el panorama de los modelos de lenguaje de código abierto. Los esfuerzos iniciales orientados a ampliar el acceso a tecnologías hasta entonces concentradas en laboratorios con alta capacidad computacional han derivado en un ecosistema en el que los modelos abiertos alcanzan niveles de desempeño comparables a los de alternativas propietarias en determinados dominios y condiciones de uso.

Esta mejora de rendimiento resulta muy relevante para organizaciones que operan bajo requisitos estrictos de soberanía tecnológica. Por primera vez, resulta técnicamente viable desplegar sistemas de IA de altas prestaciones utilizando exclusivamente modelos cuyos pesos y arquitecturas son accesibles y, en muchos casos, acompañados de información suficiente para su evaluación y auditoría, sin recurrir a servicios en la nube comerciales. La posibilidad de ejecutar estas capacidades en infraestructura propia, sin dependencia operativa de proveedores externos, constituye un habilitador esencial para su aplicación en entornos clasificados y redes restringidas.

El análisis que sigue examina las principales familias de modelos abiertos disponibles en la actualidad, evaluando sus características técnicas, sus fortalezas en distintos dominios y las consideraciones de licenciamiento relevantes para contextos de Defensa. La selección se ha orientado hacia modelos que combinan un rendimiento competitivo con requisitos de cómputo compatibles con hardware accesible, priorizando aquellos publicados bajo licencias que permiten su uso sin restricciones en aplicaciones gubernamentales, militares o de seguridad nacional.

LLaMA (Meta): Dentro del ecosistema de modelos de código abierto, la familia LLaMA desarrollada por Meta tiene un rol principal. LLaMA 3 y 3.1 introdujeron configuraciones de 8B, 70B y 405B parámetros, con ventanas de contexto extendidas hasta 128.000 tokens y rendimiento comparable a modelos propietarios de referencia. LLaMA 3.2 amplió el rango hacia modelos compactos (1B–3B) y variantes multimodales, mientras que LLaMA 3.3 demostró mejoras sustanciales en eficiencia, alcanzando con 70B el rendimiento previo de 405B. La llegada de LLaMA 4, basada en arquitecturas MoE, habilitó contextos de hasta 10 millones de tokens y modelos como Scout o Maverick, destinados a tareas de largo contexto y análisis documental a gran escala. La licencia Llama Community License permite su adopción institucional con restricciones mínimas, lo que la hace muy adecuada para despliegues offline [27].

Mistral AI: Familia de modelos LLM que se caracteriza por su orientación a la eficiencia computacional. Mistral 7B, publicado bajo licencia Apache 2.0, introdujo mejoras arquitectónicas (Sliding Window Attention, GQA) que optimizaron el rendimiento por parámetro. Mixtral 8x7B, basado en MoE, logró capacidades equivalentes a modelos densos de ~70B activando sólo 13B parámetros, lo que facilita su ejecución en hardware limitado. Modelos posteriores, como Mistral Small 3.1 (24B) y Mistral Large 2 (123B), ampliaron las capacidades hacia variantes multimodales y modelos de alto rendimiento ejecutables en infraestructura propia [28].

Qwen: La familia Qwen [29], desarrollada por Alibaba Cloud, se caracteriza por un soporte multilingüe amplio y por una gama de tamaños que abarca aproximadamente de 0,5B a 72B parámetros. La generación Qwen 2.5 incluye modelos orientados a comprensión general, razonamiento y programación, con un desempeño competitivo respecto a otros modelos abiertos de su segmento y ventanas de contexto de gran capacidad. La variante Qwen 2.5-Coder está especializada en tareas de ingeniería de software y análisis de código, con soporte para múltiples lenguajes de programación. Los modelos se distribuyen bajo licencia Apache 2.0, lo que permite su utilización en entornos comerciales y gubernamentales.

DeepSeek: La familia DeepSeek presenta configuraciones de alta eficiencia en el entrenamiento y la inferencia de modelos de gran escala. DeepSeek-V2 y DeepSeek-V3 emplean arquitecturas MoE combinadas con variantes de atención optimizadas, obteniendo resultados próximos a los de modelos propietarios entrenados con presupuestos sensiblemente superiores. El modelo DeepSeek-R1, centrado en razonamiento y entrenado mediante refuerzo, se consideró la primera alternativa abierta comparable a los modelos de razonamiento especializados de referencia. La licencia MIT permite uso sin restricciones [30].

Phi: La familia Phi, desarrollada por Microsoft, se caracteriza por el entrenamiento de modelos de tamaño moderado a partir de conjuntos de datos sintéticos seleccionados y filtrados con criterios de calidad. Las generaciones Phi-3 y Phi-4 muestran niveles de desempeño elevados en tareas de matemáticas, programación y razonamiento, en algunos casos comparables o superiores a los de modelos abiertos de mayor tamaño [31]. Modelos como Phi-4 14B pueden ejecutarse en GPUs de consumo mediante técnicas de cuantización, lo que reduce de forma significativa los requisitos de memoria y cómputo y los hace viables en entornos offline con recursos limitados. La licencia MIT permite su uso en organismos gubernamentales.

Gemma: La familia Gemma (Google DeepMind) ofrece modelos derivados de la serie Gemini con optimizaciones orientadas a eficiencia y escalabilidad. Gemma 2, en variantes de 2B, 9B y 27B, alcanza un rendimiento competitivo comparado con modelos sustancialmente mayores. Estas características permiten su ejecución tanto en estaciones de trabajo con GPU mediante técnicas de cuantización como en dispositivos edge equipados con aceleradores dedicados (NPUs), lo que abre la posibilidad de su despliegue en entornos móviles y sistemas embebidos avanzados. La variante de 27B puede ejecutarse en una única GPU con cuantización a 4 bits, lo que la convierte en una opción viable para organizaciones que buscan maximizar capacidad sin infraestructuras avanzadas. Su licencia, aunque relativamente permisiva, incluye restricciones específicas que deben evaluarse en contextos de Defensa [32].

2.6.4 Servicios auxiliares para ecosistema offline

Un ecosistema completo de IA offline requiere servicios adicionales más allá del LLM principal. El ecosistema de código abierto ha desarrollado soluciones estables para cubrir cada una de estas necesidades, diseñadas para operar sin dependencia de servicios externos. La integración con los sistemas de IA offline descritos en secciones anteriores permite construir plataformas adaptadas que ofrecen las funcionalidades requeridas en los entornos desplegados.

Esta sección examina los servicios auxiliares más relevantes para despliegues offline implementados en este trabajo, evaluando sus capacidades técnicas, requisitos de recursos, facilidad de integración y consideraciones específicas para el contexto analizado:

LibreTranslate: LibreTranslate proporciona traducción automática completamente offline, basada en los modelos Argos Translate que utilizan la arquitectura OpenNMT. El proyecto se distribuye bajo licencia AGPL. Soporta más de 50 idiomas con traducción offline, incluyendo las principales lenguas europeas, árabe, chino, japonés, coreano y ruso. LibreTranslate expone una API REST que facilita su integración con otros componentes del ecosistema.

Stable Diffusion: Stable Diffusion permite generar imágenes a partir de descripciones textuales sin dependencia de servicios en la nube. El modelo se distribuye bajo licencia permisiva, habilitando su uso en contextos gubernamentales. Soporta modificación de imágenes existentes (image-to-image), edición selectiva de regiones (inpainting) y extensión de imágenes (outpainting).

2.7 Técnicas de personalización de LLM

Los LLM preentrenados poseen conocimiento general extenso, pero con frecuencia necesitan ser adaptados para dominios específicos o tareas particulares. Existen dos enfoques principales para esta adaptación: Retrieval-Augmented Generation (RAG), que enriquece el contexto del modelo con información externa en tiempo de inferencia, y *fine-tuning*, que modifica los pesos del modelo mediante entrenamiento adicional.

2.7.1 Embeddings: concepto y modelos

Los embeddings son representaciones vectoriales que codifican el significado semántico del texto en vectores numéricos. A diferencia de representaciones tradicionales como *bag of words* o TF-IDF, que se basan principalmente en la frecuencia de términos, los embeddings neuronales sitúan en posiciones próximas del espacio vectorial aquellos textos que son semánticamente similares, aunque utilicen palabras distintas, lo que permite medir la similitud semántica.

La selección del modelo de embeddings condiciona directamente la calidad de la recuperación en sistemas RAG. Los factores determinantes incluyen la dimensionalidad del vector (que afecta tanto a la expresividad como al coste de almacenamiento y búsqueda), el contexto máximo soportado, la cobertura lingüística para aplicaciones multilingües, y las consideraciones de licenciamiento.

Se destacan los siguientes modelos de embeddings:

BGE-M3: Desarrollado por BAAI, es uno de los modelos de embeddings multilingües más utilizado en entornos offline. El modelo soporta más de 100 idiomas con rendimiento destacado en español, procesando secuencias de hasta 8.192 tokens. Licencia MIT [33].

E5-Mistral-7B-Instruct: Basado en Mistral-7B, ofrece embeddings de muy alta calidad a costa de mayor tamaño. Recomendado para despliegues donde la infraestructura lo permite y la calidad de recuperación es crítica. Dimensión: 4096; tamaño: 14GB aproximadamente. Licencia MIT [28].

GTE-Qwen2 (Alibaba): Familia de modelos de Alibaba con equilibrio entre rendimiento y eficiencia. Disponible en múltiples tamaños desde 1.536 hasta 3.584 dimensiones. Soporte multilingüe. Dimensión: 1536-3584. Licencia Apache 2.0 [34].

Nomic-Embed-Text: Modelo compacto y eficiente. El código es públicamente accesible. Dimensión: 768 dimensiones; soporte para 8.192 tokens. Licencia Apache 2.0 [35].

La Tabla 2-2 muestra los principales modelos de embeddings evaluados para su uso en entornos offline.

Modelo	Dimensión	Tamaño	Contexto	Fortaleza
BGE-M3	1024	2 GB	8192	Multilingüe, versátil
E5-Mistral-7B	4096	14 GB	32768	Máxima precisión
GTE-Qwen2-1.5B	1536-3584	1-4 GB	32768	Equilibrio calidad/eficiencia
Nomic-Embed	768	550 MB	8192	Compacto, eficiente

Tabla 2-2 Modelos de embeddings para uso offline (Fuente: Elaboración propia)

2.7.2 Retrieval-Augmented Generation (RAG): arquitectura y flujo de procesamiento

Los modelos de lenguaje, a pesar de su capacidad generativa, presentan dos limitaciones fundamentales: su conocimiento queda congelado en la fecha de entrenamiento y carecen de acceso directo a información específica de cada organización. RAG es una solución que aborda estas limitaciones combinando la capacidad generativa de los LLM con sistemas de recuperación de información externa.

El principio de funcionamiento de RAG consiste en enriquecer el contexto del modelo con fragmentos de documentos relevantes recuperados en tiempo de consulta. Cuando un usuario formula una pregunta, el sistema busca en una base documental los fragmentos más relacionados semánticamente, los incorpora al prompt junto con la pregunta original, y el modelo genera una respuesta fundamentada en esa información. Este enfoque reduce significativamente las alucinaciones al basar las respuestas en fuentes verificables y permite actualizar el conocimiento del sistema simplemente añadiendo o modificando documentos, sin necesidad de reentrenar el modelo [36].

La recuperación semántica que proporciona RAG, basada en *embeddings* tal y como se describe en la sección anterior, permite identificar documentos conceptualmente relacionados aunque utilicen vocabulario diferente. Esta capacidad resulta especialmente útil en entornos de Defensa, donde la terminología puede variar entre documentos doctrinales, operativos y técnicos, pero los conceptos permanecen relacionados [37].

Arquitectura y Componentes

Un sistema RAG típico consta de los siguientes componentes:

- **Base de Conocimiento:** Repositorio de documentos que contiene la información especializada del dominio. Puede incluir manuales técnicos, documentación, informes, procedimientos operativos o cualquier texto relevante.
- **Modelo de Embeddings:** Transforma tanto los documentos como las consultas en representaciones vectoriales de alta dimensionalidad. Modelos como BGE-M3, E5-Mistral o Nomic-Embed-Text, analizados en la sección precedente, capturan la semántica del texto permitiendo comparaciones de similitud conceptual.
- **Base de Datos Vectorial:** Almacena los embeddings de los documentos y permite búsquedas eficientes de similitud. Opciones populares incluyen ChromaDB, FAISS, Qdrant y Milvus.
- **Sistema de Recuperación:** Busca los documentos más relevantes para cada consulta. Puede utilizar búsqueda semántica, búsqueda híbrida o reordenamiento (reranking).
- **Modelo de Lenguaje (LLM):** Recibe la consulta del usuario junto con los documentos recuperados como contexto, generando una respuesta fundamentada.

Flujo de Procesamiento RAG

El flujo de procesamiento en un sistema RAG se compone principalmente de dos fases:

1. **Fase de Indexación:** Prepara la base de conocimiento para su consulta eficiente. Los documentos de la base de conocimiento se dividen en fragmentos (chunks) de tamaño apropiado, típicamente entre 256 y 1024 tokens con solapamiento del 10-20% para preservar continuidad contextual. Cada fragmento se transforma en su representación vectorial mediante el modelo de embeddings y se almacena en la base de datos vectorial junto con metadatos.
2. **Fase de Consulta:** Se ejecuta en tiempo real cuando el usuario formula una pregunta. El sistema genera el embedding de la consulta utilizando el mismo modelo empleado en la indexación, recupera de la base vectorial los k fragmentos con mayor similitud según la métrica configurada

(típicamente similitud coseno), construye un prompt que integra la pregunta original con los fragmentos recuperados como contexto, y finalmente el LLM genera una respuesta que se fundamenta explícitamente en la información proporcionada.

Este flujo garantiza que las respuestas del sistema reflejen el conocimiento contenido en la base documental, reduciendo la propensión a alucinaciones característica de los LLM que operan sin contexto externo, y habilitando la verificación de las fuentes que sustentan cada respuesta.

La Figura 2-1 muestra el flujo de procesamiento característico de un sistema RAG, desde la consulta del usuario hasta la generación de la respuesta.

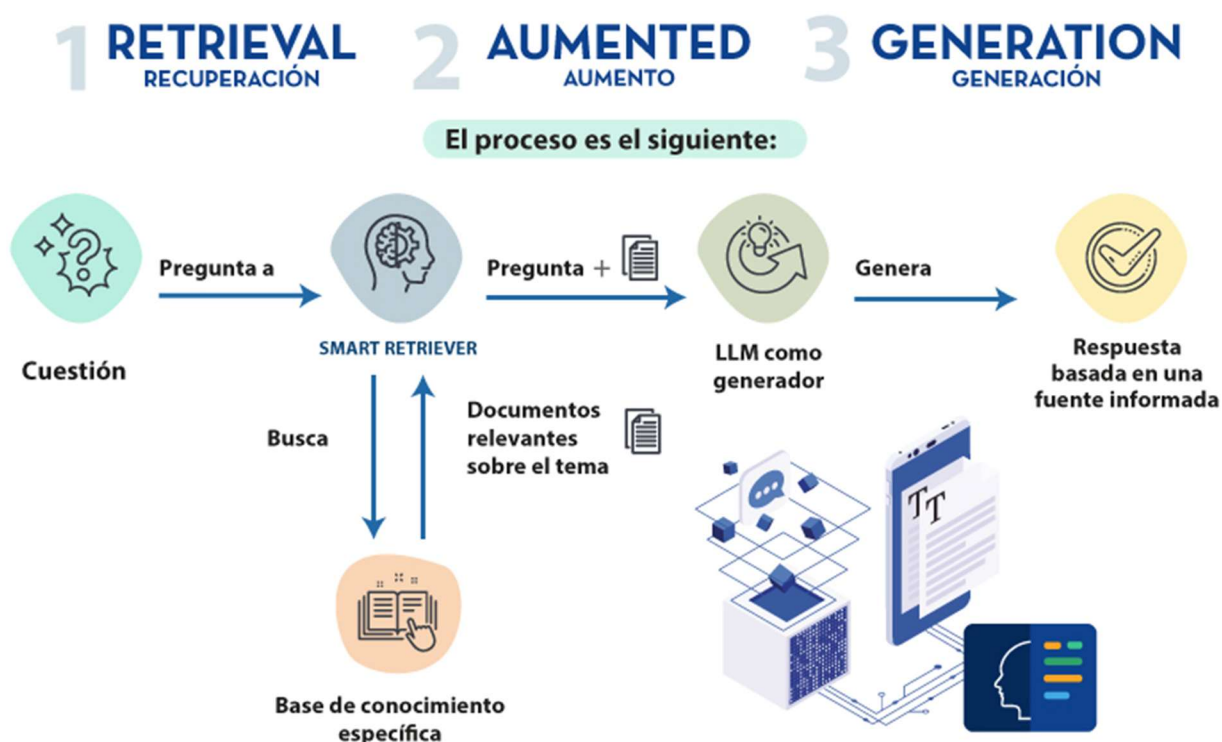


Figura 2-1 Flujo de procesamiento RAG (Fuente: Elaboración propia)

2.7.3 Bases de datos vectoriales

Las bases de datos vectoriales proporcionan el almacenamiento persistente y los mecanismos de búsqueda eficiente que sustentan la fase de recuperación en sistemas RAG. La selección de la base de datos apropiada debe considerar el volumen de vectores a indexar, los requisitos de latencia, las capacidades de filtrado por metadatos, y la complejidad operativa aceptable para el equipo de administración. Se destacan:

- **ChromaDB:** Solución adecuada para prototipado y despliegues de escala moderada. Su API nativa en Python y el almacenamiento en SQLite/DuckDB permiten integrarlo sin dependencias externas ni infraestructura adicional. El rendimiento es adecuado hasta volúmenes en torno al millón de vectores, pero se degrada en cargas mayores. Licencia Apache 2.0 [38].
- **FAISS:** Solución de referencia para búsqueda de similitud a gran escala, con algoritmos de indexación que permiten ajustar la precisión y el rendimiento. La aceleración mediante GPU habilita el procesamiento de miles de millones de vectores. Requiere mayor esfuerzo de integración al ofrecer una API de bajo nivel. Licencia MIT [39].

- **Qdrant:** Base de datos vectorial orientada a funcionalidad avanzada con operación sencilla. Permite filtrado por metadatos durante la búsqueda, recuperación híbrida (densa + dispersa) y acceso mediante API REST. Su modo distribuido posibilita escalado horizontal, mientras que RocksDB garantiza persistencia de baja latencia. Licencia Apache 2.0 [40].
- **Milvus:** Plataforma diseñada para entornos de escala empresarial, con arquitectura distribuida diseñada para volúmenes masivos y alta disponibilidad. Ofrece RBAC, aceleración GPU y soporte para múltiples tipos de índices. Licencia Apache 2.0 [41].

La Tabla 2-3 muestra una comparativa de las principales bases de datos vectoriales evaluadas para la implementación del sistema RAG.

Base de datos	Escala recomendada	Complejidad operativa	Caso de uso principal
ChromaDB	< 1M vectores	Baja	Prototipado, equipos pequeños
FAISS	Alta (dependiente de memoria)	Media	Máximo rendimiento, investigación
Qdrant	1M - 100M vectores	Media	Producción equilibrada
Milvus	> 100M vectores	Alta	Empresarial, alta disponibilidad

Tabla 2-3 Comparativa de bases de datos vectoriales (Fuente: Elaboración propia)

2.7.4 Fine-tuning y técnicas PEFT

Fine-tuning es una técnica de aprendizaje por transferencia en la que un modelo previamente preentrenado sobre un conjunto amplio y generalista de datos se adapta posteriormente a un dominio o tarea específicos mediante entrenamiento adicional sobre un conjunto de datos especializado. En lugar de entrenar un modelo desde cero, se reutiliza la representación lingüística y semántica adquirida durante el preentrenamiento (que suele implicar grandes volúmenes de datos y un coste computacional elevado) y se ajustan sus parámetros para mejorar el comportamiento en el contexto de aplicación seleccionado.

El *fine-tuning* tradicional consiste en volver a entrenar todos los parámetros del modelo sobre un nuevo conjunto de datos. Aunque suele proporcionar buenos resultados, presenta limitaciones claras: exige recursos computacionales elevados, obliga a mantener múltiples copias completas del modelo para cada tarea específica y puede provocar que el modelo pierda parte de las capacidades adquiridas en su entrenamiento original.

Las técnicas de *Parameter-Efficient Fine-Tuning* (PEFT) proponen otro tipo de solución, en lugar de modificar todos los pesos, entrenan únicamente un subconjunto reducido de parámetros adicionales, manteniendo congelado el modelo base. De este modo, se disminuyen los requisitos de memoria y cómputo, se facilita la gestión de versiones para distintas tareas y se mantiene mejor el comportamiento original del modelo [42].

LoRA: Low-Rank Adaptation

LoRA (*Low-Rank Adaptation*) es una de las técnicas de PEFT más empleadas para adaptar modelos de gran tamaño sin modificar todos sus parámetros. Parte de la observación de que, durante el ajuste fino, las variaciones efectivas en los pesos suelen concentrarse en un subespacio de rango bajo, lo que permite representarlas mediante matrices de menor dimensión sin pérdida significativa de capacidad [43].

Esta adaptación de modelos tiene varias ventajas. En primer lugar, disminuye de forma notable el uso de memoria y de recursos de GPU durante el entrenamiento y la inferencia, lo que facilita el ajuste de modelos grandes en entornos con recursos restringidos. En segundo lugar, permite reutilizar un mismo modelo preentrenado para múltiples tareas simplemente cargando y descargando distintos adaptadores LoRA, reduciendo las necesidades de almacenamiento frente a mantener copias completas del modelo. Finalmente, los pesos aprendidos mediante LoRA pueden fusionarse con los pesos originales en el momento del despliegue, sin añadir latencia apreciable en la inferencia, lo que hace posible obtener un comportamiento próximo al de un modelo completamente reajustado, pero con un uso mucho más eficiente de los recursos.

QLoRA: Quantized LoRA

QLoRA combina LoRA con cuantización a 4 bits, permitiendo *fine-tuning* de modelos de hasta 65B parámetros en una sola GPU de 48GB [44]. Introduce innovaciones técnicas como 4-bit NormalFloat (NF4), Double Quantization para reducir aún más la huella de memoria, y Paged Optimizers para gestión de memoria que evita picos durante el entrenamiento.

En el contexto de este trabajo, QLoRA se utilizó para entrenar, a modo de ejemplo, un adaptador especializado en ciberseguridad sobre el modelo Mistral-7B empleando únicamente 24 GB de VRAM [45].

2.8 Agentes autónomos y herramientas

En su configuración estándar, los modelos de lenguaje operan como componentes de inferencia que transforman una entrada en una salida generada a partir del contexto disponible, sin acceso directo a recursos externos en tiempo de ejecución. Esta característica limita su aplicabilidad cuando la tarea requiere información no contenida en el prompt o en los parámetros del modelo, o cuando es necesario interactuar con el entorno mediante operaciones como consultas a bases de datos, ejecución de código, acceso controlado a sistemas de archivos o invocación de servicios internos.

Los enfoques basados en agentes amplían estas capacidades al integrar el modelo dentro de un bucle de control que combina razonamiento, planificación y ejecución de acciones mediante herramientas. En este esquema, el LLM actúa como motor de decisión que selecciona operaciones, encadena pasos y consume los resultados para refinar el estado de la tarea hasta alcanzar un objetivo definido, bajo restricciones de seguridad y trazabilidad. Esta arquitectura habilita, entre otros, escenarios de análisis y generación asistidos que combinan consultas estructuradas (SQL, APIs internas, etc.) con síntesis documental; asistencia técnica y operativa basada en ejecución controlada de acciones (búsqueda en repositorios, gestión de incidencias, verificación de configuraciones); y automatización de flujos de trabajo mediante orquestación de herramientas (ejecución de scripts, validación de resultados y generación de informes).

2.8.1 Arquitecturas de agentes LLM

Un agente basado en LLM puede describirse, de forma general, mediante cuatro componentes funcionales que operan de manera coordinada:

- **Núcleo LLM:** actúa como motor de decisión. Interpreta el objetivo del usuario, integra las observaciones disponibles del entorno, selecciona la acción siguiente y evalúa los resultados obtenidos para determinar si el objetivo se ha alcanzado o si es necesario continuar con nuevas acciones. Las capacidades del agente dependen en gran medida del modelo utilizado y de su alineamiento con el dominio de aplicación.
- **Memoria:** permite mantener y recuperar contexto a lo largo del tiempo. Incluye memoria de corto plazo (historial y estado de la interacción) y, cuando se requiere persistencia, mecanismos de memoria de largo plazo soportados por almacenamiento estructurado o semiestructurado, frecuentemente mediante bases de datos vectoriales para recuperación semántica.

- **Sistema de Planificación:** habilita al agente para abordar tareas complejas mediante descomposición en subtareas más sencillas. Las capacidades de planificación incluyen descomposición jerárquica de objetivos, priorización de subtareas, establecimiento de hitos y revisión de resultados intermedios para ajustar la estrategia cuando las acciones no producen el efecto esperado.
- **Herramientas:** Las herramientas constituyen la interfaz del agente con el mundo exterior, proporcionando capacidades que el modelo de lenguaje no posee intrínsecamente como por ejemplo, acceso a bases de datos, ejecución de código, consulta de APIs, manipulación de archivos o interacción con sistemas especializados. Cada herramienta se define mediante un esquema que especifica su función, parámetros de entrada y formato de salida, permitiendo al modelo seleccionar y parametrizar la herramienta apropiada para cada situación.

2.8.2 Model Context Protocol (MCP)

Model Context Protocol (MCP) [46], presentado por Anthropic en el año 2024, define un estándar abierto para conectar aplicaciones LLM con fuentes de datos y herramientas externas. Antes de su aparición, la integración entre modelos y herramientas se realizaba de forma *ad hoc*, lo que generaba un ecosistema fragmentado en el que cada combinación requería conectores específicos. MCP introduce una capa de abstracción que separa clientes (aplicaciones que orquestan el modelo) y servidores (expositores de herramientas y recursos), permitiendo que cualquier cliente compatible consuma herramientas de cualquier servidor compatible sin desarrollar integraciones específicas para cada caso.

2.8.3 Arquitectura MCP

MCP define una arquitectura en tres capas con responsabilidades diferenciadas. El host MCP es la aplicación que integra el modelo de lenguaje y gestiona la interacción con el usuario (por ejemplo, un cliente de escritorio o una interfaz web con soporte MCP). El cliente MCP implementa el protocolo y proporciona la capa de comunicación: establece y mantiene las conexiones, gestiona la serialización de mensajes y el control de errores, y expone al host una interfaz uniforme para descubrir e invocar capacidades externas. Los servidores MCP encapsulan fuentes de datos o funcionalidades específicas y las publican mediante la interfaz estandarizada del protocolo, pudiendo cada servidor ofrecer una o varias capacidades dentro de su dominio.

En cuanto a la comunicación, MCP emplea mensajería estructurada en *JSON* con un patrón de intercambio basado en solicitudes, respuestas y notificaciones. Respecto al transporte, admite *stdio* para servidores locales ejecutados como procesos independientes (habitual en entornos offline) y HTTP(S) para escenarios remotos que requieren comunicación asíncrona o integración a través de red.

Las capacidades expuestas por un servidor se organizan en tres categorías: *prompts*, entendidos como plantillas parametrizables que orientan la interacción del modelo; *resources*, que proporcionan acceso controlado a información estructurada o semiestructurada (por ejemplo, contenido de archivos o registros de bases de datos); y *tools*, que representan funciones ejecutables invocables con parámetros para realizar operaciones concretas dentro del dominio del servidor.

2.8.4 Ejemplos de servidores MCP

Una vez descrita la arquitectura y las capacidades expuestas por MCP (*prompts*, *resources* y *tools*), se introducen a continuación diferentes ejemplos de servidores representativos por su aplicabilidad en entornos con requisitos de control del dato y operación offline. Estos servidores materializan la integración estandarizada de herramientas y fuentes de información, extendiendo el alcance funcional de los LLM hacia dominios operativos concretos sin recurrir a integraciones *ad hoc*.

MCP-Filesystem permite al LLM leer, escribir y navegar el sistema de archivos de forma controlada, con restricciones configurables de acceso. Esta capacidad habilita flujos de trabajo donde el agente procesa documentos locales, genera informes o gestiona archivos de configuración bajo supervisión.

SQLite MCP Server proporciona acceso a bases de datos relacionales SQLite a través de MCP, exponiendo operaciones de lectura y escritura, inspección de esquemas y ejecución de consultas SQL parametrizables. Puede configurarse en modo solo lectura cuando se desea evitar modificaciones sobre datos operativos.

GhidraMCP integra el framework de ingeniería inversa Ghidra dentro del ecosistema MCP, haciendo accesibles sus capacidades a modelos de lenguaje en tareas de ingeniería inversa [15]. Entre las funciones expuestas se encuentran la decompilación de binarios a pseudocódigo legible, el análisis de estructuras de datos y funciones, el renombrado asistido de símbolos a partir de información semántica, el trazado de referencias cruzadas y la generación de grafos de llamadas, así como la identificación de patrones criptográficos. Esta integración permite que agentes basados en LLM interactúen de forma automatizada con proyectos de análisis binario, apoyando tareas de reversing y análisis de malware.

Git MCP Server facilita el acceso a repositorios Git, habilitando operaciones de lectura, búsqueda en el histórico, comparación entre commits (diff) y navegación por la estructura del repositorio. En entornos de desarrollo con requisitos de control y auditoría, permite que el agente asista en la revisión de cambios, generación de resúmenes de versiones, identificación de modificaciones relevantes y apoyo a la documentación de la evolución del software, sin exponer el repositorio fuera del perímetro autorizado.

Integración con herramientas de gestión y conocimiento: En escenarios donde se requiere automatizar flujos de trabajo de documentación y gestión de incidencias, pueden emplearse servidores MCP que expongan capacidades de plataformas como Confluence y Jira (u otras herramientas equivalentes). Estas integraciones permiten, bajo control de permisos, operaciones como búsqueda y recuperación de páginas, creación y actualización de documentación, consulta y actualización de tickets, vinculación de incidencias con evidencias o artefactos técnicos, y la generación de resúmenes de estado. En un entorno corporativo o institucional, este patrón habilita flujos de trabajo reproducibles en los que el agente consolida información desde repositorios internos, genera borradores de documentación y sincroniza resultados en sistemas de gestión, manteniendo trazabilidad de acciones y aplicando controles de acceso acordes a la clasificación de la información.

2.9 Retos para la adopción de IA offline en Defensa

La implantación de sistemas de IA en entornos de Defensa offline presenta retos específicos que trascienden las dificultades técnicas habituales de un proyecto TIC convencional. La concurrencia de requisitos estrictos de seguridad, limitaciones de conectividad, condicionantes presupuestarios y obligaciones de cumplimiento configura un escenario en el que el diseño, el despliegue y la operación deben abordarse de forma integral y coordinada. En este contexto, resulta esencial tener en cuenta tanto el rendimiento del modelo como la capacidad de mantener el sistema a lo largo de su ciclo de vida bajo procedimientos controlados, auditables y reproducibles.

El análisis que sigue identifica y clasifica los principales retos asociados, proponiendo enfoques prácticos orientados a su gestión.

2.9.1 Retos tecnológicos

Los retos tecnológicos en despliegues de IA offline abarcan desde la selección y provisión de infraestructura hasta la operación y mantenimiento del sistema a lo largo de todo su ciclo de vida, en un contexto en el que la actualización remota y el soporte directo del proveedor suelen estar muy limitados.

Requisitos de hardware especializado

El uso de modelos de lenguaje de gran capacidad sigue dependiendo, en la práctica, de GPUs de gama alta. El coste de adquisición, la disponibilidad en el mercado y las necesidades asociadas de

espacio, potencia eléctrica y refrigeración constituyen barreras relevantes, especialmente cuando se pretende dotar de capacidades de IA a varios centros o unidades distribuidas geográficamente. En redes clasificadas, estas dificultades se ven incrementadas por requisitos adicionales de seguridad física y electromagnética.

Gestión del ciclo de vida en entornos desconectados

La ausencia de conectividad externa altera de forma notable la gestión del ciclo de vida de los sistemas de inteligencia artificial. Cada fase presenta condicionantes específicos que obligan a establecer procedimientos propios, planificaciones más conservadoras y un uso cuidadoso de los recursos disponibles con el fin de garantizar la continuidad operativa en redes aisladas.

En la fase de selección y evaluación, los criterios de análisis deben incorporar, además de las métricas de rendimiento habituales, aspectos como la trazabilidad y el origen de los datos de entrenamiento, la disponibilidad de código fuente y pesos para permitir auditorías técnicas, la identificación de posibles sesgos o vulnerabilidades y la procedencia del modelo en el contexto geopolítico vigente.

El despliegue inicial exige una preparación detallada antes de la instalación en el entorno aislado. Las herramientas de gestión de paquetes (pip, npm, Docker) asumen por defecto acceso a repositorios remotos, por lo que deben emplearse técnicas que posibiliten su uso en local. Los medios de transferencia han de ajustarse a los procedimientos de seguridad establecidos de forma que la transferencia de software hacia la red desconectada quede adecuadamente documentada y controlada.

La actualización de sistemas constituye uno de los puntos más delicados. La distribución de nuevas versiones de modelos mediante soportes físicos introduce una latencia operativa significativa frente a esquemas basados en servicios en la nube. Un LLM en formato cuantizado puede alcanzar fácilmente decenas de gigabytes, lo que obliga a establecer procedimientos formales que incluyan la verificación de integridad mediante algoritmos criptográficos, la validación de firmas digitales, el análisis de seguridad de los soportes empleados y la realización de pruebas en un entorno de preproducción antes de su puesta en servicio.

Por último, la retirada y sustitución de sistemas debe contemplar el borrado seguro de modelos, conjuntos de datos y registros asociados conforme a los procedimientos de clasificación aplicables, así como la documentación de las lecciones aprendidas y la migración ordenada de usuarios y procesos dependientes hacia la nueva solución. Una gestión rigurosa de esta fase es clave para evitar la coexistencia innecesaria de versiones obsoletas en redes sensibles y para mantener la coherencia del marco de seguridad, cumplimiento y gobernanza tecnológica de la organización.

Obsolescencia tecnológica acelerada

El ritmo de avance en el campo de los LLM es significativamente superior al de otras tecnologías de la ingeniería del software. Modelos que se consideran referencia en un momento determinado pueden quedar desplazados en pocos meses por nuevas arquitecturas, técnicas de entrenamiento o conjuntos de datos mejorados. Esta dinámica genera una problemática constante entre la estabilidad operativa, que favorece mantener soluciones consolidadas, y la conveniencia de incorporar versiones que ofrecen mejoras sustanciales de capacidad, eficiencia o seguridad.

En este contexto, resulta necesario establecer criterios explícitos para determinar cuándo una actualización está justificada. Entre los factores a considerar se incluyen: mejoras demostrables en tareas directamente vinculadas a la misión, sustentadas en evaluaciones reproducibles; corrección de vulnerabilidades, fallos o degradaciones identificadas en el modelo o en sus dependencias; incorporación de capacidades relevantes (soporte de idiomas, modalidades, dominios técnicos, etc.) con impacto operativo; y reducciones significativas de consumo de recursos (memoria, latencia, energía) que

permitan ampliar el despliegue o mejorar la disponibilidad del servicio. A partir de estos criterios, puede definirse una política de versiones que distinga entre actualizaciones menores (parches, ajustes incrementales y mantenimiento de dependencias) y mayores (cambio de modelo, arquitectura o pipeline), con una periodicidad alineada con la criticidad del sistema, el esfuerzo logístico y la capacidad de validación de la organización, evitando tanto la obsolescencia tecnológica como la actualización continua sin evidencia objetiva de mejoría.

Limitaciones de rendimiento y escalabilidad

En entornos desconectados, el rendimiento y la capacidad de servicio de los sistemas de IA quedan acotados fundamentalmente por el hardware disponible, sin posibilidad de recurrir a recursos elásticos en la nube para absorber puntas de demanda. En periodos de carga elevada, esta limitación se traduce en incrementos de latencia, colas de espera y, en última instancia, en una reducción efectiva de la disponibilidad del servicio para determinados colectivos de usuarios.

La gestión de estas restricciones requiere una planificación específica. En primer lugar, el dimensionamiento de la infraestructura debe incorporar márgenes para escenarios de máxima carga, en función de perfiles de uso y prioridades operativas. En segundo lugar, resulta recomendable establecer mecanismos de gestión de colas y políticas de priorización que garanticen el acceso preferente de usuarios o procesos críticos cuando la capacidad se vea comprometida. En tercer lugar, la utilización sistemática de técnicas de optimización, como la cuantización de modelos, la selección de arquitecturas más eficientes o el uso de variantes de menor tamaño para tareas menos exigentes, permite aumentar el rendimiento efectivo sobre la misma base hardware.

Cuando la infraestructura lo permite, la distribución de carga entre varias instancias y el escalado horizontal pueden mejorar de forma significativa la capacidad de servicio. Sin embargo, en contextos offline este escalado implica la adquisición, certificación, instalación y validación de nuevos equipos, con plazos de aprovisionamiento que pueden resultar elevados.

2.9.2 Retos organizativos

Gestión del cambio y resistencia

La introducción de asistentes de IA puede generar preocupación entre el personal, particularmente cuando se percibe como un posible sustituto de tareas o funciones existentes. Resulta necesario articular una estrategia de gestión del cambio que presente la IA como una herramienta de apoyo y aumento de capacidades y no como un mecanismo de sustitución.

Desarrollo de competencias

El uso eficaz de sistemas de IA requiere capacidades distribuidas en diferentes niveles de la organización. Los usuarios finales necesitan formación básica en formulación de consultas, interpretación crítica de las respuestas, identificación de limitaciones del sistema y criterios para decidir cuándo procede utilizar la herramienta y cuándo es necesaria una revisión adicional. Paralelamente, el personal técnico debe adquirir competencias específicas en administración de infraestructuras con GPU, operación de plataformas de contenedores, gestión de modelos y bases de datos vectoriales, así como en monitorización y diagnóstico de sistemas de IA.

Estos perfiles técnicos especializados son escasos en el mercado, lo que hace necesario adoptar estrategias combinadas de formación interna, colaboración con universidades y centros de investigación, y contratación selectiva de personal especializado.

Adaptación de procesos organizativos

La incorporación de IA en una organización no debería consistir únicamente en superponer nuevas herramientas sobre los procedimientos ya existentes, concebidos para un contexto diferente. Desde una perspectiva de ingeniería de procesos, resulta necesario rediseñar los flujos de trabajo, identificando de

forma explícita qué actividades se automatizan parcial o totalmente, qué tareas permanecen bajo responsabilidad humana y en qué puntos deben establecerse controles y validaciones obligatorias.

2.9.3 Retos económicos

Estructura de costes e inversión inicial

La implantación de capacidades de IA offline implica una estructura de costes distinta a la de los servicios en la nube y concentrada, en gran medida, en una inversión inicial elevada. Esta inversión abarca, como mínimo, cuatro ámbitos: hardware (GPUs, servidores, almacenamiento, electrónica de red), infraestructura física (espacio en centro de datos, adecuación de salas, sistemas de refrigeración acordes con la densidad térmica de las GPUs, capacidad eléctrica y sistemas de respaldo), software (licencias cuando proceda, herramientas de orquestación y monitorización, aunque el uso de soluciones de código abierto puede atenuar este componente), y personal (contratación, especialización y retención de perfiles técnicos cualificados).

El coste total de propiedad del sistema, necesario para comparar de forma objetiva arquitecturas offline, debe incorporar además del coste inicial, los costes recurrentes y de renovación: ciclos de actualización de hardware para mantener niveles de rendimiento aceptables, contratos de mantenimiento y soporte de la infraestructura, formación continua del personal técnico y de usuario, gastos operativos de energía, refrigeración y espacio en centro de datos, así como el coste de oportunidad asociado a la inmovilización de recursos frente a alternativas basadas en servicios externos.

Cuantificación de beneficios intangibles

Los beneficios principales de la IA offline en contextos de Defensa como la soberanía tecnológica, el cumplimiento normativo o la protección de información clasificada, presentan un carácter fundamentalmente cualitativo y resultan difícilmente cuantificables mediante indicadores financieros tradicionales. Las metodologías de justificación deben incorporar valoración de riesgos evitados (coste potencial de una brecha de seguridad, sanciones por incumplimiento normativo), capacidades habilitadas (misiones que resultarían inviables sin IA soberana), y comparación con alternativas (coste de no disponer de capacidades de IA, dependencia de proveedores extranjeros). A ello se añade la habilitación de capacidades operativas concretas, como tareas de análisis, simulación o apoyo a la decisión, que resultarían inviables sin disponer de una plataforma de IA bajo control propio.

2.9.4 Retos normativos y de cumplimiento

La adopción de soluciones de inteligencia artificial en entornos de defensa no se limita únicamente a consideraciones técnicas y organizativas, sino que se desarrolla dentro de un marco de cumplimiento que integra requisitos de ciberseguridad, protección de datos, regulación específica aplicable a sistemas de IA, disposiciones relativas al tratamiento de información clasificada, entre otros. Estos condicionantes impactan de forma directa en el diseño arquitectónico y en los procesos de operación, auditoría, gestión de cambios y actualización de modelos, al exigir garantías verificables sobre control del dato, trazabilidad del procesamiento y gobernanza del ciclo de vida del sistema. Las principales referencias normativas son:

- **Esquema Nacional de Seguridad (ENS).** El Real Decreto 311/2022 [47] establece la política de seguridad para la protección de la información en el sector público español. Aunque el ENS no introduce requisitos específicos para la IA, su carácter tecnológicamente neutro obliga a que las soluciones de IA, incluidas las basadas en modelos generativos, se sometan a los principios de gestión de riesgos, seguridad por defecto, control de accesos, trazabilidad y protección de la información, de acuerdo con la categoría de seguridad asignada.

- **Reglamento Europeo de Inteligencia Artificial.** El Reglamento (UE) 2024/1689 [48], en vigor desde agosto de 2024, excluye explícitamente en su ámbito de aplicación los sistemas utilizados exclusivamente para propósitos militares, de defensa o seguridad nacional.
- **Normativa de información clasificada NATO.** La política de seguridad de la OTAN C-M(2002)49 [49] no regula de forma específica la inteligencia artificial, sino que establece requisitos generales para el tratamiento, almacenamiento y transmisión de información clasificada, independientemente de la tecnología empleada. En consecuencia, los sistemas basados en IA se integran dentro de este marco como sistemas de información, quedando sujetos a las mismas restricciones de clasificación y de uso de redes acreditadas. Esto implica que cualquier sistema de IA que procese información NATO SECRET o superior debe desplegarse en infraestructuras autorizadas y desconectadas de redes externas, siendo compatibles únicamente soluciones *on-premise* u *offline*.

Un análisis exhaustivo de este marco, incluyendo la interacción entre normativa nacional en materia de seguridad de la información, regulación europea sobre IA y requisitos específicos asociados a información clasificada, excede el alcance de la presente memoria y podría constituir un trabajo monográfico. En consecuencia, el presente estudio se limita a identificar estos condicionantes como requisitos de contexto y a considerarlos en las decisiones de diseño y despliegue, en particular en la justificación de soluciones *offline* orientadas a maximizar el control organizativo, la soberanía tecnológica y la verificabilidad de la operación en entornos con restricciones elevadas.

2.9.5 Retos de seguridad específicos de IA

Aunque los sistemas de IA *offline* reducen la superficie de exposición frente a amenazas externas, no eliminan los riesgos específicos asociados al uso de modelos de lenguaje. Muchos de estos riesgos se originan en la propia forma en que estos modelos procesan instrucciones y datos, y deben tratarse como una nueva capa de seguridad a gestionar, adicional a la ciberseguridad clásica de redes y sistemas [50].

Prompt injection

Prompt injection consiste en manipular el comportamiento del modelo mediante entradas diseñadas para alterar sus instrucciones internas. Este tipo de ataque puede realizarse tanto en las consultas directas del usuario como en datos que el sistema consume de forma rutinaria como, por ejemplo, documentos, campos de bases de datos o resultados de herramientas. El resultado potencial incluye la revelación de información sensible, la ejecución de acciones no previstas o la desactivación de controles lógicos definidos en el propio sistema. [51]

Las estrategias de mitigación incluyen validación y sanitización de entradas antes de su procesamiento, separación clara entre instrucciones de alto nivel (políticas, reglas del sistema) y datos de usuario, aplicación estricta del principio de mínimo privilegio en las herramientas que el modelo puede invocar y la monitorización de patrones de uso que permitan detectar comportamientos anómalos.

Data poisoning

El data poisoning se produce cuando un atacante consigue introducir datos manipulados en los conjuntos utilizados para entrenar, ajustar o alimentar al sistema, con el propósito de inducir comportamientos erróneos o crear puertas traseras difíciles de identificar. Este riesgo no se limita al entrenamiento inicial de grandes modelos sino que también afecta a fases posteriores, como el *fine-tuning* interno o la actualización de bases documentales en sistemas RAG.

Para sistemas RAG, donde la base de conocimiento se actualiza continuamente, el riesgo de envenenamiento requiere atención específica. Las mitigaciones incluyen validación de procedencia de todos los documentos incorporados, control de versiones que permita identificar y revertir cambios

problemáticos, detección de anomalías en el contenido mediante análisis automatizado, y controles de acceso estrictos que limiten quién puede modificar la base de conocimiento [52].

Jailbreaking

Las técnicas de jailbreaking constituyen variantes específicas de prompt injection diseñadas para eludir los mecanismos de seguridad y alineación incorporados en los modelos durante su entrenamiento. Mediante prompts cuidadosamente contruidos, es posible inducir al modelo a generar contenido que normalmente rechazaría o a ignorar restricciones de uso.

La defensa contra jailbreaking requiere actualización continua de los modelos utilizados conforme se descubren nuevas técnicas de ataque, incorporación de ejemplos de ataques en los procesos de ajuste y evaluación, y el diseño de capas de control adicionales (por ejemplo, filtros de salida o validadores externos) que supervisen el contenido generado cuando la aplicación opera en dominios sensibles [53].

Alucinaciones y desinformación

Se denomina alucinación a la generación de respuestas que contienen información incorrecta presentada como si fuera cierta. Este fenómeno puede pasar desapercibido para el usuario si no dispone de conocimientos previos o de fuentes independientes con las que contrastar el contenido, y en contextos operativos podría conducir a decisiones apoyadas en datos erróneos.

Las arquitecturas RAG disminuyen este riesgo al apoyar las respuestas en documentos procedentes de repositorios controlados, pero no lo eliminan por completo: el sistema puede interpretar mal un texto, combinar de forma inadecuada fragmentos de diferentes fuentes o extraer conclusiones que no se desprenden de los originales. Por este motivo, los flujos de trabajo críticos deben incluir verificación humana obligatoria antes de utilizar la información generada, prestando especial atención a la comprobación independiente de hechos concretos como cifras, fechas, ubicaciones o identificadores.

Medidas de mitigación

Para mitigar los riesgos descritos, resulta necesario aplicar un conjunto coherente de medidas técnicas y organizativas. En primer lugar, el sistema debe diseñarse conforme al principio de mínimo privilegio, restringiendo sus capacidades a lo estrictamente necesario para la misión prevista y evitando habilitar funciones no requeridas, como la ejecución de código o el acceso directo a determinados sistemas. Asimismo, es recomendable establecer mecanismos de validación tanto sobre las entradas como sobre las salidas, de modo que se filtren instrucciones potencialmente maliciosas y se detecte contenido inapropiado o sospechoso antes de que tenga efectos operativos.

La explotación segura del sistema exige también un registro detallado de las interacciones y acciones realizadas, que permita auditorías posteriores y facilite la detección de patrones de uso anómalos. Siempre que sea posible, la ejecución de herramientas y componentes auxiliares debería realizarse en entornos aislados (*sandbox*), de forma que un fallo o compromiso quede contenido y no afecte al resto de la infraestructura.

Finalmente, en aquellos casos en que las respuestas del sistema puedan influir en decisiones sensibles o difundirse de forma amplia, debe mantenerse una supervisión humana explícita, de manera que la responsabilidad última sobre la interpretación y uso de la información generada permanezca claramente atribuida.

3 DISEÑO DE LA SOLUCIÓN

El presente capítulo describe el diseño de la plataforma de inteligencia artificial offline propuesta para entornos desconectados a partir del análisis del estado del arte realizado en el capítulo anterior. El diseño se articula en torno a una arquitectura de microservicios desplegada sobre infraestructura local, con Open WebUI como componente central de interacción y Ollama como motor de inferencia de modelos de lenguaje.

El diseño cubre tres escenarios de despliegue: puesto individual aislado, grupo de trabajo en red local y arquitectura global centralizada. La implementación práctica, descrita en el capítulo siguiente, se limita al puesto individual y al escenario de grupo de trabajo por ser el caso de uso más frecuente.

3.1 Principios de diseño

La plataforma adopta una arquitectura de microservicios como principio fundamental de diseño. Este enfoque descompone la aplicación en un conjunto de servicios autónomos y débilmente acoplados, cada uno responsable de una funcionalidad específica y desplegable de forma independiente. Cada servicio gestiona sus propios datos y se comunica a través de APIs bien definidas.

Sobre la arquitectura de microservicios se superponen una serie de principios transversales que garantizan el cumplimiento de los requisitos introducidos en la sección anterior.

El principio de operación completamente offline constituye el requisito fundacional del sistema. Todos los componentes deben ser capaces de funcionar sin ningún tipo de conectividad a redes externas, incluyendo la fase de instalación inicial. Esto implica que las dependencias de software, los modelos de lenguaje, los modelos de embeddings y cualquier otro recurso necesario deben poder transferirse mediante medios físicos y desplegarse sin acceso a repositorios en línea.

El principio de trazabilidad completa exige que toda interacción con el sistema quede registrada de forma auditable. Cada consulta realizada, cada documento procesado y cada respuesta generada deben asociarse a un usuario identificado, una marca temporal precisa y los parámetros de configuración empleados.

El principio de transparencia y auditabilidad determina la preferencia por componentes de código abierto frente a soluciones propietarias. El uso de software cuyo código fuente es públicamente accesible permite su auditoría por equipos de seguridad de la organización, garantiza la ausencia de funcionalidades ocultas y elimina la dependencia de proveedores externos para el mantenimiento a largo plazo.

El principio de mínimo privilegio establece que cada componente del sistema debe operar únicamente con los permisos estrictamente necesarios para cumplir su función. Los servicios se ejecutan

con usuarios no privilegiados y el acceso a recursos del sistema operativo se restringe mediante contenedores (*Docker*).

3.2 Requisitos del sistema

Los requisitos del sistema se derivan del análisis realizado en el estado del arte y de los principios de diseño establecidos. Se organizan en tres categorías:

- **Funcionales:** interacción conversacional mediante interfaz web, consulta semántica sobre corpus documental (RAG), traducción automática multilingüe offline, generación de imágenes, asistencia a programación en IDE y extensibilidad mediante MCP.
- **Gestión y seguridad:** gestión de usuarios con roles diferenciados, registro de auditoría de interacciones y operación en red aislada.
- **Capacidad:** soporte entre 5-10 usuarios concurrentes.

3.3 Escenarios de despliegue de la solución

Se proponen tres arquitecturas de despliegue, cada una optimizada para diferentes escenarios de uso, niveles de seguridad y capacidades de procesamiento.

3.3.1 Escenario de despliegue monopuesto

La arquitectura individual está diseñada para escenarios que requieren el máximo aislamiento, donde cada usuario dispone de un sistema completamente independiente sin conexión de red y ejecutado en un equipo sin GPU dedicada. Esta configuración resulta idónea en entornos donde la exfiltración remota debe ser técnicamente inviable.

Ventajas: Esta arquitectura proporciona máximo aislamiento físico, inviabilidad de exfiltración de datos por red (cada sistema opera como una isla independiente), y resulta ideal para información de máxima clasificación.

Limitaciones: modelos del orden de 3B de parámetros, no permite colaboración entre usuarios, las actualizaciones requieren acceso físico al equipo, y el coste de administración resulta mayor al requerir intervención individual en cada puesto.

La Tabla 3-1 muestra las especificaciones técnicas de la arquitectura individual (puesto aislado).

Característica	Especificación
Hardware	CPU Intel i7/i9, 32 GB RAM, SSD 512 GB, sin tarjeta de red
Modelos	Hasta 3B parámetros (cuantización Q4)
Componentes	LM Studio o Ollama, ChromaDB embebido
Usuarios	1 (operador individual)

Tabla 3-1 Especificaciones arquitectura individual (Fuente: Elaboración propia)

3.3.2 Escenario de despliegue en grupo de trabajo

La arquitectura de grupo de trabajo implementa un servidor con GPU compartido para un conjunto reducido de usuarios (entre 5 y 10) conectados mediante una red local aislada. Esta configuración proporciona un equilibrio entre capacidades del modelo y requisitos de seguridad.

Ventajas: esta arquitectura ofrece mayor capacidad del modelo, respuestas más rápidas gracias a la aceleración GPU, posibilidad de compartir bases de conocimiento RAG entre usuarios, administración centralizada y mejor relación coste/capacidad.

Limitaciones: requiere infraestructura de red local, lo que incrementa la superficie de ataque, constituye un único punto de fallo y precisa gestión de usuarios y permisos.

La Tabla 3-2 muestra las especificaciones técnicas de la arquitectura de grupo de trabajo.

Característica	Especificación
Hardware	CPU multinúcleo, 64 GB RAM, GPU 24 GB VRAM, SSD 2 TB
Modelos	Hasta 27B parámetros (cuantización Q4)
Componentes	Ollama, Open WebUI, ChromaDB, LibreTranslate, Stable Diffusion
Usuarios	5-10 concurrentes
Red	LAN aislada, sin conectividad externa

Tabla 3-2 Especificaciones de la arquitectura de grupo de trabajo (Fuente: Elaboración propia)

3.3.3 Arquitectura global/centralizada

Infraestructura centralizada orientada a organizaciones de gran tamaño, basada en un clúster de servidores con múltiples GPUs de alto rendimiento, capacidad para modelos de más de 70B parámetros, balanceo de carga y mecanismos de alta disponibilidad. Esta arquitectura queda fuera del alcance de la implementación práctica del presente trabajo debido a su complejidad y coste.

3.4 Componentes principales de la solución

La implementación de una plataforma de IA offline requiere seleccionar componentes que cumplan simultáneamente requisitos de funcionalidad, rendimiento y operación en entornos desconectados.

3.4.1 Componentes seleccionados

Runtime de Inferencia LLM (Ollama)

Se evaluaron las soluciones Ollama, LM Studio y OnPrem.LLM, y se seleccionó Ollama por su equilibrio entre facilidad de uso y capacidades de integración, su API compatible con OpenAI, gestión dinámica de modelos, soporte nativo de embeddings y capacidad de operación offline completa.

Interfaz de Usuario (Open WebUI)

Se seleccionó Open WebUI por su integración nativa con Ollama, capacidades RAG incluidas, gestión de usuarios con roles y experiencia de usuario similar a ChatGPT, lo que reduce la curva de aprendizaje [24].

Base de datos vectorial (ChromaDB)

Se seleccionó ChromaDB por su integración nativa con Open WebUI, simplicidad de despliegue (no requiere servidor separado) y capacidad suficiente para corpus documentales de tamaño moderado (cientos a miles de documentos) [38].

Modelo de Embeddings (BGE-M3)

Se seleccionó BGE-M3 por su rendimiento en español y otros idiomas, capacidad de recuperación híbrida y tamaño manejable que permite su ejecución simultánea con el modelo de lenguaje principal.

La denominación "M3" hace referencia a sus tres características principales: Multi-Linguality (Multilingüe), Multi-Functionality (Multifuncionalidad) y Multi-Granularity (Multigranularidad).

Traducción (LibreTranslate)

Se seleccionó LibreTranslate por su capacidad de funcionamiento offline, API REST y soporte de más de 50 idiomas.

Generación de imágenes (Stable Diffusion)

Se seleccionó Stable Diffusion por su capacidad de generar imágenes en local, facilidad de integración y por ser de código abierto.

3.4.2 Modelos de lenguaje

Para entornos de puesto aislado (arquitectura individual, solo CPU) se seleccionaron los modelos incluidos en la Tabla 3-3.

Modelo	Parámetros	Contexto	Licencia
Phi-3 Mini	3.8B	16K	MIT
Qwen 2.5 3B	3B	32K	Apache 2.0
LLaMA 3.2 3B	3B	128K	Llama

Tabla 3-3 Modelos LLM seleccionados para entorno individual (Fuente: Elaboración propia)

Para entornos con GPU dedicada (arquitectura grupo de trabajo) se seleccionaron los modelos incluidos en la Tabla 3-4.

Modelo	Parámetros	Contexto	Licencia
Mistral 7B Instruct	7B	32K	Apache 2.0
LLaMA 3.1 8B	8B	128K	Llama
Qwen 2.5 7B	7B	128K	Apache 2.0
GPT-OSS	20B	128K	Apache 2.0
CodeLlama 7B	7B	100K	Llama
Gemma 3	27B	128K	Gemma License
Modelos específicos HuggingFace	Variable	Variable	Según modelo
DeepSeek-R1	14B	128K	Apache 2.0

Tabla 3-4 Modelos LLM para entorno de grupo de trabajo (Fuente: Elaboración propia)

3.5 Diseño de la arquitectura de grupo de trabajo

3.5.1 Arquitectura general

El sistema se organiza en tres capas diferenciadas. La capa de presentación (Open WebUI), servicios (Ollama, RAG, auxiliares) y datos (ChromaDB, almacenamiento). Los componentes se despliegan como contenedores *Docker* en una red interna dedicada.

3.5.2 Flujo de interacción

El flujo básico de interacción con el sistema consiste en que el usuario accede mediante un navegador web a Open WebUI que actúa como único punto de entrada expuesto. Open WebUI autentica al usuario, procesa la consulta, determina si requiere contexto RAG, invoca a Ollama para la generación de la respuesta y devuelve el resultado al usuario.

Los servicios auxiliares se invocan en función del tipo de tarea. Este flujo asegura que todas las peticiones pasan por un único punto de control, Open WebUI, que actúa como orquestador de los servicios internos y como responsable de garantizar la trazabilidad de las operaciones.

3.5.3 Comunicación entre servicios

Los servicios se comunican mediante HTTP(S). Únicamente Open WebUI expone puerto hacia la red departamental. Los servicios *backend* (Ollama, ChromaDB, LibreTranslate) son accesibles únicamente desde la red interna. La Figura 3-1 muestra la arquitectura general de la solución propuesta para el entorno de grupo de trabajo.

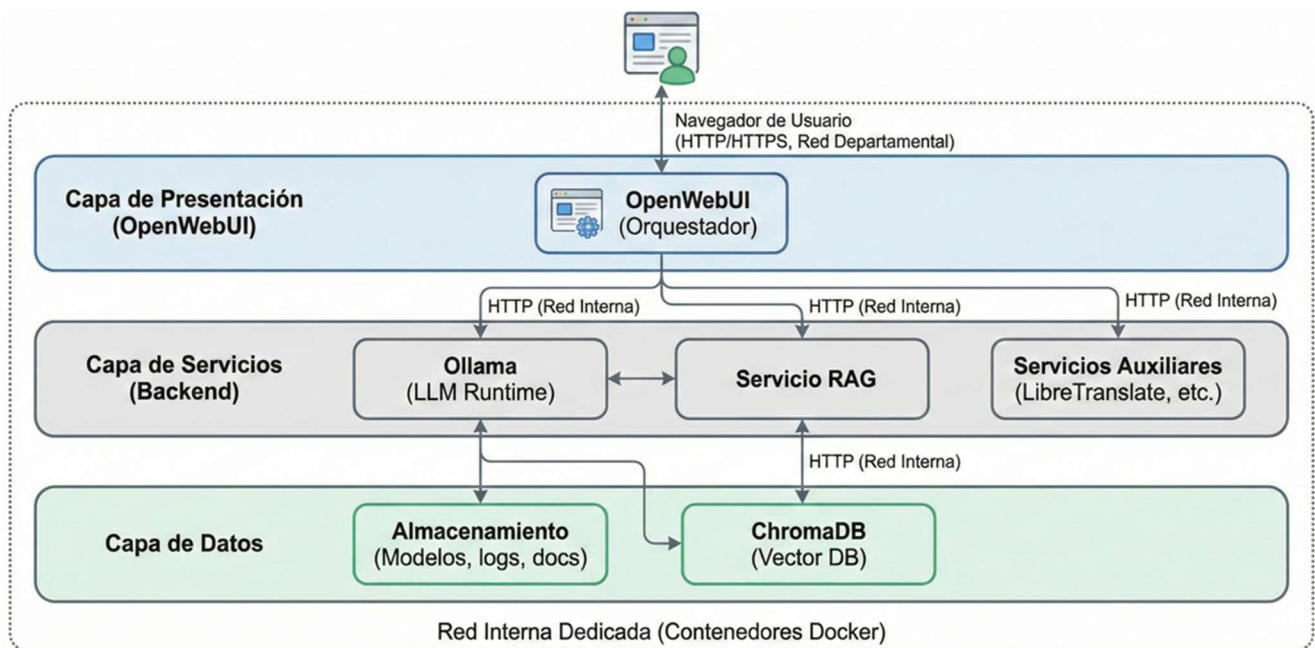


Figura 3-1 Arquitectura general de la solución de grupo de trabajo (Fuente: Elaboración propia)

3.6 Diseño de la interfaz de usuario

La interfaz de usuario constituye el punto de acceso principal al sistema, actuando como capa de intermediación entre los usuarios y los servicios de IA. Se despliega íntegramente dentro del perímetro de la organización y resulta accesible exclusivamente desde la red interna.

Incluye todas las funcionalidades nativas de Open WebUI como chat conversacional con modelos de lenguaje, gestión de documentos para su uso en flujos RAG, administración de usuarios y roles, panel de configuración del sistema, gestión de modelos disponibles y ajuste de parámetros de inferencia entre otras.

3.7 Diseño del subsistema RAG

El diseño del subsistema RAG integra procesos de ingesta y preparación de documentos, metodologías de fragmentación (*chunking*) y estrategias de recuperación de información. La arquitectura de la base de conocimiento se estructura a través de los componentes que se detallan a continuación.

3.7.1 Pipeline de ingesta

El proceso de ingesta comprende la extracción de texto desde documentos en formatos PDF, DOCX y TXT, su normalización, la segmentación en fragmentos con solapamiento, la generación de embeddings mediante el modelo bge-m3 y el almacenamiento de los vectores resultantes en ChromaDB junto con los metadatos de origen.

3.7.2 Estrategia de fragmentación

Se aplica una estrategia de segmentación recursiva que respeta la estructura lógica del documento, incluyendo secciones y párrafos. Se ha definido un tamaño objetivo de 512 tokens, dimensión que permite capturar unidades de información completas sin saturar la ventana de contexto de los modelos. Complementariamente, se establece un solapamiento del 10 % entre fragmentos consecutivos para evitar cortes abruptos en ideas clave, garantizando así la continuidad semántica necesaria durante la recuperación.

3.7.3 Estrategia de recuperación

Se emplea una estrategia de recuperación semántica densa, fundamentada en el cálculo de la similitud coseno sobre los vectores generados. Se ha seleccionado el modelo BGE-M3 debido a su capacidad avanzada de análisis semántico y soporte multilingües. El diseño actual prioriza la recuperación densa por su eficiencia computacional y su integración nativa con la base de datos vectorial ChromaDB.

3.7.4 Organización en colecciones

Los documentos se organizan en colecciones temáticas (normativa, procedimientos, informes, documentación técnica, ámbito). Cada colección constituye un espacio de búsqueda independiente. El usuario puede seleccionar colecciones activas antes de cada consulta.

La Figura 3-2 muestra el funcionamiento del subsistema RAG implementado en la solución.

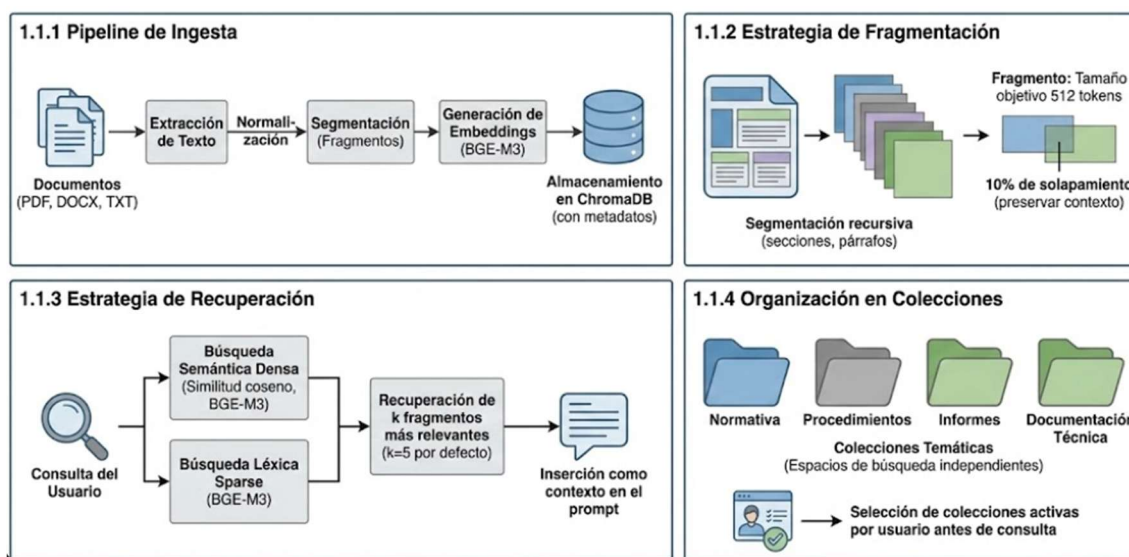


Figura 3-2 Funcionamiento RAG (Fuente: Elaboración propia)

3.8 Diseño de servicios auxiliares

3.8.1 Traducción

LibreTranslate se despliega como un microservicio independiente con interfaz REST, que expone *endpoints* para la traducción de texto, la detección automática de idioma y la consulta de los idiomas disponibles. La comunicación se realiza exclusivamente mediante peticiones HTTP(S) desde los servicios *backend* de Open WebUI [54].

3.8.2 Generación de imágenes

Stable Diffusion se despliega como servicio dedicado con interfaz REST, encargado de la generación de imágenes a partir de descripciones textuales. Open WebUI lo invoca de forma condicionada al tipo de solicitud del usuario, integrando las imágenes generadas en la conversación [55].

3.8.3 Asistencia a programación

La asistencia al desarrollo se implementa mediante la extensión Continue para Visual Studio Code, conectada al servidor local de Ollama. Esta integración proporciona chat contextual sobre el código presente en el editor, sugerencias de autocompletado y explicación de fragmentos, reutilizando el mismo modelo LLM que da servicio a la plataforma [56].

3.8.4 Integración MCP

La plataforma habilita arquitecturas de agentes, en las que el modelo de lenguaje orquesta en tiempo de ejecución y de forma controlada la invocación de herramientas externas a través de servidores MCP, manteniendo el aislamiento y la trazabilidad de todas las operaciones.

3.9 Casos de uso incluidos en la solución

Esta sección identifica los principales casos de uso contemplados en el diseño de la solución, en los que la inteligencia artificial offline aporta valor operativo significativo en entornos con restricciones de conectividad y seguridad.

3.9.1 Análisis y procesamiento de documentación

En el ámbito del análisis y procesamiento de documentación se identifican los siguientes casos de uso prioritarios.

Resumen automático de informes: Los modelos de lenguaje integrados en la plataforma pueden generar resúmenes ejecutivos a partir de informes extensos de inteligencia, operativos o administrativos. El sistema extrae los puntos clave, identifica entidades relevantes y estructura la información de forma concisa.

Extracción de información estructurada: Conversión de documentos no estructurados en información estructurada, como tablas o bases de datos, a partir de contenido textual. Esta capacidad resulta especialmente útil para procesar grandes volúmenes de documentación heredada.

Asistente de búsqueda documental: Sistema RAG sobre repositorios documentales que permite consultas en lenguaje natural. Los usuarios pueden formular preguntas sobre el contenido de miles de documentos y obtener respuestas contextualizadas con referencias explícitas.

3.9.2 Apoyo a la ciberseguridad

Los casos de uso más relevantes incluidos en el diseño de la solución en relación al apoyo a la ciberseguridad son los siguientes:

Análisis de logs y eventos: El sistema permite el procesamiento de grandes volúmenes de registros de seguridad con el fin de identificar patrones anómalos, correlacionar eventos dispersos y generar alertas contextualizadas que faciliten la labor de los analistas. Esta capacidad se apoya en las capacidades

de razonamiento del modelo de lenguaje, complementadas opcionalmente mediante RAG sobre documentación interna, procedimientos operativos o guías de respuesta a incidentes.

Generación asistida de informes de incidentes: El sistema proporciona apoyo a la elaboración de informes de incidentes de seguridad mediante capacidades de redacción asistida, estructuración de contenidos y respuesta a consultas específicas del analista. El uso de RAG permite fundamentar las respuestas y sugerencias en procedimientos organizativos previamente indexados, reforzando la coherencia y trazabilidad del informe resultante.

Asistente para análisis de malware: La integración con herramientas especializadas como Ghidra, a través del protocolo MCP, permite apoyar de forma asistida tareas de ingeniería inversa. El sistema facilita la interpretación del comportamiento del código analizado, la explicación de funciones y estructuras relevantes, y la identificación de técnicas y tácticas asociadas a marcos de referencia como MITRE ATT&CK, actuando siempre como apoyo al analista y bajo supervisión humana.

Revisión de código: Análisis estático de código fuente para identificar vulnerabilidades potenciales, malas prácticas de programación o patrones inseguros antes del despliegue en producción.

3.9.3 Formación y capacitación

Los casos de uso más relevantes incluidos en el diseño de la solución en relación al apoyo a la formación y capacitación son los siguientes:

Chat de formación: El sistema puede emplearse como herramienta de apoyo al aprendizaje mediante interacción conversacional, permitiendo a los usuarios formular consultas sobre procedimientos, normativa, doctrina y equipamiento a partir de repositorios documentales internos. Esta funcionalidad está orientada a resolver dudas de carácter básico e intermedio y a facilitar el acceso autónomo a la información, actuando como complemento a la formación reglada y sin sustituir la función de los instructores humanos.

Apoyo a la evaluación de conocimientos: La plataforma puede asistir en la preparación de materiales de evaluación mediante la generación de propuestas de preguntas, ejercicios y supuestos prácticos, así como en la elaboración de comentarios orientativos sobre respuestas abiertas proporcionadas por los alumnos.

3.10 Limitaciones y decisiones de diseño

El diseño presentado en este capítulo se ajusta a los objetivos definidos para el presente trabajo, priorizando una solución práctica y viable con recursos limitados. Por ese motivo, se han adoptado determinadas decisiones de diseño orientadas a simplificar la arquitectura y a acotar el alcance funcional de la plataforma. Esta sección identifica los aspectos que quedan fuera del alcance del diseño propuesto y expone las principales limitaciones asumidas, detallando las razones técnicas, operativas u organizativas.

A continuación, se describen de forma resumida las funcionalidades que no se han incluido en el diseño de la solución, junto con las razones técnicas, operativas u organizativas que justifican su exclusión.

Arquitectura centralizada y alta disponibilidad: No se diseña ni implementa una arquitectura global federada ni un sistema de alta disponibilidad. La incorporación de estos elementos requeriría infraestructuras de clúster, estrategias de replicación, balanceo de carga y recursos de operación continuada que exceden tanto el alcance como la naturaleza académica del proyecto.

Modelos de más de 70B parámetros: Los modelos de lenguaje de muy gran escala no se consideran en esta solución debido a sus elevados requisitos de hardware, especialmente en términos de capacidad de cómputo, memoria y consumo energético. La selección se ha centrado en modelos con tamaños

comprendidos entre 7B y 27B parámetros, que ofrecen un compromiso adecuado entre rendimiento, eficiencia y viabilidad operativa en entornos offline.

Entrenamiento completo de modelos: El proyecto no contempla el entrenamiento desde cero de un modelo de lenguaje. Esta actividad exige grandes volúmenes de datos, por lo que se limita al uso de técnicas de ajuste fino ligero (LoRA/QLoRA) aplicadas a modelos ya entrenados.

Certificación formal de seguridad: La plataforma no persigue procesos de certificación, no obstante, el diseño incorpora principios compatibles con buenas prácticas de seguridad.

Monitorización avanzada: El sistema utiliza únicamente logs locales y mecanismos de auditoría de Open WebUI y de los microservicios. Aunque la integración con plataformas de monitorización avanzada o sistemas SIEM sería técnicamente viable mediante agentes específicos, dicha funcionalidad no se ha implementado en el marco de este proyecto.

4 IMPLEMENTACIÓN Y RESULTADOS

Para la implementación y validación experimental de la solución propuesta se emplearon dos entornos locales de ejecución y un entorno adicional de supercomputación orientado a tareas de validación y pruebas de entrenamiento avanzado.

En primer lugar, las pruebas funcionales se llevaron a cabo sobre dos estaciones de trabajo locales, una de ellas sin GPU dedicada y otra equipada con una GPU NVIDIA RTX 4090. En estos entornos se desplegaron dos soluciones complementarias, que se describen a continuación.

- LM Studio, utilizado como entorno de pruebas para la ejecución local y comparativa de modelos de lenguaje.
- Ollama, configurado como entorno de ejecución principal dentro de la arquitectura propuesta, integrando el sistema RAG, los servicios auxiliares (traducción, generación de imágenes y código) y la interfaz de usuario Open WebUI, todo ello bajo un esquema de operación sin acceso a internet.

De forma complementaria, se empleó el sistema de supercomputación FinisTerra III, del CESGA, para abordar las tareas de validación y pruebas de entrenamiento y ajuste fino de modelos de lenguaje. Este entorno permitió evaluar la escalabilidad del proceso de personalización de modelos mediante técnicas LoRA/QLoRA, utilizando conjuntos de datos reducidos, específicos del dominio de la ciberseguridad.

4.1 Entornos de implementación

4.1.1 Estación de trabajo local (sin GPU dedicada)

La infraestructura local empleada se basa en una estación de trabajo de propósito general, cuyas especificaciones técnicas se detallan en la Tabla 4-1.

Componente	Especificación
CPU	Intel Core i7
Memoria RAM	32 GB DDR4
Almacenamiento	SSD NVMe 512 GB
Sistema operativo	Ubuntu 25.04

Tabla 4-1 Estación de trabajo local sin GPU (Fuente: Elaboración propia)

El sistema prescinde de tarjeta de red y de cualquier interfaz de comunicación externa. Los modelos LLM desplegados se limitan a tamaños del orden de 3.000 millones de parámetros, utilizando cuantización a 4 bits para optimizar la ejecución en CPU. En esta configuración, las velocidades de generación observadas oscilan entre 8 y 15 tokens por segundo, en función del modelo seleccionado y de la longitud del contexto, lo que resulta adecuado para un uso interactivo individual en entornos aislados.

El coste estimado de la estación de trabajo empleada, basada en componentes de gama media y sin aceleración mediante GPU, se sitúa en torno a los 1.000 euros. Este valor refuerza la viabilidad económica de la solución para despliegues individuales, especialmente en escenarios que requieren aislamiento operativo y control estricto de la infraestructura.

4.1.2 Estación de trabajo local (con GPU dedicada)

Para el despliegue completo del entorno *offline*, que integra la totalidad de los servicios de la plataforma, se empleó una estación de trabajo de altas prestaciones, cuya configuración técnica se recoge en la Tabla 4-2.

Componente	Especificación
GPU	NVIDIA RTX 4090 (24 GB VRAM)
CPU	AMD Ryzen 9 7950X (16 núcleos, 32 hilos)
Memoria RAM	64 GB DDR5
Almacenamiento	SSD NVMe 2 TB
Sistema operativo	Ubuntu 25.04 LTS

Tabla 4-2 Estación de trabajo local con GPU (Fuente: Elaboración propia)

Esta configuración permite ejecutar, en condiciones de inferencia local, modelos de lenguaje de hasta aproximadamente 14B parámetros mediante cuantización a 4 bits, manteniendo velocidades de generación superiores a 40 tokens por segundo, en función del modelo empleado y del tamaño del contexto utilizado.

El coste estimado de la estación de trabajo empleada, equipada con GPU dedicada y orientada a cargas intensivas de inferencia y ajuste de modelos de lenguaje, se sitúa en torno a los 4.000 euros. Este perfil de hardware permite evaluar capacidades avanzadas de inteligencia artificial en entornos controlados, con un coste significativamente inferior al asociado a infraestructuras de supercomputación o al uso continuado de servicios cloud equivalentes.

4.1.3 Infraestructura de supercomputación (CESGA)

Para las pruebas de entrenamiento y ajuste fino de modelos se empleó el superordenador FinisTerra III, ubicado en el CESGA. Este entorno de computación de altas prestaciones proporciona la capacidad de cómputo necesaria para abordar procesos de entrenamiento que exceden las posibilidades de una estación de trabajo local, permitiendo evaluar las posibilidades de *fine-tuning* de modelos LLM sobre conjuntos de datos de mayor tamaño.

4.2 Implementación del puesto individual aislado (sin GPU)

La arquitectura de puesto individual aislado está diseñada para escenarios que requieren el máximo aislamiento físico y una operación completamente desconectada. Esta sección describe el proceso de implementación sobre una estación de trabajo sin GPU dedicada, utilizando LM Studio como plataforma de inferencia.

4.2.1 LM Studio

LM Studio es una aplicación de escritorio multiplataforma que permite ejecutar modelos de lenguaje localmente sin dependencias externas. Su selección para el puesto individual se justifica por las siguientes características, que se enumeran a continuación.

- Interfaz gráfica intuitiva que no requiere conocimientos técnicos avanzados.
- Gestión integrada de modelos (descarga, importación, eliminación).
- Ejecución optimizada sobre CPU mediante *backend llama.cpp*.
- Funcionamiento completamente offline una vez instalado.
- Servidor local compatible con API OpenAI y mcp para integraciones.

Instalación de LM Studio

La instalación de LM Studio se realiza mediante la descarga y configuración de los binarios oficiales disponibles para sistemas Linux.

```
# Descarga de LM Studio
wget https://releases.lmstudio.ai/linux/LM-Studio-x86_64.AppImage

# Estable permisos de ejecución
chmod +x LM-Studio-x86_64.AppImage

# Ejecución
./LM-Studio-x86_64.AppImage
```

Una vez completada la instalación, LM Studio queda disponible como aplicación de escritorio. La Figura 4-1 muestra la interfaz principal de LM Studio tras su instalación.

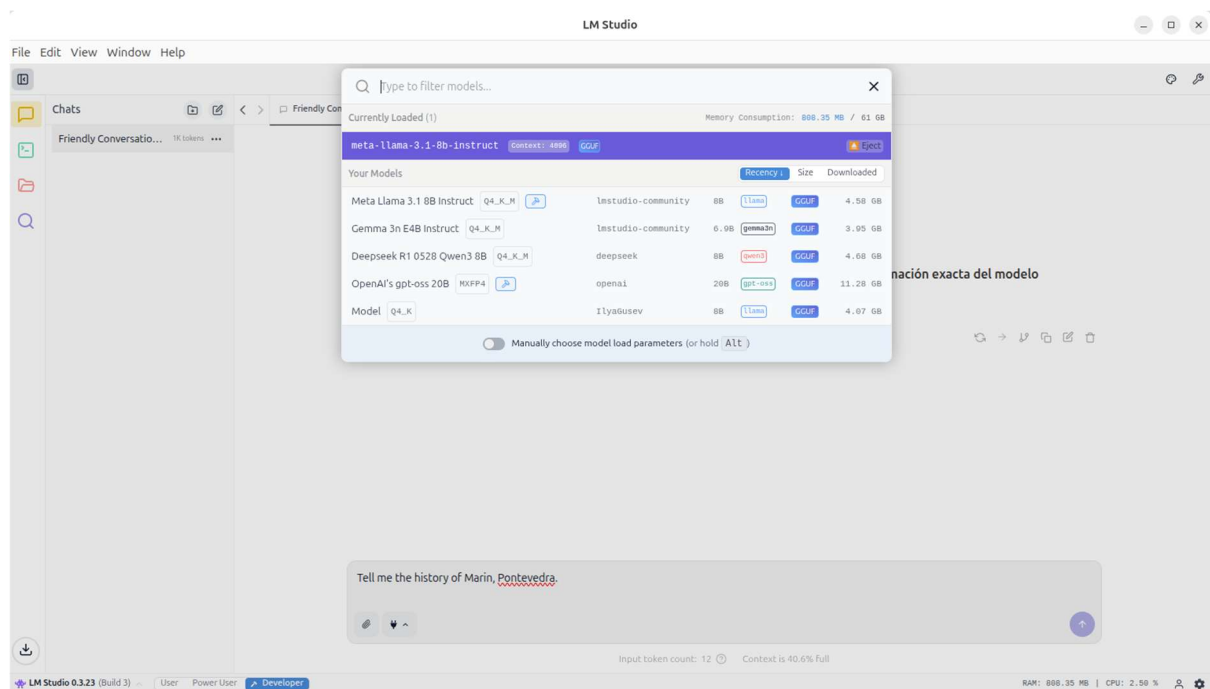


Figura 4-1 Funcionamiento de LM Studio (Fuente: Elaboración propia)

Adicionalmente, aunque LM Studio se utiliza principalmente como entorno de ejecución local orientado a un único puesto, se ha desarrollado, a modo de piloto y como ejemplo de mínima complejidad, una extensión que permite la interacción distribuida con los modelos mediante un servicio

de acceso web. Este prototipo se incorpora como prueba de concepto de que, con una capa de interfaz web desarrollada a medida y mecanismos adecuados de autenticación y control de acceso, LM Studio podría emplearse también como motor de inferencia en escenarios distintos al monopuesto, proporcionando capacidades LLM a usuarios remotos dentro de un perímetro controlado. En este sentido, sería viable diseñar una interfaz web específica para la organización, adaptada a sus flujos de trabajo, requisitos de seguridad y políticas internas. El desarrollo completo de una interfaz de este tipo, con nivel de madurez y robustez suficiente para su uso operativo, excede el alcance de este trabajo.

4.2.2 Modelos desplegados y rendimiento

Se desplegaron modelos optimizados para ejecución en CPU, priorizando aquellos con buen rendimiento en tareas de asistencia general y soporte en español. La Tabla 4-3 muestra los modelos desplegados y evaluados en el entorno individual.

Modelo	Parámetros	Tamaño	Caso de uso
Phi-3	3.8B	2.2 GB	Asistencia general y razonamiento
Qwen 2.5 Instruct	3B	1.8 GB	Multilingüe con buen soporte en español
Gemma-3n	4B	3.3 GB	Asistencia general, razonamiento y multilingüe.

Tabla 4-3 Modelos desplegados en entorno individual (Fuente: Elaboración propia)

Los modelos de 7B parámetros (Mistral 7B, Llama 3.1 8B) son ejecutables en este entorno pero con rendimiento limitado, generando latencias perceptibles que afectan a la experiencia de uso interactivo por lo que se recomienda su uso únicamente para tareas no urgentes o procesamiento por lotes.

4.2.3 Instalación de RAG

Para incorporar capacidades RAG en el puesto individual aislado se integró AnythingLLM, una plataforma modular orientada a la gestión de documentación y a la construcción de flujos RAG en entornos locales. Esta solución permite la ingesta de documentos, la generación de embeddings de forma local y la exposición de un servicio RAG compatible con distintos modelos de lenguaje.

El despliegue se realizó mediante contenedores Docker, garantizando el aislamiento del componente, la reproducibilidad de la configuración y la coherencia del entorno de ejecución en escenarios completamente desconectados. La Figura 4-2 muestra la interfaz de AnythingLLM desplegada en el entorno individual.

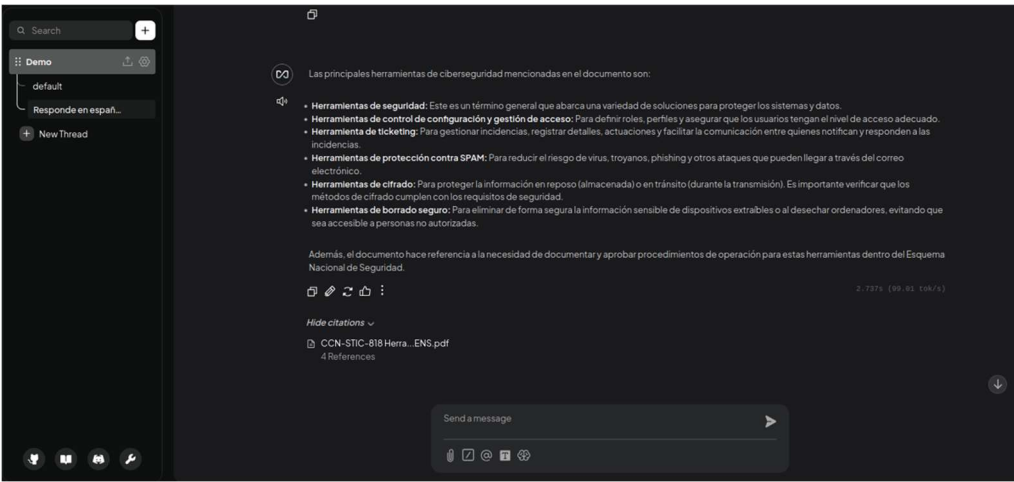


Figura 4-2 AnythingLLM en entorno individual (Fuente: Elaboración propia)

4.2.4 Resultados obtenidos

La validación del entorno monopuesto permitió verificar la operatividad completa del sistema en condiciones de aislamiento total. El equipo experimental, totalmente aislado, arrancó sin incidencias y ejecutó todas las funcionalidades previstas sin dependencia de servicios o recursos externos.

Los modelos de 3-4B parámetros (Phi-3 Mini, Qwen 2.5, Gemma-3n) alcanzaron velocidades de generación comprendidas entre 10 y 18 tokens por segundo, con *Time to First Token* (latencias de primera respuesta) de 1 a 4 segundos. Aunque estos valores son inferiores a los obtenidos en entornos con aceleración GPU, resultan adecuados para un uso interactivo no intensivo. Los modelos de 7B parámetros son técnicamente ejecutables pero con rendimiento limitado, del orden de 5 a 8 tokens por segundo, que afecta la experiencia de usuario.

La integración de AnythingLLM proporcionó capacidades RAG funcionales sobre un corpus de prueba, demostrando que es posible disponer de recuperación documental incluso en el entorno más restringido. Las pruebas de estabilidad durante ocho horas de operación continuada no revelaron degradación de rendimiento ni errores de memoria.

En conjunto, el entorno validado cumple el objetivo planteado para escenarios de máximo aislamiento, en los que los requisitos de seguridad y control priman sobre el rendimiento. La solución resulta adecuada para el procesamiento y consulta de información de forma completamente offline por parte de usuarios individuales.

4.3 Implementación del entorno de grupo de trabajo (con GPU)

La arquitectura de grupo de trabajo proporciona capacidades de inteligencia artificial compartidas para equipos reducidos, entre 5 y 15 usuarios concurrentes, mediante un servidor central dotado de GPU dedicada. En este escenario, una estación de trabajo equipada con una NVIDIA RTX 4090 actúa como nodo principal de cómputo, concentrando la ejecución de los modelos de lenguaje y los servicios asociados. Los usuarios acceden a la plataforma a través de la red interna, utilizando una interfaz web que funciona como punto único de acceso al sistema.

Esta sección describe el despliegue del entorno completo, abarcando desde el motor de inferencia hasta la capa de interfaz de usuario.

4.3.1 Despliegue del motor de inferencia (Ollama)

Ollama constituye el núcleo de inferencia de la plataforma, al encargarse de la carga, gestión y ejecución de los modelos de lenguaje aprovechando la aceleración por GPU.

Aunque la arquitectura general de la solución se basa en microservicios desplegados en contenedores *Docker*, el motor de inferencia se ejecuta directamente sobre el sistema anfitrión como servicio gestionado por *systemd*. Esta decisión responde a criterios de simplicidad operativa, acceso directo a la GPU y reducción de latencia, manteniendo el aislamiento lógico del sistema mediante el control de red y el uso de contenedores para el resto de componentes.

La instalación se realiza a partir de los binarios oficiales para sistemas Linux, configurando el servicio como demonio persistente.

Instalación de Ollama

A continuación se muestran los comandos mínimos para instalar Ollama en Linux, verificar su despliegue como servicio systemd y ejecutar el servidor en modo manual:

```
# Instalación de Ollama en Linux
curl -fsSL https://ollama.com/install.sh | sh
# Verificación del servicio
sudo systemctl status ollama
# Ejecución manual
ollama serve
```

Configuración de entorno seguro

Dado que el sistema está diseñado para operar en modo offline, se establecen las siguientes variables de entorno de seguridad y persistencia:

```
# Variables de entorno
export OLLAMA_HOST=127.0.0.1:11434
export OLLAMA_MODELS=/home/user/ollama_models
mkdir -p "$OLLAMA_MODELS"
```

Gestión de modelos LLM

Ollama incorpora un gestor integrado que permite importar, crear, listar y ejecutar modelos de lenguaje, tanto preentrenados como personalizados. La gestión de los modelos se realiza íntegramente en local, sin dependencia de conectividad externa. En los apartados siguientes se describen los comandos básicos empleados para la administración de modelos dentro de la plataforma.

```
# Descargar un modelo base (ejemplo:llama3)
ollama pull llama3
# Ejecutar el modelo en modo interactivo
ollama run llama3
# Eliminar modelos no necesarios
ollama rm llama3
```

Modelos disponibles

ollama list

NAME	ID	SIZE
qwen2.5:latest	845dbda0ea48	4.7 GB
codellama:latest	8fdf8f752f6e	3.8 GB
codestral:latest	0898a8b286d5	12 GB
llama3.2-vision:11b	6f2f9757ae97	7.8 GB
llama3:latest	365c0bd3c000	4.7 GB
mistral:latest	6577803aa9a0	4.4 GB
deepseek-r1:14b	c91df5ceaa77	7.3 GB
gemma2:9b	e5288a3df611	6.1 GB
devstral:latest	9bd74193e939	14 GB
F:Q4_K_M	df861406466e	4.7 GB

hf.co/mradermacher/DeepHat-V1-7B-GGU

... Otros modelos de prueba.

Al iniciarse, ollama run llama se inicia un diálogo interactivo:

Ejemplo de ejecución de llama3 con ollama

user@pc-ia:~\$ ollama run llama3

>>> ¿Dónde está la ciudad de Marín?

La ciudad de Marín se encuentra en España, específicamente en la comunidad autónoma de Galicia. Está ubicada en la provincia de Pontevedra, en el noroeste del país.

Marín es una ciudad con un rango administrativo de partido judicial y capital del municipio que lleva su nombre. Es conocida por ser un importante centro comercial y empresarial en la región, y también por ser un lugar turístico debido a sus playas, monumentos históricos y eventos culturales...

Para la medición del rendimiento de las consultas se emplea el modo verbose de Ollama, que proporciona métricas detalladas de tiempo y procesamiento, incluyendo la duración total de la inferencia, el tiempo de carga del modelo, el número de tokens procesados y generados, así como las tasas de evaluación y generación.

```
# Medida de rendimiento en el entorno propuesto
```

```
user@pc-ia:~$ ollama run mistral:latest --verbose
```

```
>>> Explica qué es un buffer overflow en 200 palabras
```

Un buffer overflow es una vulnerabilidad informática que ocurre cuando se escriben más datos en una memoria de almacenamiento temporal (buffer) que la capacidad permitida por su tamaño fijo. Este sobrecargue de datos puede provocar que los datos se escriban en áreas de memoria no intencionadas, lo que puede resultar en comportamientos inesperados del sistema operativo o del programa afectado, y potencialmente permitir ataques maliciosos, como la ejecución de código no autorizado.

En general, los buffers sobrecargados pueden provocar una variedad de problemas, incluyendo la corrupción de datos, la pérdida de información y el funcionamiento no deseado del software. Algunos tipos de buffer overflow son más peligrosos que otros: por ejemplo, cuando un atacante puede controlar los datos sobrecargados y ejecutar código arbitrario en la memoria del sistema, lo que permite a dicho atacante acceder a información confidencial, alterar el funcionamiento normal del sistema o instalar malware.

Debido a su potencial para causar daño significativo, los buffer overflows son una preocupación importante en seguridad informática y han sido una fuente común de ataques cibernéticos durante mucho tiempo. Las prácticas de programación segura, como la validación de datos, el uso de bibliotecas seguras y la verificación dinámica del código, pueden ayudar a evitar buffer overflows y otras vulnerabilidades en el software.

```
total duration:      2.629107426s
```

```
load duration:      17.177383ms
```

```
prompt eval count:  19 token(s)
```

```
prompt eval duration: 182.464704ms
```

```
prompt eval rate:   104.13 tokens/s
```

```
eval count:         387 token(s)
```

```
eval duration:      2.427871041s
```

```
eval rate:          159.40 tokens/
```

4.3.2 Despliegue de la interfaz de usuario (Open WebUI)

Open WebUI proporciona la interfaz web para el acceso multiusuario al sistema, integrando funcionalidades de chat, capacidades RAG y gestión básica de usuarios. Su despliegue se realiza mediante contenedores *Docker*, lo que permite encapsular la aplicación y sus dependencias, facilitar la portabilidad entre entornos y garantizar la reproducibilidad del sistema.

Despliegue con Docker

A continuación, se presenta un ejemplo de ejecución del contenedor que muestra el despliegue de Open WebUI con soporte de GPU y persistencia de datos.

```
# Ejemplo de ejecución con soporte GPU

docker run -d -p 3010:8080 --gpus all -v open-webui:/app/backend/data --name open-webui ghcr.io/open-webui/open-webui:cuda
```

Open WebUI proporciona la interfaz web para acceso multiusuario, integrando chat, RAG y gestión de usuarios. La Figura 4-3 muestra la interfaz de usuario de Open WebUI.

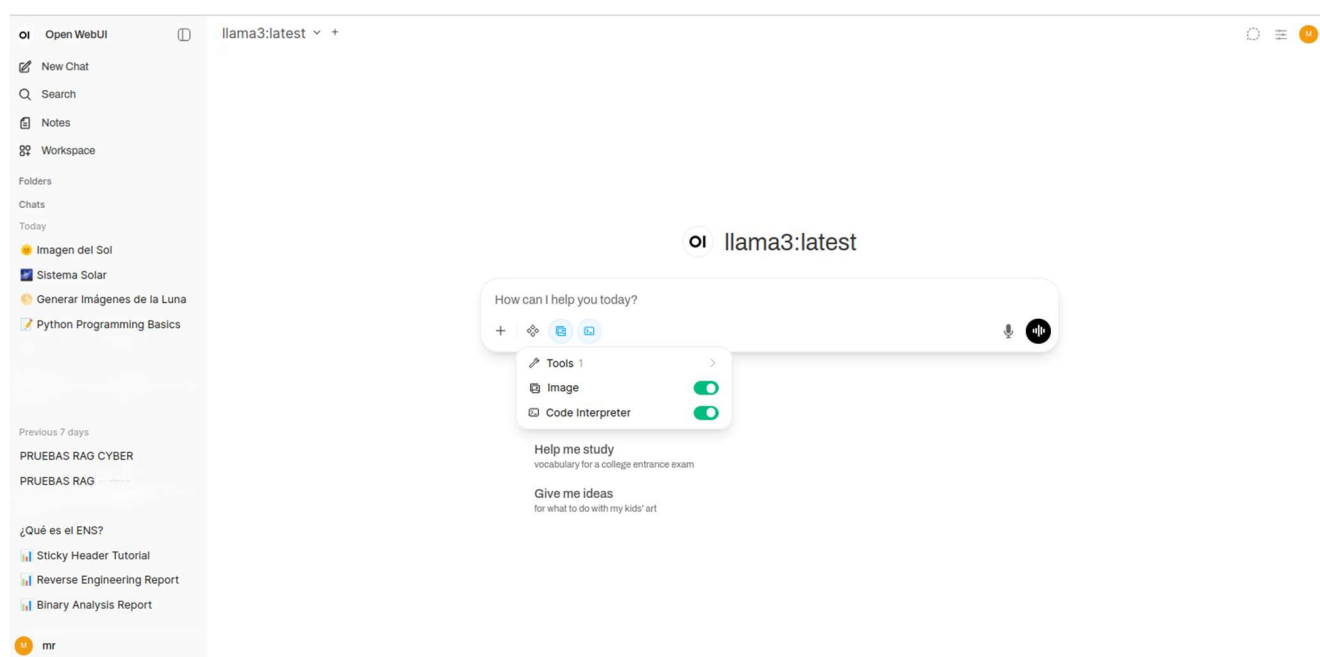


Figura 4-3 Interfaz de Open WebUI (Fuente: Elaboración propia)

Se configuraron tres roles con permisos diferenciados. El rol de usuario dispone de acceso a funcionalidades básicas, incluyendo chat conversacional, consulta RAG y exportación de conversaciones. El rol de usuario avanzado amplía estas capacidades al incorporar la gestión de colecciones RAG y el ajuste de determinados parámetros de funcionamiento. Por su parte, el rol de administrador cuenta con control completo sobre el sistema, incluyendo la gestión de usuarios, modelos y configuración global de la plataforma.

El acceso a la plataforma se realiza a través de un navegador web, mediante una interfaz expuesta en la red interna de la organización, con autenticación basada en credenciales locales. De forma opcional, el sistema podría integrarse con servicios corporativos de directorio, como LDAP, aunque esta funcionalidad no se ha implementado en el alcance del presente trabajo.

La interfaz de usuario ofrece una experiencia de uso comparable a la de plataformas comerciales de asistencia conversacional, facilitando la adopción por parte de usuarios no técnicos y reduciendo la curva de aprendizaje.

4.3.3 Modelos desplegados

Durante la realización del trabajo se evaluaron distintos LLM, seleccionados en función de su disponibilidad, capacidades técnicas y adecuación a los escenarios de uso definidos. Dado el rápido ritmo de evolución del ecosistema de modelos LLM, algunos de ellos fueron actualizados de manera periódica a lo largo del proceso experimental.

En la siguiente tabla se recogen los principales modelos analizados en el entorno de grupo de trabajo, junto con su tamaño aproximado y el caso de uso para el que fueron empleados. La evaluación se centró en aspectos como la calidad de generación, el razonamiento, el soporte multilingüe, la programación asistida y, en determinados casos, la capacidad multimodal. La Tabla 4-4 muestra los modelos evaluados en el entorno de grupo de trabajo.

Modelo	Parámetros	Tamaño	Caso de uso
Mistral	7B	4.4 GB	Modelo principal, asistencia general
LLaMA 3.1	8B	4.7 GB	Razonamiento complejo, instrucciones
Qwen 2.5	7B	4.7 GB	Multilingüe, soporte español
CodeLlama	7B	3.8 GB	Generación y revisión de código
Codestral	22B	12 GB	Programación avanzada (Mistral)
Devstral	24B	14 GB	Desarrollo y agentes de código
DeepSeek-R1	14B	7.3 GB	Razonamiento profundo
Gemma 3	27B	6.1 GB	Propósito general, eficiente
LLaMA 3.2 Vision	11B	7.8 GB	Multimodal, análisis de imágenes
GPT-OSS	20B	16 GB	Razonamiento complejo, instrucciones
DeepHat-V1	7B	4.7 GB	Modelo especializado (Q4_K_M)

Tabla 4-4 Modelos evaluados en entorno de grupo de trabajo (Fuente: Elaboración propia)

4.3.4 Implementación del sistema RAG

El sistema RAG permite realizar consultas sobre documentación interna de la organización, fundamentando las respuestas generadas en fuentes verificables y controladas. De este modo, se mejora la fiabilidad de la información proporcionada por el modelo y se reduce la aparición de respuestas no fundamentadas.

El subsistema RAG se articula en torno a los siguientes componentes principales. En primer lugar, el módulo de embeddings se basa en el modelo BGE-M3, ejecutado a través del motor de inferencia Ollama, lo que garantiza coherencia en el entorno de ejecución y operación completamente offline. En segundo lugar, la base de datos vectorial empleada es ChromaDB, integrada directamente en Open WebUI, lo que simplifica el despliegue y la gestión del sistema. Finalmente, para el presente trabajo, los formatos de documentos soportados se han limitado a pdf, doc(x), xls(x), txt, md y html, al considerarse suficientes para cubrir los casos de uso analizados.

El pipeline de procesamiento y recuperación se configura mediante un conjunto reducido de parámetros clave. El tamaño de fragmento (*chunk size*) se ha fijado en 512 tokens, como compromiso entre preservar contexto suficiente y mantener una recuperación precisa. Se aplica un solapamiento del 10 % entre fragmentos consecutivos, equivalente a aproximadamente 51 tokens, con el objetivo de preservar la continuidad semántica y minimizar pérdidas de información en los límites de fragmentación.

El número de fragmentos recuperados por consulta (top-k) se establece en 5, permitiendo incorporar al contexto un conjunto acotado de documentos relevantes sin penalizar el rendimiento. El modelo de embeddings utilizado en todas las fases del pipeline es BGE-M3, garantizando consistencia entre indexación y consulta.

Proceso de ingesta

El proceso de ingesta documental en el sistema RAG sigue un flujo estructurado y reproducible, compuesto por las siguientes fases:

- Carga del documento a través de la interfaz web de Open WebUI, que incorpora el fichero al repositorio documental gestionado por el sistema RAG.
- Extracción de texto.
- Segmentación recursiva del contenido.
- Generación de embeddings, mediante el modelo BGE-M3.
- Almacenamiento en la base de datos vectorial ChromaDB, junto con metadatos asociados.

Corpus documentales generados

Para la fase experimental se construyeron dos corpus documentales diferenciados, con el fin de cubrir tanto documentación normativa y técnica como material heterogéneo destinado a pruebas de análisis.

El corpus de normativa está compuesto por 529 documentos, que incluyen normativa y guías de referencia, entre ellas las guías CCN-STIC. Por su parte, el corpus de ciberseguridad reúne 226 documentos de carácter técnico relacionados con esta disciplina, obtenidos a partir de fuentes editoriales especializadas como Elsevier.

La generación de ambos corpus se automatizó mediante scripts de descarga y preparación, diseñados con criterios de reutilización y extensibilidad para facilitar la construcción de nuevos conjuntos de datos en trabajos futuros. A modo de referencia, el código fuente correspondiente se incluye en el Anexo I.

Sobre esta base documental se definió un conjunto de pruebas experimentales orientadas a comparar el comportamiento del sistema en distintos escenarios, evaluando su funcionamiento sin recuperación documental y con RAG, así como con y sin directrices explícitas de *prompting*. El objetivo de estas pruebas fue analizar el impacto de dichas configuraciones en términos de precisión de las respuestas, trazabilidad de la información empleada y grado de estructuración del contenido generado.

Prueba 1: Consulta normativa específica

Consulta: "¿Cuáles son los requisitos de control de acceso en el ENS para nivel alto?"

Los resultados de la Prueba 1 muestran que la calidad y utilidad de las respuestas ante una consulta normativa específica dependen de forma directa de la existencia del RAG y del prompt del sistema. En ausencia de RAG, el modelo genera respuestas genéricas sin referencias normativas específicas, mientras que la activación de RAG sin un prompt de sistema adecuado permite recuperar documentación relevante, aunque sin una estructuración clara. Por el contrario, la combinación de RAG con un prompt de sistema explícito produce respuestas precisas, estructuradas y con trazabilidad de fuentes. La Tabla 4-5 recoge de forma comparativa los resultados obtenidos para cada una de las configuraciones evaluadas.

Condición	Respuesta obtenida	Observaciones
Sin RAG	Respuesta genérica sobre control de acceso, sin mencionar artículos específicos del ENS	Información correcta pero no específica
Con RAG, sin prompt	Menciona requisitos del ENS con fragmentos relevantes, sin estructura clara	Recupera documentos correctos
Con RAG + prompt sistema	Respuesta estructurada citando medidas específicas con referencia al documento fuente	Óptimo: preciso y verificable

Tabla 4-5 Consulta sobre normativa específica (Fuente: Elaboración propia)

La Figura 4-4 muestra un ejemplo de respuesta con citación de fuentes del sistema RAG.

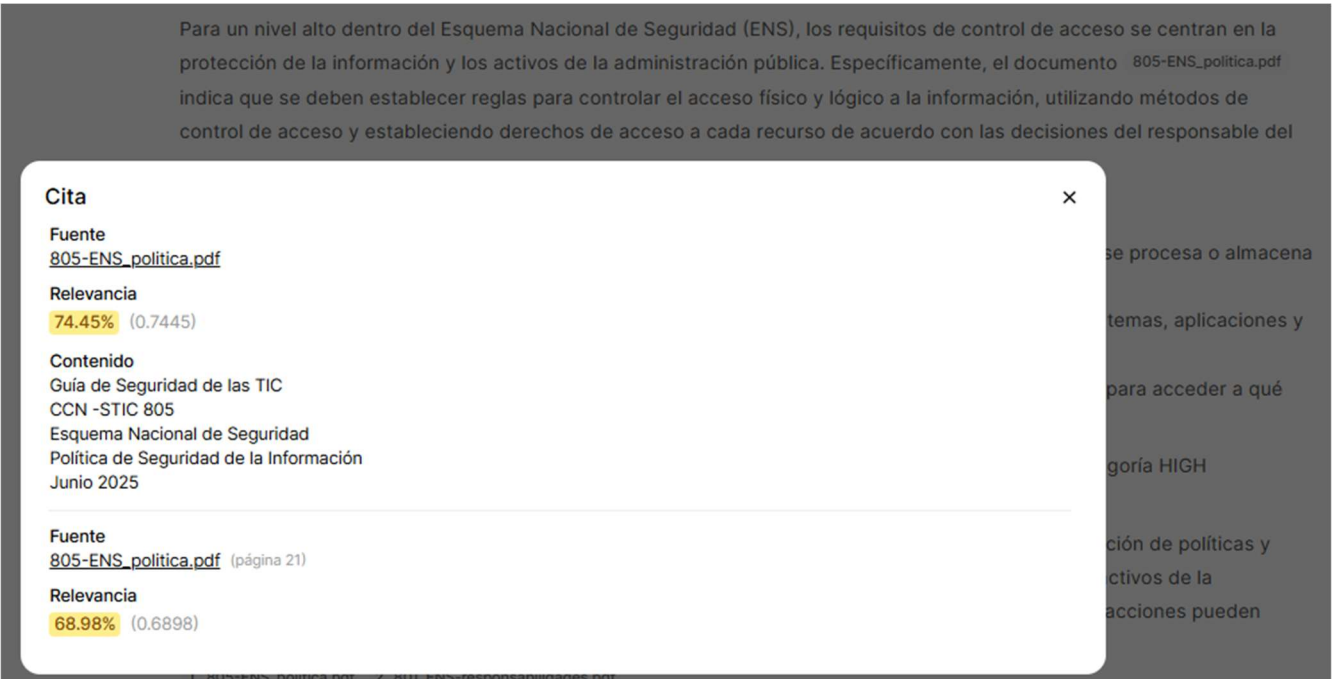


Figura 4-4 Ejemplo de fuente RAG en Open WebUI (Fuente: Elaboración propia)

Prueba 2: Consulta que requiere información no indexada

Consulta: "¿Cuál es la configuración recomendada para el firewall de Fortinet?"

Los resultados de la Prueba 2 muestran el comportamiento del sistema ante consultas que requieren información no indexada en el corpus documental. En ausencia de RAG, el modelo proporciona una configuración genérica basada en su conocimiento interno, con el consiguiente riesgo de ofrecer información desactualizada o no verificable. Cuando se habilita RAG sin un prompt de sistema adecuado, el modelo intenta generar una respuesta combinando fragmentos no relacionados, lo que degrada la coherencia y utilidad del resultado. Por el contrario, la combinación de RAG con un prompt de sistema explícito permite al modelo reconocer de forma clara la ausencia de documentación específica sobre el producto consultado en el corpus, comportamiento considerado óptimo al identificar correctamente la limitación del conocimiento disponible.

La Tabla 4-6 resume de forma comparativa los resultados obtenidos para cada una de las configuraciones evaluadas.

Condición	Respuesta obtenida	Observaciones
Sin RAG	Proporciona configuración genérica basada en conocimiento del modelo	Riesgo de información desactualizada
Con RAG, sin prompt	Intenta responder mezclando fragmentos no relacionados	Fragmentos recuperados no relevantes
Con RAG + prompt sistema	Indica explícitamente que no dispone de documentación específica sobre Fortinet en el corpus	Óptimo: reconoce limitación

Tabla 4-6 Consulta de información no indexada (Fuente: Elaboración propia)

La Figura 4-5 muestra la respuesta generada por el sistema para una consulta que requiere información no indexada.

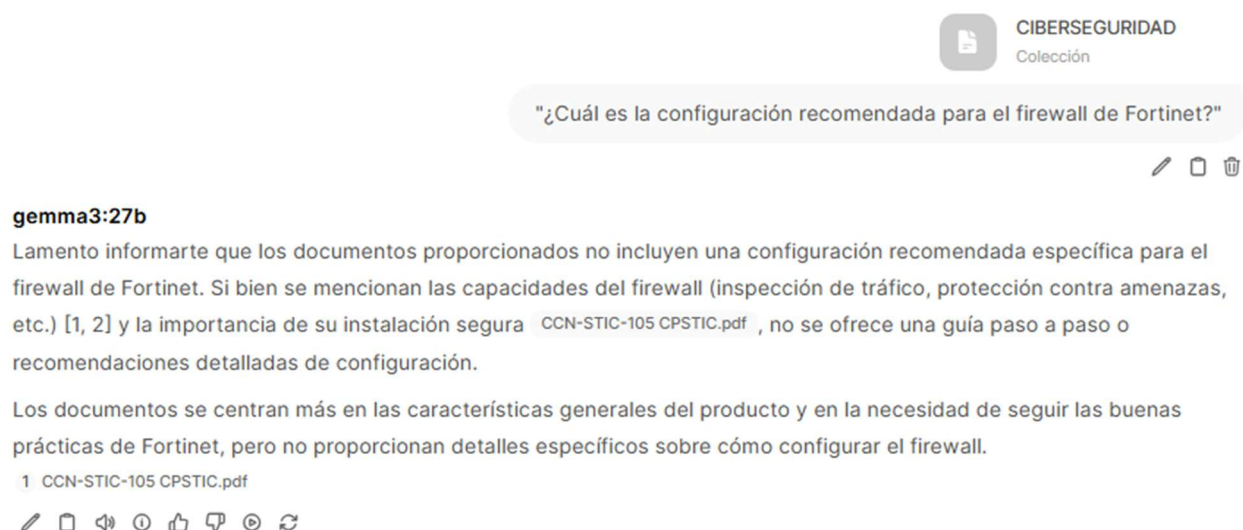


Figura 4-5 Respuesta en Open WebUI (Fuente: Elaboración propia)

Prueba 3: Consulta en inglés sobre corpus científico

Consulta: "What are the main techniques for EDR evasion?"

Los resultados de la Prueba 3 ponen de manifiesto la influencia del sistema RAG y del prompt de sistema en consultas realizadas en lengua inglesa sobre un corpus científico. En ausencia de RAG, el modelo genera una respuesta general basada en su conocimiento interno, sin referencia a literatura científica específica. Cuando se habilita RAG sin un prompt de sistema que establezca directrices claras, el modelo recupera fragmentos de publicaciones relevantes, aunque presenta inconsistencias en el idioma y en el formato de la respuesta. Por el contrario, la combinación de RAG con un prompt de sistema explícito permite generar una respuesta estructurada en inglés, manteniendo la coherencia lingüística y citando fuentes científicas concretas, comportamiento considerado óptimo en contextos de consulta académica.

La Tabla 4-7 recoge de forma comparativa los resultados obtenidos para cada una de las configuraciones evaluadas.

Condición	Respuesta obtenida	Observaciones
Sin RAG	Respuesta general sobre evasión de EDR	Conocimiento del modelo base
Con RAG, sin prompt	Recupera fragmentos de <i>papers</i> en inglés, respuesta mezclada EN/ES	Recuperación correcta, formato inconsistente
Con RAG + prompt sistema	Respuesta estructurada en inglés citando <i>papers</i> específicos (autor, año, conferencia)	Óptimo: mantiene idioma y cita fuentes

Tabla 4-7 Consulta en inglés sobre corpus científico (Fuente: Elaboración propia)

La Figura 4-6 muestra la respuesta del sistema a una consulta en inglés sobre el corpus científico indexado.

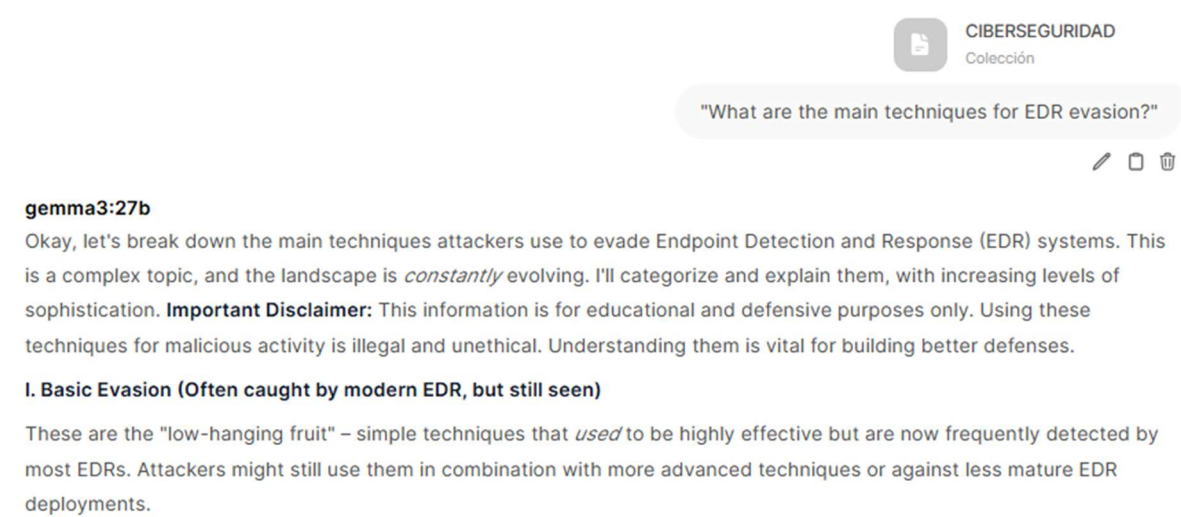


Figura 4-6 Respuesta en Open WebUI (Fuente: Elaboración propia)

Conclusiones de la validación

Los resultados obtenidos en las pruebas realizadas permiten concluir que el subsistema RAG funciona de manera adecuada y consistente en los escenarios evaluados. En particular, se han observado los siguientes aspectos relevantes.

- Recuperación efectiva: los fragmentos recuperados son relevantes y alineados con la consulta formulada.
- Citación de fuentes: la interfaz presenta de forma explícita los documentos utilizados como contexto para la generación de la respuesta, lo que permite al usuario verificar el origen de la información empleada.
- Tiempos aceptables: la latencia adicional introducida por el uso de RAG, situada aproximadamente entre 2 y 4 segundos, se considera compatible con un uso interactivo en los entornos analizados.
- Importancia del prompt de sistema: Las pruebas evidencian que un prompt de sistema adecuadamente definido, que proporcione instrucciones claras y estructuradas, mejora de forma notable la precisión, la coherencia y la verificabilidad de los resultados. bien diseñado mejora significativamente la calidad de las respuestas.
- Se destaca la citación de fuentes, el reconocimiento explícito cuando la información no está disponible y la consistencia en formato e idioma de respuesta.

4.3.5 Servicios auxiliares

Los servicios auxiliares amplían la plataforma de inteligencia artificial *offline* mediante la incorporación de capacidades especializadas, tales como traducción automática, generación y edición de imágenes y asistencia a la programación. Estas funcionalidades se exponen como microservicios locales y se integran de forma unificada a través de Open WebUI.

Traducción: LibreTranslate

LibreTranslate proporciona traducción automática multilingüe en modo *offline* mediante un microservicio REST local, integrable tanto con la interfaz de usuario Open WebUI como con los distintos *pipelines* internos de la plataforma.

Su despliegue se realiza mediante contenedores *Docker*, con persistencia de los modelos de traducción en volúmenes dedicados, lo que elimina dependencias externas y garantiza la privacidad de la información procesada, así como la trazabilidad de las operaciones.

A continuación se describen los comandos mínimos necesarios para desplegar y ejecutar LibreTranslate en un entorno *Docker*.

```
# Instalación de LibreTranslate en Docker

docker volume create lt-local

docker run -it --rm \
  --name libretranslate \
  -v lt-local:/app \
  libretranslate/libretranslate:latest-cuda
```

Tras su despliegue, LibreTranslate expone una API REST accesible de forma local a través del puerto configurado. Esta interfaz permite realizar operaciones de traducción automática y detección de idioma, facilitando su integración con Open WebUI y con otros componentes internos de la plataforma, sin necesidad de conectividad a internet. La Figura 4-7 muestra la interfaz web del servicio LibreTranslate desplegado localmente.

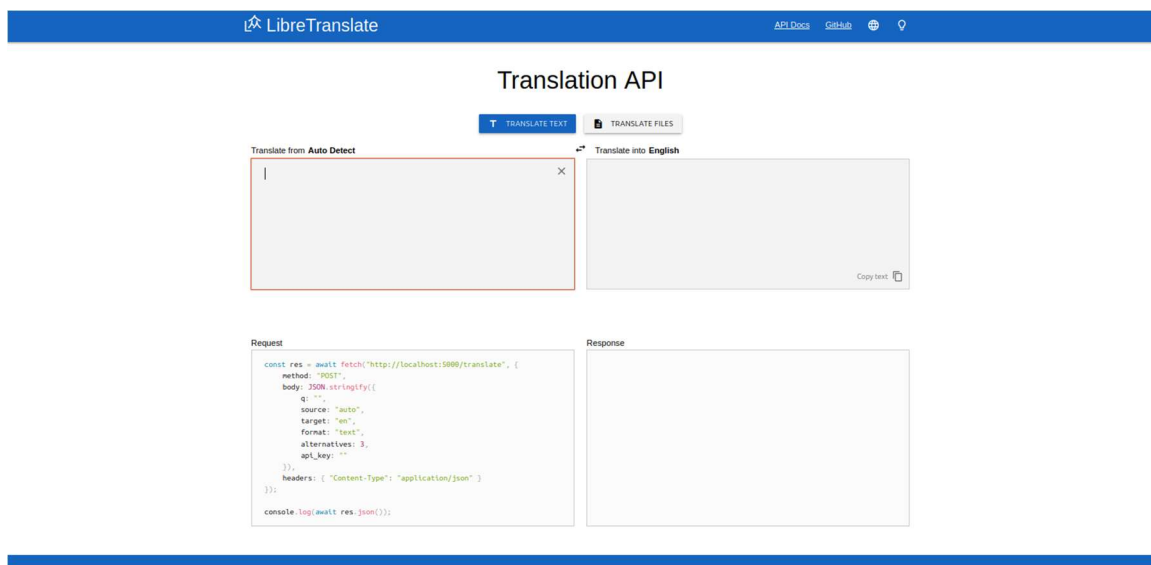


Figura 4-7 Interfaz LibreTranslate (Fuente: Elaboración propia)

Para integrar LibreTranslate con Open WebUI, se incorpora una interfaz basada en OpenAPI que expone el servicio de traducción como un *endpoint* local. A través de esta interfaz, Open WebUI realiza llamadas HTTP al servicio, indicando explícitamente el idioma de origen, el idioma de destino y el texto a traducir, de forma transparente para el usuario y manteniendo la operación completamente desconectada de internet.

En el Anexo II se incluye una especificación OpenAPI del servicio LibreTranslate en su despliegue local, donde se recogen los endpoints disponibles y los esquemas básicos de petición y respuesta para su integración.

Ejemplo de uso:

traducir a es: مع LibreTranslate مع Open WebUI مرحبًا! أعمل على دمج

La Figura 4-8 muestra un ejemplo de uso de la funcionalidad de traducción integrada en Open WebUI.



Figura 4-8 Integración de LibreTranslate en Open WebUI (Fuente: Elaboración propia)

LibreTranslate proporciona capacidades de traducción automática multilingüe. Entre los idiomas soportados se incluyen aquellos de uso habitual en documentación técnica y normativa europea, como español, inglés, francés y alemán, así como lenguas cooficiales y regionales empleadas en entornos administrativos, entre ellas catalán, gallego y euskera. Adicionalmente, el servicio ofrece soporte para otros idiomas de amplia implantación internacional, como árabe, chino, japonés o coreano.

Generación de imágenes: Stable Diffusion

Stable Diffusion proporciona capacidades de generación y edición de imágenes en entorno *air-gapped*, a partir de descripciones textuales (*text-to-image*) o de imágenes de partida (*image-to-image*), entre otras modalidades [57].

A continuación se describen los comandos mínimos necesarios para el despliegue de Stable Diffusion en el entorno de ejecución definido.

```
# Instalación de StableDiffusion:
git clone https://github.com/AUTOMATIC1111/stable-diffusion-webui.git
cd stable-diffusion-webui
# Instalar requirements.txt y ejecutar
webui.sh -api
```

La funcionalidad se integra mediante API como servicio auxiliar y puede invocarse directamente desde el chat de Open WebUI, donde el usuario introduce el prompt de generación o edición. Open WebUI envía la solicitud al motor de generación de imágenes desplegado localmente y devuelve el resultado en la conversación, permitiendo un flujo de trabajo unificado dentro de la misma interfaz. La Figura 4-9 muestra la configuración de integración de Stable Diffusion en Open WebUI.

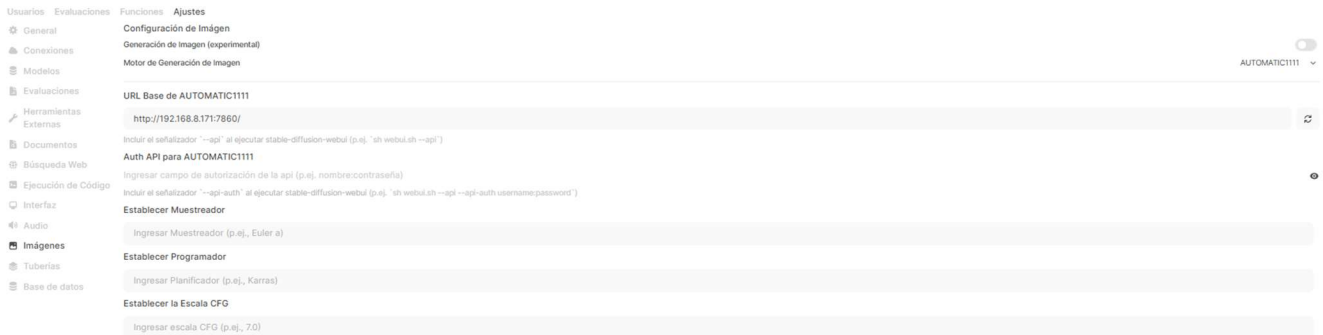


Figura 4-9 Integración de StableDiffusion en Open WebUI (Fuente: Elaboración propia)

Desde el propio chat de Open WebUI, se hace la llamada para generar las imágenes seleccionando el icono de imagen disponible después de la integración de StableDiffusion. La Figura 4-10 muestra el icono de acceso a la generación de imágenes desde el chat de Open WebUI.

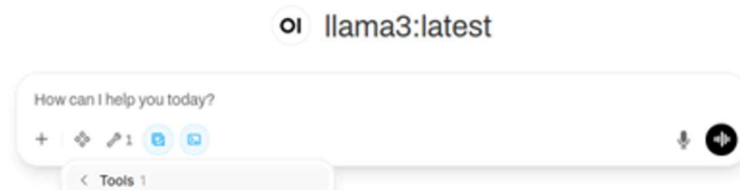


Figura 4-10 Stable Diffusion en Open WebUI (Fuente: Elaboración propia)

Asistencia a programación: Continue + Ollama

La asistencia a la programación se incorporó como una funcionalidad complementaria de la plataforma, orientada a proporcionar apoyo básico a tareas de desarrollo y revisión de código en un entorno completamente offline. Esta capacidad amplía el alcance funcional del sistema más allá de la interacción conversacional general, integrando asistencia contextual sobre código fuente dentro del ecosistema de la solución propuesta.

Inicialmente, esta capacidad se materializó mediante la funcionalidad de asistencia al desarrollo integrada en Open WebUI, permitiendo la consulta y explicación de fragmentos de código directamente desde la interfaz web. De forma adicional, se integró, mediante el plugin *Continue*, *Visual Studio Code* con Ollama para obtener asistencia al desarrollo de forma *air-gapped* [58].

```
# Instalación de Continue en Visual Studio Code:
code --install-extension continue-x.x.x.vsix
```

La Figura 4-11 muestra la interfaz de Continue integrada en Visual Studio Code.

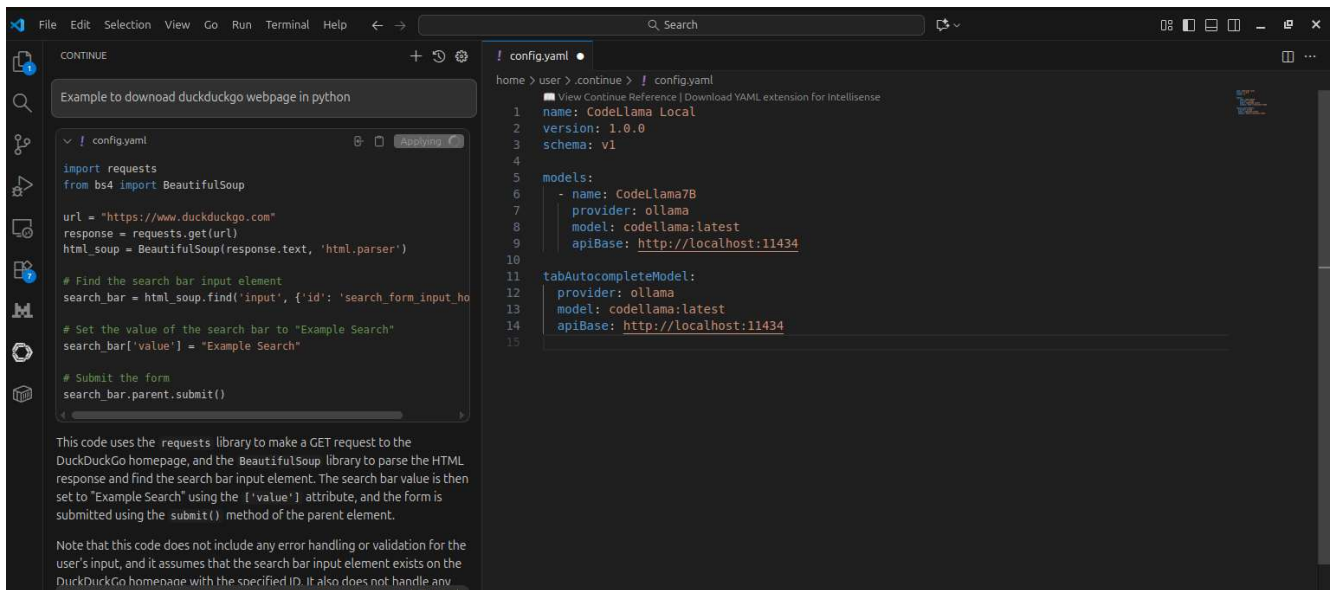


Figura 4-11 Interfaz de Continue en Visual Studio Code (Fuente: Elaboración propia)

4.3.6 Integración con agentes MCP

Model Context Protocol (MCP) es un protocolo que permite conectar modelos de lenguaje con herramientas externas de forma estandarizada, habilitando la ejecución de acciones como consultas, lecturas controladas o análisis, mediante interfaces bien definidas. Este enfoque facilita la incorporación de capacidades especializadas sin necesidad de modificar el modelo de lenguaje, manteniendo la operación offline y favoreciendo la trazabilidad, el control y la auditabilidad de las interacciones.

En el marco de la solución propuesta, Open WebUI soporta la integración con servidores MCP. Para este trabajo se ha desarrollado de forma específica el conector GhidraMCP, orientado al apoyo en tareas de ingeniería inversa y análisis de binarios. A través de esta integración, el modelo puede solicitar información relevante del proceso de análisis (funciones, estructuras o resultados del desensamblado) y generar explicaciones y salidas estructuradas que asisten al analista.

De forma adicional, se encuentran en desarrollo otros agentes que se comunican mediante MCP, con el objetivo de integrar la solución con herramientas propietarias ya existentes en la organización, así como con el gestor documental y el gestor de incidencias corporativos.

La Figura 4-12 muestra el diagrama de bloques del plugin de Ghidra.

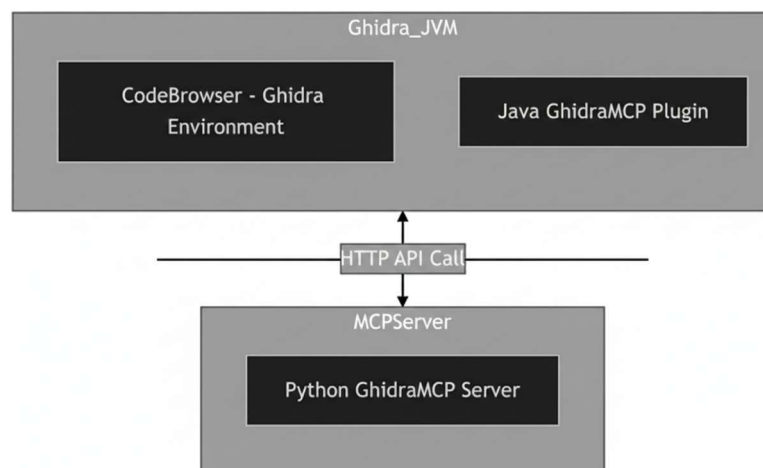


Figura 4-12 Diagrama de bloques del plugin de Ghidra (Fuente: Elaboración propia)

Integración MCP GHIDRA

Para integrar Ghidra con Open WebUI mediante el Model Context Protocol (MCP), se llevaron a cabo los siguientes pasos, orientados a habilitar la comunicación controlada entre el modelo de lenguaje y la herramienta de análisis, manteniendo la operación en entorno offline y garantizando la trazabilidad de las interacciones [59].

La Figura 4-13 muestra la configuración del plugin MCP en Ghidra.

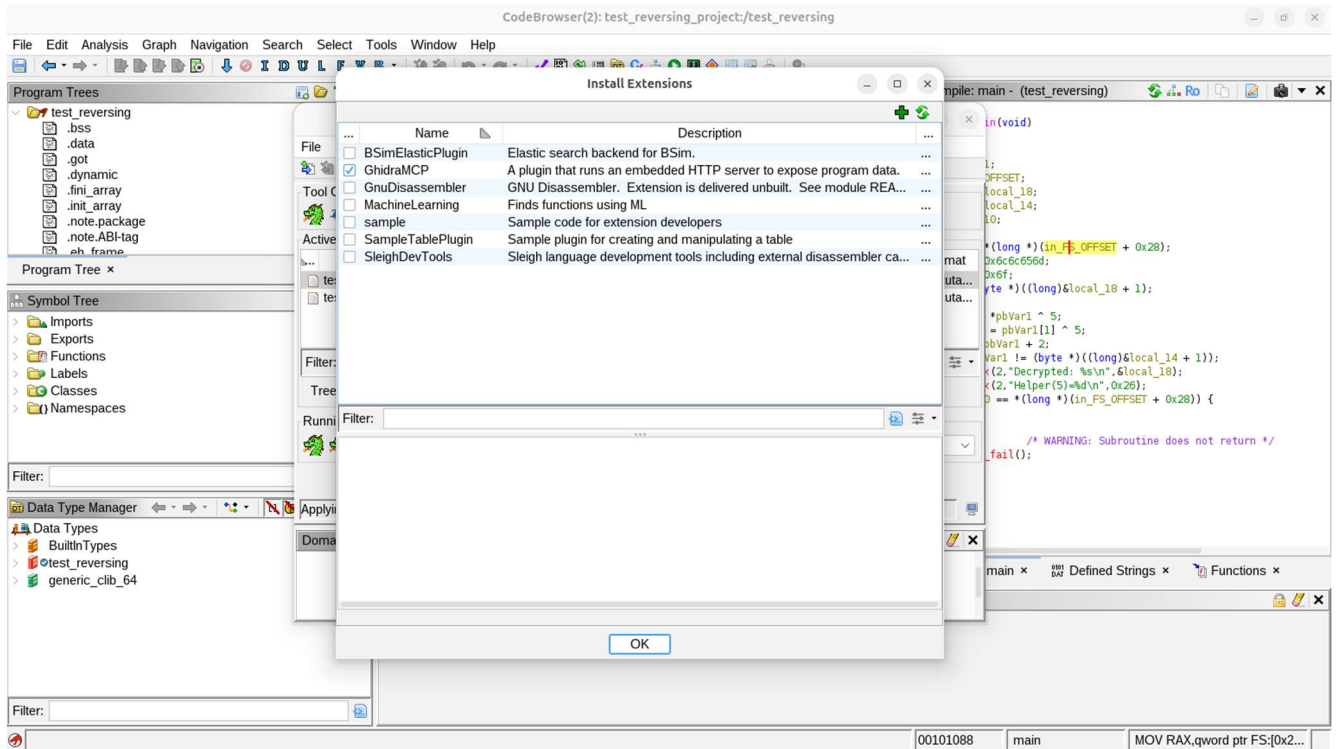


Figura 4-13 Configuración de Ghidra (Fuente: Elaboración propia)

La Figura 4-14 muestra la integración del servidor MCP de Ghidra en Open WebUI.

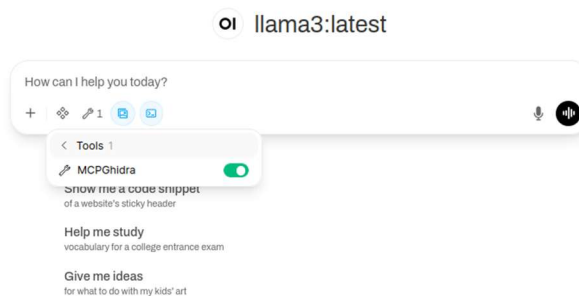


Figura 4-14 Integración de Ghidra en Open WebUI (Fuente: Elaboración propia)

Como prueba de concepto de la interoperabilidad del sistema en un entorno aislado, la Figura 4-15 muestra un ejemplo de ejecución real resultante de la integración entre la interfaz de usuario Open WebUI y el entorno de ingeniería inversa Ghidra.



Figura 4-15 Integración Ghidra-MCP en Open WebUI (Fuente: Elaboración propia)

4.3.7 Resultados obtenidos

La validación del entorno de trabajo se llevó a cabo mediante un conjunto de pruebas funcionales, de integración, de rendimiento y de estabilidad, con el objetivo de evaluar la operatividad global de la plataforma y su adecuación a los requisitos de diseño definidos para un escenario de uso de entre 10 y 15 usuarios concurrentes.

Las pruebas funcionales se orientaron a comprobar el correcto comportamiento de las capacidades básicas del sistema desde la perspectiva tanto del usuario final como del administrador. En este contexto, se analizaron los principales flujos de interacción, incluyendo los procesos de autenticación, la gestión de conversaciones, la selección y cambio de modelos de lenguaje, así como las funciones de administración de usuarios.

La Tabla 4-8 recoge de forma detallada las pruebas funcionales realizadas y los resultados obtenidos para cada una de ellas.

Prueba	Descripción	Resultado
Autenticación	Login con credenciales válidas e inválidas	Correcto
Autorización	Verificación de permisos por rol	Correcto
Chat básico	Conversación simple sin RAG	Correcto
Chat con RAG	Consulta sobre documentos indexados	Correcto
Cambio de modelo	Selección de diferentes modelos	Correcto
Historial	Persistencia y recuperación de conversaciones	Correcto
Exportación	Descarga de conversaciones en formato JSON	Correcto
Gestión usuarios	Alta, baja y modificación desde panel admin	Correcto

Tabla 4-8 Resultados pruebas funcionales (Fuente: Elaboración propia)

Las pruebas funcionales se completaron sin incidencias, evidenciando el cumplimiento de los requisitos definidos para cada uno de los perfiles de usuario. El sistema de control de roles operó conforme al modelo de autorización establecido, restringiendo a los usuarios básicos a funciones de consulta y habilitando a los administradores el acceso a las capacidades completas de gestión de la plataforma.

Por su parte, las pruebas de integración permitieron validar la interoperabilidad entre los distintos servicios desplegados, confirmando el funcionamiento extremo a extremo de los procesos de inferencia, del RAG y de los servicios auxiliares, tales como traducción automática, generación de imágenes y asistencia al desarrollo de código. La Tabla 4-9 recoge los resultados obtenidos en las pruebas de integración realizadas, detallando los componentes evaluados y el alcance de la validación efectuada.

Prueba	Componentes	Alcance verificado	Resultado
Open WebUI -Ollama	Interfaz / Motor de inferencia	Generación de respuestas y estabilidad de la sesión de inferencia	Correcto
Open WebUI - ChromaDB	Interfaz / Vector DB	Ingesta, indexación y recuperación de fragmentos (RAG)	Correcto
Open WebUI - LibreTranslate	Interfaz / Traducción	Traducción de textos vía endpoint local y retorno al chat	Correcto
Open WebUI -Stable Diffusion	Interfaz / Imagen	Generación de imágenes desde prompts y entrega del resultado	Correcto
Ollama -BGE-M3	Motor / Embeddings	Cálculo de embeddings para documentos y consultas	Correcto

Tabla 4-9 Resultados pruebas de integración (Fuente: Elaboración propia)

Las mediciones de rendimiento obtenidas permitieron evaluar la viabilidad de la solución en escenarios de uso interactivo. El modelo Mistral de 7 mil millones de parámetros alcanzó una tasa media de generación superior a 150 tokens por segundo, superando el umbral mínimo establecido de 40 tokens por segundo para garantizar una interacción fluida.

Por su parte, los modelos de mayor tamaño como Codestral (22B) y Devstral (24B), registraron valores comprendidos entre 50 y 60 tokens por segundo, manteniéndose por encima del umbral definido y resultando compatibles con un uso interactivo en las condiciones de prueba consideradas.

Auditoría y trazabilidad

Open WebUI, a través de su panel de administración, permite la gestión de usuarios y roles, así como la visualización del conjunto de datos indexado —colecciones y documentos— empleado por el sistema RAG. Sin embargo, en escenarios que requieren capacidades avanzadas de auditoría, tales como la generación de informes periódicos, el análisis de métricas agregadas, la obtención de evidencias de actividad o una trazabilidad detallada del uso del sistema, resulta necesario acceder directamente a la base de datos de Open WebUI.

Con este propósito, y como apoyo a la validación experimental del presente trabajo, se desarrolló un script específico en Python orientado al análisis de la base de datos *SQLite* de Open WebUI. Dicho script actúa como una herramienta de extracción automatizada de métricas, permitiendo la obtención de evidencias cuantitativas sobre el uso real de la plataforma.

En concreto, la herramienta desarrollada permite obtener información cuantitativa sobre distintos aspectos del uso del sistema, incluyendo estadísticas de usuarios, como el número de usuarios registrados, la distribución por roles y la actividad reciente; estadísticas relativas a las conversaciones, tales como el número total, su evolución temporal y los usuarios más activos; estadísticas sobre los

modelos de lenguaje, incluyendo los modelos registrados y su frecuencia relativa de utilización; así como métricas específicas del sistema RAG, como el número de documentos y colecciones indexadas. Se adjunta el resultado resumido de la ejecución del *script* de auditoría.

```
=====
ANÁLISIS DE BASE DE DATOS OPEN WEBUI
=====
Fecha de análisis: 2026-01-05 00:06:05
ESTRUCTURA DE LA BASE DE DATOS
-----
Total de tablas: 24
Total de registros: 16,616
ESTADÍSTICAS DE USUARIOS
-----
Total usuarios registrados: 11
Usuarios activos última semana: 7
Usuarios activos último mes: 11
ESTADÍSTICAS DE CONVERSACIONES
-----
Total conversaciones: 128
Promedio por usuario: 11.6
USO DE MODELOS
-----
Modelos disponibles: 7
SISTEMA RAG
-----
RAG habilitado: Sí
Total documentos: 16,520
Tipos de documentos: {"CCN-STIC-599B18.pdf", "CCN-STIC-652A Seguridad en
contenedores.pdf",...
=====
```

En conjunto, este desarrollo constituye una prueba de concepto de la capacidad de auditoría que puede alcanzarse sobre Open WebUI en un entorno offline. Demuestra que, además de la administración básica, se podrían implementar mecanismos de toma de evidencias, trazabilidad y reporting mediante herramientas propias, manteniendo el control local de los datos y la independencia de servicios externos. El código fuente del script se incluye en el Anexo III de este trabajo, al que se remite para su consulta detallada.

Limitaciones identificadas

Las pruebas realizadas permitieron identificar una serie de limitaciones que deben ser consideradas en escenarios de despliegue en producción.

- Dependencia del prompt de sistema: la calidad de las respuestas generadas mediante RAG depende en gran medida de un diseño adecuado del prompt de sistema. En ausencia de instrucciones explícitas, el modelo tiende a combinar conocimiento interno con información recuperada, sin una delimitación clara entre ambas fuentes.
- Generación de imágenes: Stable Diffusion, con la configuración estándar, ofrece resultados válidos para ilustraciones conceptuales, pero no resulta adecuado para diagramas técnicos precisos o para necesidades específicas.
- Escalabilidad limitada: la arquitectura probada no contempla alta disponibilidad ni balanceo de carga. Para despliegues organizacionales de mayor escala sería necesario evolucionar hacia una arquitectura distribuida.
- Ventana de contexto: los modelos actuales ofrecen ventanas que oscilan entre 8.192 tokens (modelos base) y 128.000 tokens o superiores, lo que requiere seleccionar el modelo adecuado según el caso de uso o implementar estrategias de segmentación de documentos en caso necesario.

Estas limitaciones resultan aceptables en el contexto del trabajo y quedan documentadas para su consideración en futuras evoluciones del sistema.

4.4 Infraestructura de supercomputación (CESGA)

De forma análoga al análisis realizado para los escenarios de entorno monopuesto y de grupo de trabajo reducido (hasta 15 usuarios), se estudia la viabilidad de emplear infraestructuras de supercomputación como apoyo a la solución propuesta. En particular, se considera el uso del supercomputador FinisTerra III del CESGA, orientado a tareas de ejecución avanzada y a la experimentación con modelos de mayor tamaño o cargas computacionales intensivas.

Para la implantación de servicios como Ollama, Open WebUI, LibreTranslate, Stable Diffusion y modelos orientados a la asistencia a la programación como Code LLaMA, sobre FinisTerra III, resulta necesario seguir una serie de pasos específicos, adaptados al entorno HPC y a sus políticas de acceso, asignación de recursos y ejecución:

- Acceso a la infraestructura, mediante autenticación remota y asignación de recursos en nodos de cómputo habilitados para GPU.
- Despliegue de los servicios base, incluyendo la instalación y configuración de Ollama y Open WebUI en el entorno de usuario autorizado.
- Ejecución del motor de inferencia (Ollama), asegurando la correcta visibilidad de los dispositivos GPU asignados.
- Ejecución de Open WebUI como interfaz de acceso a los modelos y servicios auxiliares.
- Exposición controlada de los servicios, mediante túneles SSH, que permiten el acceso desde la máquina local del usuario a los puertos internos del nodo de cómputo, por ejemplo:

Posteriormente, se procedió a la instalación de Ollama y Open WebUI en el entorno de supercomputación, siguiendo un procedimiento análogo al aplicado en el escenario de grupo de trabajo. Una vez desplegados ambos servicios, se verificó su correcto funcionamiento y se llevaron a cabo evaluaciones iniciales de rendimiento en las condiciones propias del entorno HPC.

```
# Ejecutar Ollama
nohup ollama serve > "$WEBUI_STATE/logs/ollama.log" 2>&1 &

# Ejecutar Open WebUI
"$ENV/bin/open-webui" serve --host 127.0.0.1 --port 8080 > "$WEBUI_STATE/logs/Open
WebUI_$(date +%F-%H%M%S).log" 2>&1 &

# verificar funcionamiento:

LISTEN 0      128      127.0.0.1:11434      0.0.0.0:*
users:(("ollama",pid=1219651,fd=3))

LISTEN 0      128      127.0.0.1:8080      0.0.0.0:*      users:(("open-
webui",pid=1219291,fd=44))

# Acceso remoto a Open-WebUI a través del puerto 8080:
ssh -J user@ft3.cesga.es user@ip -L 8080:127.0.0.1:8080 -L 11434:127.0.0.1:11434

# Open WebUI queda accesible a través de la dirección 127.0.0.1 en la máquina
local
```

El empleo de infraestructuras de supercomputación como FinisTerra III (CESGA) proporciona una capacidad de cómputo muy superior a la de los entornos locales, resultando especialmente adecuado para tareas intensivas tales como el entrenamiento y ajuste fino de modelos, la ejecución de pruebas controladas sobre grandes volúmenes de datos o la experimentación con modelos de mayor tamaño. No obstante, al tratarse de una infraestructura externa a la organización, su utilización no se ajusta a un escenario de operación completamente aislado o *air-gapped*, ya que los modelos y servicios asociados se ejecutan fuera del perímetro organizativo y requieren conectividad de red, aunque esta sea de carácter controlado.

En consecuencia, y atendiendo a los requisitos de soberanía del dato, control operativo y aislamiento que definen los entornos objetivo de este trabajo, el despliegue permanente de la solución sobre infraestructura HPC externa queda descartado como opción de explotación en producción. La infraestructura del CESGA se empleó, por tanto, de forma puntual y acotada como entorno de apoyo para actividades específicas de carácter experimental, incluyendo la evaluación de modelos de mayor tamaño, el uso intensivo de recursos GPU, el desarrollo y validación de scripts, así como la realización de pruebas controladas de rendimiento y compatibilidad.

Adicionalmente, durante la fase de validación se constató que la asignación bajo demanda de nodos de cómputo, condicionada por políticas de planificación y sistemas de colas propios del entorno HPC, junto con la disponibilidad temporal limitada de los recursos en cada sesión, dificulta tanto la usabilidad continuada de la plataforma como la realización de evaluaciones cuantitativas plenamente reproducibles.

4.5 Fine-tuning del modelo

Con el objetivo de evaluar la viabilidad de adaptar un modelo de lenguaje generalista a un dominio especializado en ciberseguridad, se desarrolló una prueba de concepto de *fine-tuning* empleando la técnica LoRA. Esta aproximación se plantea con fines académicos y experimentales, orientada a comprobar que el proceso de especialización resulta técnicamente viable en un entorno local y completamente offline, sin pretender alcanzar niveles de rendimiento equiparables a entrenamientos a gran escala.

Como modelo base se utilizó Mistral-7B-Instruct. El proceso de *fine-tuning* se ejecutó inicialmente en la infraestructura de supercomputación FinisTerra III (CESGA), con el objetivo de validar el procedimiento en un entorno HPC y aprovechar capacidades avanzadas de cómputo para pruebas controladas. Posteriormente, el mismo proceso fue replicado en una estación de trabajo local equipada con una GPU NVIDIA RTX 4090, con el fin de verificar la reproducibilidad de los resultados y confirmar la viabilidad del entrenamiento en un entorno local y offline.

4.5.1 Dataset de entrenamiento

Para el proceso de *fine-tuning* se elaboró un conjunto de datos reducido, compuesto por 300 ejemplos en español relacionados con el ámbito de la ciberseguridad. El volumen del *dataset* se mantuvo intencionadamente bajo, lo que permite validar la viabilidad técnica del proceso de especialización, pero limita de forma significativa el grado de adaptación alcanzable del modelo resultante.

El conjunto de datos se estructuró en tres bloques diferenciados, compuestos por 100 ejemplos de preguntas y respuestas orientadas a tareas defensivas y de análisis forense digital (Q&A DFIR), 100 ejemplos de clasificación de tácticas y técnicas conforme al marco MITRE ATT&CK, y 100 ejemplos de generación de resúmenes forenses estructurados, incluyendo líneas temporales, vectores de entrada, impacto y evidencias. A continuación, se presenta un ejemplo de una entrada del *dataset*:

Ejemplo de formato:

```
{
  "instruction": "Clasifica la siguiente actividad sospechosa según MITRE ATT&CK. Devuelve táctica y técnica.",
  "input": "Actividad 32: Se observa explotación a través de una API pública vulnerable que permite la ejecución remota de comandos.",
  "output": "Táctica: TA0001 - Initial Access. Técnica: T1190 - Exploit Public-Facing Application."
}
```

4.5.2 Descarga y configuración del modelo base

Se seleccionó el modelo Mistral-7B-Instruct-v0.2 como punto de partida. La descarga del modelo y su carga en modo cuantizado a 4 bits se realizaron utilizando la librería Hugging Face Transformers [60] junto con la biblioteca bitsandbytes [61], que permite la cuantización eficiente de modelos de lenguaje y la ejecución de inferencia y entrenamiento con un consumo de memoria significativamente reducido, facilitando así su despliegue en hardware con recursos limitados.

Con el objetivo de documentar y facilitar la reproducibilidad del proceso, se adjunta a continuación la salida generada durante la ejecución del script de entrenamiento.

```
python train_finetune_ciber.py
Cargando dataset...
Tokenizando ejemplos...
Map: 270/270 ejemplos procesados
Map: 30/30 ejemplos procesados
Cargando modelo Mistral-7B-Instruct-v0.2 en 4 bits...
trainable params: 41,943,040 || all params: 7,283,675,136 || trainable%: 0.5758
Configurando entrenamiento LoRA...
Iniciando entrenamiento...
{'loss': 1.3006, 'grad_norm': 0.32521235942840576, 'learning_rate':
0.0001773010453362737, 'epoch': 0.59}
{'loss': 0.058, 'grad_norm': 0.13770684599876404, 'learning_rate':
9.019828596704394e-05, 'epoch': 1.18}
{'train_runtime': 340.265, 'train_samples_per_second': 1.587,
'train_steps_per_second': 0.1, 'train_loss': 0.40950747075326305,
'epoch': 2.0}
Guardando adaptador LoRA...
Entrenamiento completado. Adaptador guardado en: ./mistral-ciber-lora
```

Durante la fase de preparación de los datos se cargó el conjunto de entrenamiento desde un fichero en formato JSONL y se procedió a la tokenización de los ejemplos. En total, se procesaron 270 ejemplos destinados al entrenamiento y 30 ejemplos reservados para validación, manteniendo una separación explícita entre ambos subconjuntos.

A continuación, se cargó el modelo base Mistral-7B-Instruct-v0.2 en modo cuantizado a 4 bits, empleando la técnica QLoRA. El modelo resultante cuenta con un total de 7.283.675.136 parámetros, de los cuales 41.943.040 se marcaron como entrenables, lo que representa aproximadamente un 0,58 % del total de parámetros del modelo. El código fuente del script de entrenamiento (train_finetune_ciber.py) se incluye en el Anexo IV de este trabajo.

El proceso completo de entrenamiento mediante LoRA tuvo una duración aproximada de 340,3 segundos, alcanzando una velocidad media de 1,59 muestras por segundo y 0,10 pasos de entrenamiento por segundo.

Finalmente, se verificó el funcionamiento del modelo reentrenado mediante la ejecución de pruebas específicas de inferencia, con el objetivo de comprobar la correcta integración de los pesos LoRA y el comportamiento del modelo adaptado. El código fuente del script de evaluación (eval_lora_mistral.py) se incluye en el Anexo V de este trabajo.

A continuación se muestra la salida obtenida durante la ejecución del script de evaluación del modelo.

```
(finetunning) user@pc-ia:~/finetunning$ python eval_lora_mistral.py
Cargando modelo base y adaptador LoRA...
=== Instrucción ===
Clasifica la siguiente actividad sospechosa según MITRE ATT&CK. Devuelve táctica y técnica.
=== Entrada ===
El atacante usa PowerShell para ejecutar scripts descargados desde una URL remota y establecer persistencia.
=== Respuesta ===
Táctica: TA0005 - Defense Evasion. Técnica: T1055.002 - PowerShell.
```

4.5.3 Fusión del adaptador LoRA con el modelo base

Con el fin de obtener un modelo final autónomo, sin dependencia del modelo base original ni de adaptadores externos, se procedió a la fusión de los pesos LoRA con el modelo base mediante el script `merge_lora_mistral.py` incluido en el Anexo VI.

Para su utilización en entornos *air-gapped* y en plataformas de ejecución local como LM Studio u Ollama, el modelo fusionado se convirtió al formato GGUF [62] mediante el script `convert_hf_to_gguf.py` del proyecto `llama.cpp`. El proceso se realizó partiendo del modelo previamente fusionado (`mistral-ciber-merged`) y generando un fichero final cuantizado en precisión Q8_0, adecuado para su ejecución eficiente en entornos locales sin dependencia de servicios externos.

Durante el proceso de exportación, el script identificó automáticamente la arquitectura del modelo (`MistralForCausalLM`) y generó un fichero en formato GGUF que contiene 291 tensores, con un tamaño total aproximado de 7,7 GB. Dicho fichero corresponde a una configuración con una longitud de contexto de 32 768 tokens y una dimensión de embedding de 4 096. El modelo resultante se almacenó bajo el nombre `mistral-ciber-Q8_0.gguf`.

4.5.4 Registro en Ollama

Una vez completado el proceso de conversión del modelo ajustado a formato GGUF, se procedió a su registro en el entorno de ejecución Ollama, con el objetivo de habilitar su uso como modelo local de inferencia dentro de la arquitectura offline propuesta.

Para su incorporación al catálogo de modelos de Ollama, se definió un fichero `Modelfile` específico, en el que se referencian el modelo base y el artefacto GGUF generado. El proceso de registro se realizó mediante el comando `ollama create`, que encapsula el modelo dentro del repositorio interno de Ollama, generando las capas necesarias y el manifiesto correspondiente.

Tal y como se observa en el registro de ejecución, Ollama completó correctamente la creación del modelo, asignándole un identificador único y dejándolo disponible en el entorno local junto con el resto de modelos instalados. La verificación posterior mediante el comando `ollama list` confirmó la correcta incorporación del modelo personalizado, que aparece registrado con un tamaño aproximado de 7,7 GB, coherente con el resultado de la fase de conversión.

Este procedimiento garantiza que el modelo ajustado queda plenamente integrado en el runtime local, sin dependencias externas, y listo para ser utilizado por las distintas capas de la solución (interfaz de usuario, RAG y agentes MCP).

4.5.5 Integración en Open WebUI

Una vez generado el modelo ajustado y registrado en Ollama, este quedó disponible como un artefacto local adicional dentro del catálogo de modelos desplegados. Establecida la conexión correspondiente, el modelo mistral-ciber pudo seleccionarse desde el desplegable de modelos de Open WebUI, quedando habilitado para su utilización en sesiones de chat, en pruebas funcionales, en el RAG, entre otros. La Figura 4-19 muestra un ejemplo de consulta al modelo mistral-ciber tras el proceso de *fine-tuning*.

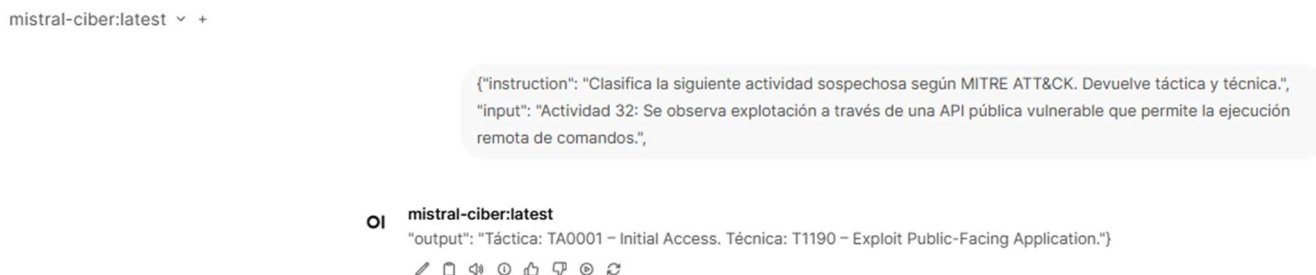


Figura 4-16 Salida del modelo mistral-ciber (Fuente: Elaboración propia)

4.6 RAG versus Fine-tuning: Criterios de selección

Una de las decisiones fundamentales en el diseño y despliegue de sistemas de inteligencia artificial aplicados a dominios específicos consiste en seleccionar el mecanismo más adecuado para la incorporación de conocimiento especializado, ya sea mediante RAG o a través de técnicas de *fine-tuning* del modelo. A partir de los resultados y observaciones obtenidos durante el desarrollo del presente trabajo, se establecen una serie de criterios orientativos que permiten guiar dicha elección.

4.6.1 RAG como técnica preferente

RAG resulta preferible en aquellos escenarios en los que la información de dominio presenta una elevada variabilidad temporal y requiere actualizaciones frecuentes sin necesidad de reentrenar el modelo de lenguaje. En entornos de Defensa, donde la documentación normativa, los informes de inteligencia de amenazas y los procedimientos operativos se revisan y actualizan de manera periódica, RAG permite incorporar dichos cambios de forma ágil mediante la adición, modificación o sustitución de documentos en el corpus de conocimiento.

Asimismo, RAG constituye una opción especialmente adecuada cuando la trazabilidad de las respuestas es un requisito fundamental. Al vincular cada respuesta generada con los documentos fuente utilizados durante el proceso de recuperación, este enfoque facilita la verificación por parte del usuario y permite la realización de auditorías posteriores, aspecto crítico en contextos regulados y de alta exigencia operativa.

Otro escenario en el que se prioriza la utilización de RAG es aquel en el que los recursos computacionales disponibles para entrenamiento son limitados. A diferencia del *fine-tuning*, que incluso mediante técnicas eficientes como LoRA requiere el uso de GPU durante el proceso de entrenamiento, RAG no precisa capacidades de cómputo intensivo más allá de la generación de embeddings, la cual puede realizarse de manera incremental y con un impacto reducido en los recursos del sistema.

Finalmente, cuando existe un corpus documental previamente estructurado y de valor operativo, RAG se presenta como una alternativa preferente. Las organizaciones que disponen de bases documentales consolidadas pueden beneficiarse de forma inmediata de este enfoque, integrando dicho conocimiento en el sistema sin necesidad de diseñar y mantener conjuntos de datos específicos para entrenamiento del modelo.

4.6.2 *Fine-tuning* como técnica preferente

El *fine-tuning* resulta preferible en aquellos escenarios en los que se requiere una adaptación profunda del comportamiento del modelo de lenguaje, incluyendo el uso de terminología altamente especializada, la generación de formatos de respuesta específicos o la adopción de estilos de comunicación particulares. A diferencia de los enfoques basados exclusivamente en la provisión de contexto, el *fine-tuning* implica la modificación directa de los pesos del modelo, lo que permite introducir cambios más estructurales y persistentes en su comportamiento.

Asimismo, el *fine-tuning* constituye una opción adecuada cuando los requisitos de latencia son especialmente estrictos y el acceso a mecanismos de recuperación documental resulta inaceptable desde el punto de vista operativo. Al generar un modelo autocontenido, este enfoque elimina la necesidad de realizar consultas adicionales a bases de datos vectoriales durante la inferencia, reduciendo la complejidad del flujo de ejecución y los tiempos de respuesta.

Finalmente, el *fine-tuning* es especialmente apropiado cuando el conocimiento a incorporar presenta un carácter estable y no requiere actualizaciones frecuentes. Conceptos fundamentales del dominio, marcos de referencia consolidados o terminología técnica madura pueden integrarse de forma permanente en el modelo mediante este enfoque, evitando la necesidad de mantener y actualizar de manera continua un corpus externo de recuperación.

4.6.3 RAG y *Fine-tuning* como técnica preferente

La combinación de técnicas de *fine-tuning* y RAG ofrece resultados satisfactorios en aquellos escenarios en los que se requiere, de manera simultánea, coherencia estructural y terminológica en las respuestas, junto con la capacidad de incorporar información específica, verificable y sujeta a actualización frecuente. Mientras que el *fine-tuning* permite especializar el comportamiento del modelo mediante la adaptación de sus pesos, el enfoque RAG complementa estas capacidades al proporcionar acceso dinámico a conocimiento externo con trazabilidad explícita de las fuentes.

En el contexto de este trabajo, el modelo ajustado mistral-ciber, resultado del proceso de *fine-tuning*, fue integrado con un sistema RAG. Si bien el modelo reentrenado no alcanza un grado de especialización significativo debido al tamaño reducido del corpus empleado durante el proceso de *fine-tuning*, su desarrollo resulta adecuado como caso ilustrativo de la viabilidad técnica y metodológica de combinar ambas técnicas dentro de una misma arquitectura.

En este sentido, la combinación de *fine-tuning* y RAG se propone como la configuración más adecuada para sistemas de IA aplicados a dominios especializados, al conjugar especialización del modelo con flexibilidad operativa y control sobre las fuentes de conocimiento.

4.7 Retos identificados y soluciones implementadas

El capítulo de Estado del Arte identificó los principales retos que dificultan la adopción de inteligencia artificial en entornos desconectados. Esta sección documenta cómo la arquitectura diseñada y el prototipo implementado abordan específicamente cada categoría de retos mediante decisiones técnicas concretas.

4.7.1 Retos tecnológicos

Dependencia de hardware especializado con disponibilidad limitada y costes elevados

La arquitectura propuesta implementa un enfoque multinivel que permite adaptar el despliegue a los recursos disponibles. El Nivel 1 (monopuesto aislado) ejecuta modelos de 3-4B parámetros en CPU convencional sin requerir GPU dedicada, proporcionando capacidades básicas de asistencia con hardware estándar de oficina. El Nivel 2 (grupo de trabajo) incorpora una GPU de consumo RTX 4090 que permite ejecutar modelos de hasta 24B parámetros con rendimiento adecuado para grupos reducidos. El Nivel 3 (departamental) escala horizontalmente mediante GPUs adicionales según necesidades.

Esta estratificación permite que organizaciones con presupuestos limitados inicien su adopción de IA offline con inversiones modestas (1.000 a 4.000 euros por puesto), escalando posteriormente según validación de utilidad y disponibilidad presupuestaria.

Gestión del ciclo de vida en entornos *air-gapped*

La imposibilidad de resolución remota de dependencias y actualizaciones automáticas se aborda mediante tres mecanismos complementarios. Primero, todos los componentes se distribuyen como contenedores Docker exportados en formato de imágenes *tar* autocontenidas que incluyen todas las dependencias necesarias. Segundo, los modelos de lenguaje se distribuyen en formato GGUF, archivos binarios únicos sin dependencias externas que contienen tanto pesos como metadatos de configuración. Tercero, los modelos de embeddings (BGE-M3) y servicios auxiliares (LibreTranslate, Stable Diffusion) se empaquetan completos con sus dependencias en las imágenes Docker correspondientes.

La arquitectura modular con Ollama como capa de abstracción permite actualizar modelos sin modificar la infraestructura dependiente: la actualización de Mistral 7B de v0.1 a v0.3 se realizó reemplazando únicamente el archivo GGUF correspondiente, sin cambios en Open WebUI ni en las integraciones RAG.

Obsolescencia acelerada de modelos

El diseño modular basado en APIs estandarizadas (OpenAI-compatible) desacopla la interfaz de usuario y las integraciones del runtime de inferencia específico. Esto permite sustituir modelos sin reescribir aplicaciones: el cambio de Mistral 7B a Qwen 2.5 7B requirió únicamente modificar la referencia del modelo en las configuraciones de Open WebUI, manteniendo funcionalidad completa. Además, La estrategia de cuantización mediante GGUF permite mantener múltiples versiones de modelos simultáneamente con coste de almacenamiento moderado.

Rendimiento limitado sin elasticidad cloud

La cuantización a 4 bits (Q4_K_M) permite ejecutar modelos competentes en hardware limitado con degradación mínima de calidad. Las pruebas documentadas en este capítulo demuestran que Mistral 7B cuantizado genera 159 tokens/segundo en RTX 4090, velocidad adecuada para uso interactivo.

4.7.2 Retos organizativos

Resistencia al cambio

La selección de Open WebUI como interfaz principal responde directamente a este reto. Su interfaz replica deliberadamente el diseño de ChatGPT, aprovechando la familiaridad que muchos usuarios han desarrollado con interfaces conversacionales comerciales. Esta decisión reduce significativamente la barrera de entrada: usuarios que han interactuado con ChatGPT pueden transferir directamente su experiencia sin formación adicional.

El sistema se posiciona como herramienta de aumento de capacidades donde el usuario mantiene control y responsabilidad sobre las decisiones. Las respuestas del sistema incluyen advertencias explícitas sobre verificación de información crítica y la interfaz permite visualizar el razonamiento y las fuentes consultadas (en modo RAG), reforzando el rol del humano como supervisor final.

Escasez de perfiles técnicos especializados

La operación de sistemas de IA tradicionalmente requiere perfiles de MLOps e ingeniería de IA, escasos en organizaciones de defensa. La arquitectura propuesta mitiga esta dependencia mediante herramientas que abstraen la complejidad subyacente: Ollama simplifica la gestión de modelos mediante comandos de alto nivel sin configuración manual de *backends*, mientras que *Docker Compose* orquesta

los servicios mediante ficheros YAML declarativos. La documentación paso a paso desarrollada permite a administradores de sistemas sin experiencia previa en IA realizar despliegues completos. Una vez operativo, el sistema requiere intervención mínima para su mantenimiento rutinario.

4.7.3 Retos económicos

Los retos económicos se mitigan parcialmente mediante una arquitectura escalable que permite adaptar la inversión inicial al alcance y a las necesidades del despliegue. En escenarios de puesto individual aislado, la solución puede implementarse sobre hardware basado en CPU con un coste aproximado del orden de 1.000 €, mientras que, en entornos de grupo de trabajo con capacidades compartidas, la incorporación de una GPU dedicada sitúa la inversión estimada en un rango aproximado de 3.000–4.000 €, en función de la configuración seleccionada.

Adicionalmente, el uso exclusivo de software de código abierto, distribuido bajo licencias permisivas como MIT o Apache 2.0, elimina los costes recurrentes asociados a licenciamiento, lo que reduce el coste total de propiedad y favorece la sostenibilidad económica de la solución a medio y largo plazo.

4.7.4 Retos normativos y de cumplimiento

El diseño incorpora desde el origen mecanismos que facilitan cumplimiento con ENS:

- Trazabilidad: Open WebUI registra todas las interacciones con marca temporal, identificador de usuario, y contenido de consulta y respuesta, generando evidencia auditable. Adicionalmente se ha desarrollado un script que permite la obtención de registros adicionales de auditoría de manera automática.
- Control de acceso: El sistema RBAC permite diferenciar permisos por rol (administrador, usuario estándar, invitado) con granularidad de operaciones permitidas (crear conversaciones, gestionar modelos, administrar usuarios).
- Segregación de red: Los servicios *backend* (Ollama, ChromaDB, LibreTranslate) no exponen puertos directamente accesibles desde el exterior, siendo alcanzables únicamente a través de Open WebUI mediante la red *Docker* interna, reduciendo superficie de ataque.

La implementación aborda los retos tecnológicos fundamentales mediante el stack seleccionado. Los retos organizativos, económicos y de seguridad requieren actuaciones complementarias que, excediendo el alcance académico, quedan documentados como base para despliegues operativos.

4.7.5 Retos de seguridad específicos de IA

Los riesgos de seguridad específicos asociados al uso de modelos de lenguaje se mitigan mediante un conjunto de estrategias complementarias. Entre ellas se incluyen el uso de prompts de sistema robustos y la separación entre instrucciones y datos de usuario para reducir el riesgo de prompt injection, y la restricción de la gestión del corpus del RAG a perfiles administradores para prevenir data poisoning. Asimismo, el uso de RAG contribuye a reducir las alucinaciones al fundamentar las respuestas en documentación controlada y permitir la citación de fuentes. No obstante, las pruebas de seguridad realizadas fueron de carácter básico, quedando pendiente una evaluación adversarial exhaustiva como trabajo futuro previo a cualquier despliegue operativo.

5 CONCLUSIONES

5.1 Consecución de objetivos

El objetivo principal de este trabajo consistía en diseñar, implementar y validar una plataforma de inteligencia artificial offline, adaptada a los requisitos específicos de entornos desconectados de Internet, que permitiese disponer de capacidades funcionales de IA generativa con control local sobre modelos, datos y ejecución.

El prototipo implementado demuestra que resulta técnicamente viable desplegar sistemas de IA generativa en entornos completamente aislados con capacidades suficientes para uso operativo. Los LLM analizados, ejecutándose en una GPU RTX 4090, alcanzaron un rendimiento de 159 tokens/segundo (Mistral 7B), superando ampliamente el objetivo mínimo de 40 tokens/segundo y proporcionando una experiencia de usuario adecuada para tareas de análisis, procesamiento documental y traducción. El ecosistema de herramientas desplegado (Ollama, Open WebUI, ChromaDB) proporciona una base estable y reproducible, mientras que la arquitectura propuesta en tres niveles de despliegue demuestra la escalabilidad de la solución desde puestos individuales aislados hasta infraestructuras departamentales compartidas.

A continuación, se analiza el grado de consecución de cada objetivo específico:

OE1: Analizar el estado del arte de los modelos LLM y las tecnologías asociadas, identificando aquellas aptas para despliegue en entornos completamente desconectados.

Objetivo alcanzado. El Capítulo 2 desarrolla un análisis exhaustivo del ecosistema de modelos de lenguaje abiertos (LLaMA, Mistral, Qwen, Phi), runtimes de inferencia (Ollama, llama.cpp, vLLM), interfaces de usuario (Open WebUI, AnythingLLM), bases de datos vectoriales (ChromaDB, FAISS, Qdrant) y servicios auxiliares (LibreTranslate, Stable Diffusion, Visual Studio Code). El análisis identifica criterios de selección específicos para entornos offline: auditabilidad del código, formato de distribución autocontenido (GGUF), requisitos de hardware, y compatibilidad con cuantización.

OE2: Identificar y estructurar los principales retos para la adopción de IA generativa en entornos aislados.

Objetivo alcanzado. Las secciones 2.9 y 4.7 categorizan sistemáticamente los retos tecnológicos (hardware especializado, actualizaciones offline, obsolescencia), organizativos (resistencia al cambio, formación, nuevos roles), económicos (inversión inicial, coste total de la solución), normativos y de seguridad.

OE3: Diseñar una arquitectura modular y extensible que integre las principales capacidades de IA generativa.

Objetivo alcanzado. El Capítulo 3 presenta una arquitectura de referencia estructurada en tres niveles de despliegue (monopuesto aislado, grupo de trabajo y departamental) que integra todas las capacidades requeridas: inferencia de modelos de lenguaje mediante Ollama, funcionalidad RAG con ChromaDB y BGE-M3, traducción automática multilingüe con LibreTranslate, generación de imágenes con Stable Diffusion, asistencia a programación mediante modelos especializados y extensibilidad mediante MCP para integración con herramientas externas.

OE4: Implementar un prototipo funcional sobre hardware accesible, validando su operación en condiciones de aislamiento completo.

Objetivo alcanzado. El Capítulo 4 documenta la implementación del prototipo sobre hardware de consumo (RTX 4090). El sistema opera sin ninguna conexión a Internet, con modelos, embeddings y dependencias almacenados localmente. Los procedimientos de instalación, configuración y despliegue han quedado documentados para facilitar su reproducibilidad.

OE5: Evaluar el rendimiento y la utilidad mediante pruebas representativas, cuantificando capacidades y limitaciones.

Objetivo alcanzado. El Capítulo 4 incluye una evaluación sistemática que abarca pruebas de rendimiento (velocidad de generación, latencia), calidad de respuestas RAG (comparativa con y sin recuperación), estabilidad del sistema (pruebas básicas de carga) y casos de uso representativos. Se han cuantificado las capacidades y documentado las limitaciones en diferentes escenarios de ejecución.

OE6: Documentar las limitaciones identificadas y proponer líneas de trabajo futuro.

Objetivo alcanzado. Las secciones 5.3 y 5.4 del presente capítulo documentan las restricciones actuales y proponen líneas de evolución para futuros desarrollos.

5.2 Lecciones aprendidas

El desarrollo del trabajo ha permitido extraer conclusiones relevantes para futuros proyectos de despliegue de IA en entornos restringidos.

Para organizaciones que inician su adopción de IA en entornos *air-gapped*, RAG se identifica como la aproximación prioritaria frente a técnicas de *fine-tuning* del modelo. Esta conclusión se sustenta en varios factores validados durante la implementación. En primer lugar, RAG permite mantener los datos sensibles en repositorios documentales controlados, evitando su incorporación directa a los pesos del modelo. En segundo lugar, facilita la actualización del conocimiento sin necesidad de reentrenamiento, lo que resulta especialmente relevante en entornos desconectados donde los procedimientos de transferencia física de modelos o datos son complejos y costosos. En tercer lugar, RAG proporciona trazabilidad explícita de las fuentes utilizadas en cada respuesta, un requisito esencial para auditoría y verificación en contextos regulados. Finalmente, este enfoque requiere significativamente menos recursos computacionales que el *fine-tuning*, tanto en términos de hardware como de tiempo de ejecución.

En este contexto, el *fine-tuning* debe reservarse para escenarios específicos en los que se requiera una modificación profunda y persistente del comportamiento del modelo, una especialización terminológica extrema o la adopción de formatos de respuesta que no pueden lograrse únicamente mediante la provisión de contexto externo.

Adicionalmente, las pruebas realizadas ponen de manifiesto que la calidad de las respuestas generadas mediante RAG depende de forma crítica del diseño del prompt de sistema. Un prompt efectivo debe incorporar, al menos, tres elementos fundamentales: instrucciones explícitas que prioricen el uso del contexto recuperado frente al conocimiento general del modelo, la exigencia de citación de fuentes cuando la información procede de documentos indexados y el requerimiento de reconocimiento explícito cuando una consulta excede el conocimiento disponible en el corpus. En ausencia de estas directrices,

el modelo tiende a combinar conocimiento interno con información recuperada sin una distinción clara, lo que reduce tanto la utilidad práctica como la trazabilidad del sistema.

Se destaca también que la utilidad global de un sistema RAG está directamente condicionada por la calidad y organización del corpus documental indexado. Documentos mal estructurados, con formatos inconsistentes o con información desactualizada degradan la calidad de las respuestas generadas, independientemente de la sofisticación del modelo de lenguaje empleado. En consecuencia, la preparación y ajuste del corpus documental, incluyendo la validación de fuentes, la estructuración homogénea de los contenidos y su actualización periódica, se identifica como una actividad crítica que requiere la asignación de roles y responsabilidades específicas en despliegues operativos.

5.3 Limitaciones identificadas

El trabajo presenta limitaciones en diferentes ámbitos que deben considerarse al interpretar los resultados y al planificar futuras implementaciones.

En primer lugar, no se abordó el entrenamiento completo de modelos de gran tamaño, superiores a 70 mil millones de parámetros, limitándose el trabajo a procesos de *fine-tuning* ligero sobre modelos de entre 7 y 8 mil millones de parámetros mediante técnicas de adaptación eficiente como LoRA. Además, el despliegue se realizó en un entorno de laboratorio, sin integración con infraestructuras de red corporativas clasificadas ni con los controles de seguridad física exigidos para el tratamiento de información clasificada. Asimismo, las pruebas de escalabilidad se restringieron a grupos reducidos de usuarios concurrentes, sin evaluar el comportamiento del sistema bajo cargas sostenidas propias de escenarios con decenas de usuarios simultáneos. Señalar también que no se implementaron mecanismos de alta disponibilidad, redundancia ni balanceo de carga, elementos necesarios para garantizar la continuidad del servicio en entornos de producción con requisitos de operación 24/7.

Más allá de estas restricciones de alcance, el trabajo pone de manifiesto limitaciones inherentes al enfoque de inteligencia artificial offline basado en modelos abiertos. En particular, los modelos desplegados localmente, con tamaños comprendidos aproximadamente entre 7 y 27 mil millones de parámetros, presentan una capacidad inferior para tareas de razonamiento complejo en comparación con los modelos propietarios actuales (Gemini 3, ChatGPT 5.2, etc.). Adicionalmente, el conocimiento incorporado en los modelos de lenguaje permanece congelado en la fecha de su entrenamiento, sin posibilidad de actualización dinámica directa, salvo mediante mecanismos externos de provisión de contexto mediante técnicas RAG. Esta limitación resulta especialmente relevante en dominios caracterizados por una rápida evolución del conocimiento, como la ciberseguridad, los marcos normativos o las tecnologías emergentes.

Finalmente, la evaluación de seguridad realizada fue de alcance básico, por lo que queda pendiente una validación exhaustiva incluyendo técnicas avanzadas de prompt injection, jailbreaking y data poisoning como requisito previo recomendable antes de cualquier despliegue operativo. Del mismo modo, no se ha abordado la certificación formal del sistema ni del hardware asociado.

5.4 Líneas de trabajo futuro

A partir de las limitaciones identificadas y los resultados obtenidos, se proponen las siguientes líneas de evolución para el sistema.

La primera línea contempla la evolución hacia una arquitectura distribuida basada en orquestadores como Kubernetes, lo que permitiría dotar al sistema de capacidades de alta disponibilidad, balanceo de carga y escalabilidad horizontal, requisitos indispensables para garantizar la continuidad del servicio en un entorno operativo real.

La segunda línea se orienta a la expansión del ecosistema de agentes mediante el desarrollo de nuevos conectores bajo el estándar MCP. Esta ampliación extendería las capacidades de la plataforma,

facilitando la interacción controlada y auditable con sistemas externos tales como bases de datos corporativas, herramientas de análisis forense o sistemas de ficheros distribuidos.

Finalmente, antes de considerar cualquier despliegue operativo, resultaría imprescindible la realización de una evaluación de seguridad exhaustiva. Esta evaluación debería incluir auditorías especializadas y pruebas de *red teaming* orientadas específicamente a sistemas de IA, abarcando ataques de *prompt injection*, *jailbreaking* y *data poisoning*, con el objetivo de certificar la robustez del sistema frente a amenazas adversarias.

5.5 Consideraciones finales

El presente trabajo demuestra que es posible disponer de capacidades avanzadas de inteligencia artificial generativa sin exponer información sensible a servicios en la nube. La madurez alcanzada por el ecosistema de software de código abierto, junto con la disponibilidad de hardware de consumo de alto rendimiento, hace viable el despliegue de estas tecnologías en entornos aislados, manteniendo el control operativo y la soberanía del dato.

La plataforma implementada constituye un punto de partida sobre el que pueden desarrollarse soluciones más completas y adaptadas a necesidades específicas. En un contexto en el que la inteligencia artificial se ha consolidado como una tecnología estratégica, resulta cada vez más relevante disponer de capacidades propias, auditables y bajo control organizativo. En este sentido, el trabajo realizado pretende contribuir a dicho objetivo, proporcionando una base técnica y un conjunto de conocimientos directamente aplicables que sirvan como fundamento para desarrollos futuros y evoluciones posteriores de la solución propuesta.

6 BIBLIOGRAFÍA

- [1] K. Hu, “ChatGPT sets record for fastest-growing user base - analyst note.” Feb. 02, 2023. [Online]. Available: <https://www.reuters.com/technology/chatgpt-sets-record-fastest-growing-user-base-analyst-note-2023-02-01/>
- [2] A. M. Turing, “I.—Computing Machinery and Intelligence,” *Mind*, vol. LIX, no. 236, pp. 433–460, Oct. 1950, doi: 10.1093/mind/LIX.236.433.
- [3] J. McCarthy, M. L. Minsky, N. Rochester, and C. E. Shannon, “A Proposal for the Dartmouth Summer Research Project on Artificial Intelligence,” Dartmouth College, Hanover, NH, Project Proposal, 1955. [Online]. Available: <http://jmc.stanford.edu/articles/dartmouth/dartmouth.pdf>
- [4] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet classification with deep convolutional neural networks,” in *Advances in Neural Information Processing Systems*, 2012, pp. 1097–1105.
- [5] A. Vaswani *et al.*, “Attention is All You Need,” in *Advances in Neural Information Processing Systems*, 2017, pp. 5998–6008. [Online]. Available: <https://arxiv.org/abs/1706.03762>
- [6] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding,” presented at the Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Minneapolis, MN, USA, 2019, pp. 4171–4186.
- [7] T. B. Brown *et al.*, “Language Models are Few-Shot Learners,” presented at the Advances in Neural Information Processing Systems, 2020, pp. 1877–1901.
- [8] Anthropic, “Claude 3 Model Card.” Nov. 10, 2025. Accessed: Nov. 10, 2025. [Online]. Available: <https://www.anthropic.com/claude-3-model-card>
- [9] Google DeepMind, “Gemini 3: State-of-the-art reasoning with unprecedented depth and nuance.” Nov. 2025. [Online]. Available: <https://deepmind.google/models/gemini/>
- [10] N. Shazeer *et al.*, “Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer,” in *International Conference on Learning Representations (ICLR)*, 2017. [Online]. Available: <https://arxiv.org/abs/1701.06538>
- [11] Meta AI, “Industry Leading, Open-Source AI | Llama.” Accessed: Jan. 16, 2026. [Online]. Available: <https://www.llama.com/>
- [12] OpenAI, “OpenAI for Countries: Empowering Nations with Sovereign AI and the Stargate Infrastructure.” Accessed: Jan. 16, 2026. [Online]. Available: <https://openai.com/news/openai-for-countries>
- [13] NVIDIA Corporation, “NVIDIA Blackwell Architecture: Empowering the Next Era of Computing and Sovereign AI,” NVIDIA, Whitepaper, 2024. [Online]. Available: <https://www.nvidia.com/en-us/data-center/nvidias-blackwell-architecture-whitepaper/>

- [14] OpenAI, “Introducing Stargate UK.” Accessed: Jan. 16, 2026. [Online]. Available: <https://openai.com/index/introducing-stargate-uk/>
- [15] Palantir Technologies, “Platform Overview: AIP Capabilities,” Palantir Technologies, Technical Documentation, 2024. [Online]. Available: <https://palantir.com/docs/foundry/platform-overview/aip-capabilities/>
- [16] Palantir Technologies, “Palantir Artificial Intelligence Platform (AIP),” 2024. [Online]. Available: <https://www.palantir.com/platforms/aip/>
- [17] Scale AI, “Scale Donovan: AI for Defense,” 2024. [Online]. Available: <https://scale.com/donovan>
- [18] “OpenEuroLLM Project.” European Union. Accessed: Jan. 07, 2026. [Online]. Available: <https://openeurollm.eu>
- [19] Ollama, “Ollama Documentation.” Accessed: Dec. 10, 2025. [Online]. Available: <https://ollama.ai/>
- [20] G. Gerganov, *llama.cpp: Port of Facebook’s LLaMA model in C/C++*. (2025). GitHub. Accessed: Dec. 01, 2025. [Online]. Available: <https://github.com/ggerganov/llama.cpp>
- [21] LM Studio, “LM Studio: Discover, download, and run local LLMs.” Accessed: Dec. 05, 2025. [Online]. Available: <https://lmstudio.ai/>
- [22] E. Luber, *LocalAI: The free, Open Source OpenAI alternative*. 2025. [Online]. Available: <https://localai.io/>
- [23] vLLM Project, “vLLM: Easy, Fast, and Cheap LLM Serving.” 2025. [Online]. Available: <https://github.com/vllm-project/vllm>
- [24] Open WebUI Community, “Open WebUI Documentation.” Accessed: Dec. 10, 2025. [Online]. Available: <https://docs.openwebui.com/>
- [25] AnythingLLM, “AnythingLLM: The all-in-one AI application.” Accessed: Dec. 05, 2025. [Online]. Available: <https://anythingllm.com/>
- [26] *Text Generation Web UI*, Dec. 05, 2025. [Online]. Available: <https://github.com/oobabooga/text-generation-webui>
- [27] H. Touvron *et al.*, “LLaMA: Open and Efficient Foundation Language Models,” arXiv, arXiv:2302.13971, 2023. [Online]. Available: <https://arxiv.org/abs/2302.13971>
- [28] A. Q. Jiang *et al.*, “Mistral 7B,” arXiv, arXiv:2310.06825, 2023. [Online]. Available: <https://arxiv.org/abs/2310.06825>
- [29] Qwen Team, “Qwen2.5 Technical Report,” arXiv, arXiv:2412.15115, 2024. [Online]. Available: <https://arxiv.org/abs/2412.15115>
- [30] D. DeepSeek AI, “DeepSeek-V3 Technical Report,” *arXiv preprint arXiv:2412.19437*, Nov. 2025, [Online]. Available: <https://arxiv.org/abs/2412.19437>
- [31] M. Abdin and others, “Phi-4 Technical Report,” *arXiv preprint arXiv:2412.08905*, 2024, [Online]. Available: <https://arxiv.org/abs/2412.08905>
- [32] Gemma Team, “Gemma: Open Models Based on Gemini Research and Technology,” *arXiv preprint arXiv:2403.08295*, 2024, [Online]. Available: <https://arxiv.org/abs/2403.08295>
- [33] J. Chen, S. Xiao, P. Zhang, K. Luo, D. Lian, and Z. Liu, “BGE M3-Embedding: Multi-Lingual, Multi-Functionality, Multi-Granularity Text Embeddings Through Self-Knowledge Distillation,” arXiv, arXiv:2402.03216, 2024. [Online]. Available: <https://arxiv.org/abs/2402.03216>
- [34] J. Zhang *et al.*, “GTE-Qwen2: A Powerful and Efficient Series of Text Embedding Models,” *arXiv preprint arXiv:2406.14289*, 2024, [Online]. Available: <https://arxiv.org/abs/2406.14289>
- [35] N. Nomic AI, “Nomic Embed: Training a Reproducible Long Context Text Embedder,” *arXiv preprint arXiv:2402.01613*, Nov. 2025, [Online]. Available: <https://arxiv.org/abs/2402.01613>
- [36] P. Lewis *et al.*, “Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks,” presented at the Advances in Neural Information Processing Systems, 2020, pp. 9459–9474.
- [37] Y. Gao and others, “Retrieval-Augmented Generation for Large Language Models: A Survey,” *arXiv preprint arXiv:2312.10997*, 2023, [Online]. Available: <https://arxiv.org/abs/2312.10997>

- [38] Chroma, “ChromaDB: The AI-native open-source embedding database.” 2024. [Online]. Available: <https://www.trychroma.com/>
- [39] J. Johnson, M. Douze, and H. Jégou, “Billion-scale similarity search with GPUs,” *IEEE Transactions on Big Data*, vol. 7, no. 3, pp. 535–547, 2021.
- [40] Qdrant, “Qdrant: Vector Database.” 2024. [Online]. Available: <https://qdrant.tech/>
- [41] Milvus, “Milvus: A cloud-native vector database.” 2024. [Online]. Available: <https://milvus.io/>
- [42] H. Hugging Face, “PEFT: Parameter-Efficient Fine-Tuning.” Nov. 23, 2025. [Online]. Available: <https://huggingface.co/docs/peft>
- [43] E. J. Hu, Y. Shen, P. Wallis, and others, “LoRA: Low-Rank Adaptation of Large Language Models,” in *Proceedings of ICLR*, 2022. [Online]. Available: <https://arxiv.org/abs/2106.09685>
- [44] T. Detrmers, A. Pagnoni, A. Holtzman, and L. Zettlemoyer, “QLoRA: Efficient Finetuning of Quantized LLMs,” in *Advances in Neural Information Processing Systems*, 2023, pp. 10088–10115.
- [45] T. Detrmers, M. Lewis, Y. Belkada, and L. Zettlemoyer, “LLM.int8(): 8-bit Matrix Multiplication for Transformers at Scale,” in *Advances in Neural Information Processing Systems*, Nov. 2025. [Online]. Available: <https://arxiv.org/abs/2208.07339>
- [46] Anthropic, “Model Context Protocol Specification.” Accessed: Nov. 10, 2025. [Online]. Available: <https://modelcontextprotocol.io/>
- [47] *Real Decreto 311/2022, de 3 de mayo, por el que se regula el Esquema Nacional de Seguridad*, vol. 106. 2022. [Online]. Available: <https://www.boe.es/eli/es/rd/2022/05/03/311>
- [48] *Reglamento (UE) 2024/1689 del Parlamento Europeo y del Consejo de 13 de junio de 2024 por el que se establecen normas armonizadas en materia de inteligencia artificial*, vol. L 2024/1689. 2024. [Online]. Available: <https://eur-lex.europa.eu/eli/reg/2024/1689/oj>
- [49] North Atlantic Treaty Organization, “Security within the North Atlantic Treaty Organisation,” Brussels, 2020.
- [50] OWASP, “OWASP Top 10 for Large Language Model Applications.” 2025. [Online]. Available: <https://owasp.org/www-project-top-10-for-large-language-model-applications/>
- [51] K. Greshake, S. Abdelnabi, S. Mishra, and others, “Not what you’ve signed up for: Compromising Real-World LLM-Integrated Applications with Indirect Prompt Injection,” *arXiv preprint arXiv:2302.12173*, 2023, [Online]. Available: <https://arxiv.org/abs/2302.12173>
- [52] E. Hubinger and others, “Sleepers Agents: Training Deceptive LLMs that Persist Through Safety Training,” *arXiv preprint arXiv:2401.05566*, Dec. 2025, [Online]. Available: <https://arxiv.org/abs/2401.05566>
- [53] A. Zou, Z. Wang, J. Z. Kolter, and M. Fredrikson, “Universal and Transferable Adversarial Attacks on Aligned Language Models,” *arXiv preprint arXiv:2307.15043*, 2023, [Online]. Available: <https://arxiv.org/abs/2307.15043>
- [54] LibreTranslate, “LibreTranslate: Free and Open Source Machine Translation API.” Accessed: Dec. 05, 2025. [Online]. Available: <https://libretranslate.com/>
- [55] Stability AI, “Stable Diffusion.” Accessed: Dec. 05, 2025. [Online]. Available: <https://stability.ai/>
- [56] Continue, “Continue: The open-source AI code assistant.” Accessed: Dec. 05, 2025. [Online]. Available: <https://continue.dev/>
- [57] AUTOMATIC1111, *Stable Diffusion Web UI*. (Dec. 17, 2025). [Online]. Available: <https://github.com/automatic1111/stable-diffusion-webui>
- [58] Continue Dev, “Continue Documentation.” Nov. 10, 2025. [Online]. Available: <https://docs.continue.dev/>
- [59] L. Wired, *GhidraMCP: Ghidra Model Context Protocol Server*. (Dec. 05, 2025). GitHub. Accessed: Dec. 05, 2025. [Online]. Available: <https://github.com/LaurieWired/GhidraMCP>
- [60] Hugging Face, “Transformers Documentation.” Nov. 23, 2025. [Online]. Available: <https://huggingface.co/docs/transformers/index>

- [61] T. Dettmers and bitsandbytes contributors, “bitsandbytes Documentation.” Nov. 23, 2025. [Online]. Available: <https://huggingface.co/docs/bitsandbytes/index>
- [62] G. Gerganov, “GGUF: GPT-Generated Unified Format Specification.” Nov. 23, 2025. [Online]. Available: <https://github.com/ggerganov/ggml/blob/master/docs/gguf.md>

ANEXO I: SCRIPT DE DESCARGA DEL CORPUS DOCUMENTAL

```
#!/usr/bin/env python3
# "--api-key" "API_KEY" "--query" "TITLE-ABS-KEY(cybersecurity) AND PUBYEAR > 2022" "--max-results"
# "30" "--pagination" "offset" "--verbose"
import os
import re
import time
import json
import argparse
from urllib.parse import quote, urlparse, parse_qs

import requests
import pandas as pd
from dateutil import parser as dateparser

def sanitize_filename(name: str) -> str:
    name = re.sub(r"[\\/*?\"<>|:]", "_", name)
    return re.sub(r"\s+", " ", name).strip()[:180]

def headers_elsevier(api_key: str, insttoken: str | None = None, accept="application/json"):
    h = {"Accept": accept, "X-ELS-APIKey": api_key.strip()}
    if insttoken:
        h["X-ELS-Insttoken"] = insttoken.strip()
    return h

def save_bytes(path, data: bytes):
    os.makedirs(os.path.dirname(path), exist_ok=True)
    with open(path, "wb") as f:
        f.write(data)
    return path

def get_landing_from_entry(entry: dict):
    fulltext = None
    scopus = None
    for link in entry.get("link", []) or []:
        if link.get("@ref") == "full-text":
            fulltext = link.get("@href")
        if link.get("@ref") == "scopus":
            scopus = link.get("@href")
    return fulltext or scopus

def scopus_search_offset(query: str, api_key: str, insttoken: str | None,
                        max_results=100, count=25, proxies=None, delay=0.2, view="STANDARD",
                        verbose=False):

    url = "https://api.elsevier.com/content/search/scopus"
    start = 0
    all_entries = []

    while True:
        params = {
            "query": query,
            "count": min(25, max(1, count)),
            "start": start,
            "httpAccept": "application/json",
            "view": view,
        }
        r = requests.get(url, headers=headers_elsevier(api_key, insttoken),
                        params=params, proxies=proxies, timeout=60)
        if r.status_code != 200:
            raise RuntimeError(f"[Scopus offset] HTTP {r.status_code}: {r.text[:800]}")
        data = r.json()
        entries = (data.get("search-results", {}) or {}).get("entry", []) or []
        if verbose:
```

```

        print(f"[Scopus offset] start={start} -> {len(entries)} entradas;
total={len(all_entries)}")
        if not entries:
            break
        all_entries.extend(entries)
        if len(all_entries) >= max_results:
            break
        start += params["count"]
        time.sleep(delay)

    return all_entries[:max_results]

def scopus_search_cursor(query: str, api_key: str, insttoken: str | None,
                        max_results=100, count=25, proxies=None, delay=0.2, view="STANDARD",
verbose=False):

    url = "https://api.elsevier.com/content/search/scopus"
    params = {
        "query": query,
        "count": min(25, max(1, count)),
        "httpAccept": "application/json",
        "cursor": "*",
        "view": view,
    }
    all_entries = []
    seen_cursors = set()

    while True:
        r = requests.get(url, headers=headers_elsevier(api_key, insttoken),
                        params=params, proxies=proxies, timeout=60)
        if r.status_code == 403:
            # pasa error hacia arriba para que el caller haga fallback
            raise PermissionError(r.text)
        if r.status_code != 200:
            raise RuntimeError(f"[Scopus cursor] HTTP {r.status_code}: {r.text[:800]}")
        data = r.json()
        entries = (data.get("search-results", {}) or {}).get("entry", []) or []
        if verbose:
            print(f"[Scopus cursor] page -> {len(entries)} entradas; total={len(all_entries)}")
        if not entries:
            break
        all_entries.extend(entries)
        if len(all_entries) >= max_results:
            break

        nxt = None
        for link in (data.get("search-results", {}) or {}).get("link", []) or []:
            if link.get("@ref") == "next":
                nxt = link.get("@href"); break
        if not nxt: break
        c = parse_qs(urlparse(nxt).query).get("cursor", [None])[0]
        if not c or c in seen_cursors: break
        seen_cursors.add(c)
        params["cursor"] = c
        time.sleep(delay)

    return all_entries[:max_results]

def try_pdf_doi(doi, api_key, insttoken, proxies=None, verbose=False):
    if not doi:
        return None, ("NO_DOI", None)
    url = f"https://api.elsevier.com/content/article/doi/{quote(doi)}"
    r = requests.get(
        url,
        headers=headers_elsevier(api_key, insttoken, accept="application/pdf"),
        params={"httpAccept": "application/pdf"},
        proxies=proxies,

```

```

        timeout=60,
        allow_redirects=True,
    )
    if verbose:
        print(f"[DOI] {doi} -> HTTP {r.status_code} CT={r.headers.get('Content-Type')}")
    if r.status_code == 200 and (r.headers.get("Content-Type", "").lower().startswith("application/pdf")):
        return r.content, ("OK", 200)
    return None, ("HTTP", r.status_code)

def try_pdf_pii(pii, api_key, insttoken, proxies=None, verbose=False):
    if not pii:
        return None, ("NO_PII", None)
    url = f"https://api.elsevier.com/content/article/pii/{quote(pii)}"
    r = requests.get(
        url,
        headers=headers_elsevier(api_key, insttoken, accept="application/pdf"),
        params={"httpAccept": "application/pdf"},
        proxies=proxies,
        timeout=60,
        allow_redirects=True,
    )
    if verbose:
        print(f"[PII] {pii} -> HTTP {r.status_code} CT={r.headers.get('Content-Type')}")
    if r.status_code == 200 and (r.headers.get("Content-Type", "").lower().startswith("application/pdf")):
        return r.content, ("OK", 200)
    return None, ("HTTP", r.status_code)

def get_pii_from_eid(eid: str, api_key: str, insttoken: str | None, proxies=None, verbose=False) -> str | None:
    url = f"https://api.elsevier.com/content/abstract/eid/{quote(eid)}"
    r = requests.get(url, headers=headers_elsevier(api_key, insttoken),
        params={"httpAccept": "application/json"},
        proxies=proxies, timeout=60)
    if verbose:
        print(f"[EID] {eid} -> PII lookup HTTP {r.status_code}")
    if r.status_code != 200:
        return None
    data = r.json()
    core = (data.get("abstracts-retrieval-response") or {})
    for path in [
        ("item", "item-info", "pii"),
        ("coredata", "pii"),
    ]:
        try:
            cur = core
            for k in path:
                cur = cur[k]
            if cur: return cur
        except Exception:
            pass
    return None

def save_landing_html(url, title, out_dir, proxies=None, verbose=False):
    if not url:
        return None
    try:
        r = requests.get(url, headers={"Accept": "text/html"},
            proxies=proxies, timeout=60, allow_redirects=True)
        if verbose:
            print(f"[Landing] {url} -> HTTP {r.status_code}")
        if r.status_code == 200:
            fname = os.path.join(out_dir, sanitize_filename(f"{title or 'landing'}.html"))
            save_bytes(fname, r.content)
            return fname
    except Exception as e:

```

```

        if verbose:
            print(f"[Landing] Error: {e}")
        return None
def main():
    parser = argparse.ArgumentParser(description="Buscar en Scopus y descargar PDFs (ScienceDirect)
o landing HTML.")
    parser.add_argument("--api-key", required=True, help="X-ELS-APIKey de Elsevier.")
    parser.add_argument("--insttoken", default=None, help="X-ELS-Insttoken si tu institución lo
requiere.")
    parser.add_argument("--query", required=True, help="Consulta Scopus, p.ej. TITLE-ABS-
KEY(cybersecurity) AND PUBYEAR > 2022")
    parser.add_argument("--count", type=int, default=25, help="Resultados por página (<=25).")
    parser.add_argument("--max-results", type=int, default=50, help="Máximo total de resultados a
recuperar.")
    parser.add_argument("--view", default="STANDARD", choices=["STANDARD", "COMPLETE"], help="Nivel
de detalle (según permisos).")
    parser.add_argument("--output-dir", default="salida_scopus", help="Directorio para PDFs/HTML y
CSV resumen.")
    parser.add_argument("--csv", default="descargas_resumen.csv", help="Nombre del CSV resumen.")
    parser.add_argument("--http-proxy", default=None, help="Proxy HTTP (ej.
http://user:pass@host:port).")
    parser.add_argument("--https-proxy", default=None, help="Proxy HTTPS (ej.
http://user:pass@host:port).")
    parser.add_argument("--delay", type=float, default=0.2, help="Delay entre peticiones (seg).")
    parser.add_argument("--pagination", choices=["offset", "cursor"], default="offset",
                        help="Tipo de paginación (cursor requiere entitlement).")
    parser.add_argument("--verbose", action="store_true", help="Más logs en consola.")
    args = parser.parse_args()

    proxies = None
    if args.http_proxy or args.https_proxy:
        proxies = {}
        if args.http_proxy: proxies["http"] = args.http_proxy
        if args.https_proxy: proxies["https"] = args.https_proxy

    os.makedirs(args.output_dir, exist_ok=True)
    pdf_dir = os.path.join(args.output_dir, "pdfs")
    html_dir = os.path.join(args.output_dir, "html")
    os.makedirs(pdf_dir, exist_ok=True)
    os.makedirs(html_dir, exist_ok=True)

    if args.pagination == "cursor":
        try:
            entries = scopus_search_cursor(
                query=args.query, api_key=args.api_key, insttoken=args.insttoken,
                max_results=args.max_results, count=args.count,
                proxies=proxies, delay=args.delay, view=args.view, verbose=args.verbose
            )
        except PermissionError as e:
            if args.verbose:
                print("[Aviso] Cursor no permitido por tu API key. Haciendo fallback a offset...")
            entries = scopus_search_offset(
                query=args.query, api_key=args.api_key, insttoken=args.insttoken,
                max_results=args.max_results, count=args.count,
                proxies=proxies, delay=args.delay, view=args.view, verbose=args.verbose
            )
    else:
        entries = scopus_search_offset(
            query=args.query, api_key=args.api_key, insttoken=args.insttoken,
            max_results=args.max_results, count=args.count,
            proxies=proxies, delay=args.delay, view=args.view, verbose=args.verbose
        )

    rows = []
    for idx, e in enumerate(entries, 1):
        title = e.get("dc:title")
        doi = e.get("prism:doi")

```

```

eid = e.get("eid") or e.get("dc:identifier")
pub = e.get("prism:publicationName")
date_raw = e.get("prism:coverDate") or e.get("prism:coverDisplayDate")
try:
    date_fmt = dateparser.parse(date_raw).date().isoformat() if date_raw else None
except Exception:
    date_fmt = date_raw

landing_url = get_landing_from_entry(e)

if args.verbose:
    print(f"\n[{idx}/{len(entries)}] {title or 'Sin título'}")

pdf_saved = None
diag_parts = []

if doi:
    pdf, info = try_pdf_doi(doi, args.api_key, args.insttoken, proxies=proxies,
verbose=args.verbose)
    diag_parts.append(f"DOI:{doi}|{info}")
    if pdf:
        fname = os.path.join(pdf_dir, sanitize_filename(f"{title or doi}.pdf"))
        save_bytes(fname, pdf)
        pdf_saved = fname

    if not pdf_saved and eid:
        pii = get_pii_from_eid(eid, args.api_key, args.insttoken, proxies=proxies,
verbose=args.verbose)
        diag_parts.append(f"PII_LOOKUP:{pii or 'NONE'}")
        if pii:
            pdf, info = try_pdf_pii(pii, args.api_key, args.insttoken, proxies=proxies,
verbose=args.verbose)
            diag_parts.append(f"PII_DOWNLOAD:{info}")
            if pdf:
                fname = os.path.join(pdf_dir, sanitize_filename(f"{title or pii}.pdf"))
                save_bytes(fname, pdf)
                pdf_saved = fname

    landing_saved = None
    if not pdf_saved:
        landing_saved = save_landing_html(landing_url, title, html_dir, proxies=proxies,
verbose=args.verbose)

    rows.append({
        "Título": title,
        "Revista": pub,
        "Fecha": date_fmt,
        "DOI": doi,
        "EID": eid,
        "Landing": landing_url or "",
        "PDF guardado": pdf_saved or "",
        "Landing guardada": landing_saved or "",
        "Diag": " ; ".join(diag_parts),
    })

time.sleep(0.1)

df = pd.DataFrame(rows)
if not df.empty and "Fecha" in df.columns:
    df["Fecha"] = pd.to_datetime(df["Fecha"], errors="coerce")
    df.sort_values(["Fecha"], ascending=[False], inplace=True, na_position="last")

csv_path = os.path.join(args.output_dir, args.csv)
df.to_csv(csv_path, index=False, encoding="utf-8")

pd.set_option("display.max_columns", None)
print("\nResumen:\n")

```

```
print(df.to_string(index=False))
print(f"\n CSV: {csv_path}")
print(f" PDFs: {pdf_dir}")
print(f" HTML landings: {html_dir}")

if __name__ == "__main__":
    main()
```

ANEXO II: LIBRETRANSLATE EN OPEN WEBUI

```
{
  "openapi": "3.0.3",
  "info": {
    "title": "LibreTranslate",
    "version": "1.0.0",
    "description": "Traducción offline vía LibreTranslate."
  },
  "servers": [
    {
      "url": "http://libretranslate:5000"
    }
  ],
  "paths": {
    "/languages": {
      "get": {
        "operationId": "languages",
        "summary": "Listar idiomas",
        "responses": {
          "200": {
            "description": "OK",
            "content": {
              "application/json": {
                "schema": {
                  "type": "array",
                  "items": {
                    "type": "object",
                    "properties": {
                      "code": {
                        "type": "string"
                      },
                      "name": {
                        "type": "string"
                      }
                    }
                  },
                  "required": [
                    "code",
                    "name"
                  ]
                }
              }
            }
          }
        }
      }
    },
    "/detect": {
      "post": {
        "operationId": "detect",
        "summary": "Detectar idioma",
        "requestBody": {
          "required": true,
          "content": {
            "application/json": {
              "schema": {
                "type": "object",
                "properties": {
                  "q": {
                    "type": "string"
                  }
                },
                "required": [
                  "q"
                ]
              }
            }
          }
        }
      }
    }
  }
}
```

```

    }
  },
  "responses": {
    "200": {
      "description": "OK",
      "content": {
        "application/json": {
          "schema": {
            "type": "array",
            "items": {
              "type": "object",
              "properties": {
                "language": {
                  "type": "string"
                },
                "confidence": {
                  "type": "number"
                }
              }
            },
            "required": [
              "language",
              "confidence"
            ]
          }
        }
      }
    }
  }
},
"/translate": {
  "post": {
    "operationId": "translate",
    "summary": "Traducir texto",
    "requestBody": {
      "required": true,
      "content": {
        "application/json": {
          "schema": {
            "type": "object",
            "properties": {
              "q": {
                "type": "string"
              },
              "source": {
                "type": "string"
              },
              "target": {
                "type": "string"
              },
              "format": {
                "type": "string",
                "enum": [
                  "text",
                  "html"
                ],
                "default": "text"
              }
            }
          },
          "required": [
            "q",
            "source",
            "target"
          ]
        }
      }
    }
  }
}

```

```

    }
  },
  "responses": {
    "200": {
      "description": "OK",
      "content": {
        "application/json": {
          "schema": {
            "type": "object",
            "properties": {
              "translatedText": {
                "type": "string"
              }
            }
          },
          "required": [
            "translatedText"
          ]
        }
      }
    }
  }
}

```

ANEXO III: SCRIPT DE AUDITORÍA DE OPEN WEBUI

```
#!/usr/bin/env python3
"""
Analizador de Base de Datos Open WebUI
=====
Extrae métricas de rendimiento, auditoría de usuarios, y estadísticas de uso
Uso: python Open WebUI_analyzer.py /ruta/a/webui.db [--output informe.json]
"""

import sqlite3
import json
import sys
from datetime import datetime, timedelta
from pathlib import Path
from collections import defaultdict
import argparse

def conectar_db(ruta_db):
    """Conecta a la base de datos SQLite."""
    if not Path(ruta_db).exists():
        print(f"Error: No se encuentra la base de datos en {ruta_db}")
        sys.exit(1)
    return sqlite3.connect(ruta_db)

def obtener_esquema(conn):
    """Obtiene el esquema completo de la base de datos."""
    cursor = conn.cursor()
    cursor.execute("SELECT name FROM sqlite_master WHERE type='table' ORDER BY name;")
    tablas = cursor.fetchall()

    esquema = {}
    for (tabla,) in tablas:
        # Escapar nombres de tabla con comillas dobles para manejar palabras reservadas
        tabla_escaped = f'"{tabla}"'
        cursor.execute(f"PRAGMA table_info({tabla_escaped});")
        columnas = cursor.fetchall()
        cursor.execute(f"SELECT COUNT(*) FROM {tabla_escaped};")
        count = cursor.fetchone()[0]
        esquema[tabla] = {
            "columnas": [{"id": c[0], "nombre": c[1], "tipo": c[2], "nullable": not c[3], "pk":
bool(c[5])} for c in columnas],
            "registros": count
        }
    return esquema

def analizar_usuarios(conn):
    """Analiza estadísticas de usuarios."""
    cursor = conn.cursor()

    resultado = {
        "total_usuarios": 0,
        "usuarios_activos_ultima_semana": 0,
        "usuarios_activos_ultimo_mes": 0,
        "distribucion_rols": {},
        "usuarios_por_mes_creacion": {},
        "ultimo_acceso_promedio_dias": None
    }

    try:
        # Total usuarios
        cursor.execute('SELECT COUNT(*) FROM "user";')
        resultado["total_usuarios"] = cursor.fetchone()[0]

        # Distribución por roles
```

```

cursor.execute('SELECT role, COUNT(*) FROM "user" GROUP BY role;')
resultado["distribucion_roles"] = dict(cursor.fetchall())

# Usuarios activos (si existe columna last_active_at)
cursor.execute('PRAGMA table_info("user");')
columnas = [c[1] for c in cursor.fetchall()]

if "last_active_at" in columnas:
    ahora = datetime.now().timestamp()
    semana = ahora - (7 * 24 * 3600)
    mes = ahora - (30 * 24 * 3600)

    cursor.execute(f'SELECT COUNT(*) FROM "user" WHERE last_active_at > {semana};')
    resultado["usuarios_activos_ultima_semana"] = cursor.fetchone()[0]

    cursor.execute(f'SELECT COUNT(*) FROM "user" WHERE last_active_at > {mes};')
    resultado["usuarios_activos_ultimo_mes"] = cursor.fetchone()[0]

# Creación por mes
if "created_at" in columnas:
    cursor.execute("""
        SELECT strftime('%Y-%m', datetime(created_at, 'unixepoch')) as mes, COUNT(*)
        FROM "user"
        WHERE created_at IS NOT NULL
        GROUP BY mes
        ORDER BY mes;
    """)
    resultado["usuarios_por_mes_creacion"] = dict(cursor.fetchall())

except Exception as e:
    resultado["error"] = str(e)

return resultado

def analizar_conversaciones(conn):
    """Analiza estadísticas de conversaciones/chats."""
    cursor = conn.cursor()

    resultado = {
        "total_conversaciones": 0,
        "conversaciones_ultima_semana": 0,
        "conversaciones_ultimo_mes": 0,
        "promedio_por_usuario": 0,
        "conversaciones_por_dia": {},
        "usuarios_mas_activos": [],
        "distribucion_mensajes_por_chat": {}
    }

    try:
        # Buscar tabla de chats (puede llamarse chat, conversation, etc.)
        cursor.execute("SELECT name FROM sqlite_master WHERE type='table' AND name LIKE '%chat%';")
        tablas_chat = [t[0] for t in cursor.fetchall()]

        if not tablas_chat:
            cursor.execute("SELECT name FROM sqlite_master WHERE type='table' AND name LIKE '%conversation%';")
            tablas_chat = [t[0] for t in cursor.fetchall()]

        if tablas_chat:
            tabla = tablas_chat[0]
            tabla_escaped = f'"{tabla}"'

            cursor.execute(f'SELECT COUNT(*) FROM {tabla_escaped};')
            resultado["total_conversaciones"] = cursor.fetchone()[0]

            # Analizar estructura
            cursor.execute(f'PRAGMA table_info({tabla_escaped});')

```

```

        columnas = [c[1] for c in cursor.fetchall()]

        # Conversaciones por usuario
        if "user_id" in columnas:
            cursor.execute(f"""
                SELECT user_id, COUNT(*) as num_chats
                FROM {tabla_escaped}
                GROUP BY user_id
                ORDER BY num_chats DESC
                LIMIT 10;
            """)
            resultado["usuarios_mas_activos"] = [{"user_id": r[0], "chats": r[1]} for r in
cursor.fetchall()]

            if resultado["total_conversaciones"] > 0:
                cursor.execute(f"SELECT COUNT(DISTINCT user_id) FROM {tabla_escaped};")
                usuarios_con_chats = cursor.fetchone()[0]
                if usuarios_con_chats > 0:
                    resultado["promedio_por_usuario"] = round(resultado["total_conversaciones"]
/ usuarios_con_chats, 2)

        # Conversaciones por día
        if "created_at" in columnas or "timestamp" in columnas:
            col_fecha = "created_at" if "created_at" in columnas else "timestamp"
            cursor.execute(f"""
                SELECT strftime('%Y-%m-%d', datetime({col_fecha}, 'unixepoch')) as dia, COUNT(*)
                FROM {tabla_escaped}
                WHERE {col_fecha} IS NOT NULL
                GROUP BY dia
                ORDER BY dia DESC
                LIMIT 30;
            """)
            resultado["conversaciones_por_dia"] = dict(cursor.fetchall())

    except Exception as e:
        resultado["error"] = str(e)

    return resultado

def analizar_modelos(conn):
    """Analiza uso de modelos."""
    cursor = conn.cursor()

    resultado = {
        "modelos_disponibles": [],
        "uso_por_modelo": {},
        "modelos_mas_usados": []
    }

    try:
        # Buscar tabla de modelos
        cursor.execute("SELECT name FROM sqlite_master WHERE type='table' AND name LIKE '%model%'")
        tablas_modelo = [t[0] for t in cursor.fetchall()]

        for tabla in tablas_modelo:
            tabla_escaped = f'"{tabla}"'
            cursor.execute(f"PRAGMA table_info({tabla_escaped});")
            columnas = [c[1] for c in cursor.fetchall()]

            if "name" in columnas or "model_id" in columnas or "id" in columnas:
                col_nombre = "name" if "name" in columnas else ("model_id" if "model_id" in columnas
else "id")

                cursor.execute(f"SELECT DISTINCT {col_nombre} FROM {tabla_escaped} LIMIT 50;")
                resultado["modelos_disponibles"].extend([r[0] for r in cursor.fetchall() if r[0]])

        # Buscar uso en chats
        cursor.execute("SELECT name FROM sqlite_master WHERE type='table' AND name LIKE '%chat%'")

```

```

tablas_chat = [t[0] for t in cursor.fetchall()]

for tabla in tablas_chat:
    tabla_escaped = f'"{tabla}"'
    cursor.execute(f"PRAGMA table_info({tabla_escaped});")
    columnas = [c[1] for c in cursor.fetchall()]

    if "model" in columnas or "model_id" in columnas:
        col_modelo = "model" if "model" in columnas else "model_id"
        cursor.execute(f"""
            SELECT {col_modelo}, COUNT(*) as usos
            FROM {tabla_escaped}
            WHERE {col_modelo} IS NOT NULL
            GROUP BY {col_modelo}
            ORDER BY usos DESC;
        """)
        resultado["uso_por_modelo"] = dict(cursor.fetchall())
        resultado["modelos_mas_usados"] = list(resultado["uso_por_modelo"].keys())[0:10]

except Exception as e:
    resultado["error"] = str(e)

return resultado

def analizar_rendimiento(conn):
    """Analiza métricas de rendimiento si están disponibles."""
    cursor = conn.cursor()

    resultado = {
        "metricas_disponibles": False,
        "tiempos_respuesta": {},
        "tokens_procesados": {},
        "errores": {}
    }

    try:
        # Buscar tablas de métricas/logs
        cursor.execute("""
            SELECT name FROM sqlite_master
            WHERE type='table'
            AND (name LIKE '%log%' OR name LIKE '%metric%' OR name LIKE '%stat%');
        """)
        tablas_metricas = [t[0] for t in cursor.fetchall()]

        if tablas_metricas:
            resultado["metricas_disponibles"] = True
            resultado["tablas_encontradas"] = tablas_metricas

            for tabla in tablas_metricas:
                tabla_escaped = f'"{tabla}"'
                cursor.execute(f"PRAGMA table_info({tabla_escaped});")
                columnas = [c[1] for c in cursor.fetchall()]
                cursor.execute(f"SELECT COUNT(*) FROM {tabla_escaped};")
                count = cursor.fetchone()[0]
                resultado[f"tabla_{tabla}"] = {
                    "columnas": columnas,
                    "registros": count
                }

        # Buscar en chats información de tokens/tiempo
        cursor.execute("SELECT name FROM sqlite_master WHERE type='table' AND name LIKE '%chat%';")
        tablas_chat = [t[0] for t in cursor.fetchall()]

        for tabla in tablas_chat:
            tabla_escaped = f'"{tabla}"'
            cursor.execute(f"PRAGMA table_info({tabla_escaped});")
            columnas = [c[1] for c in cursor.fetchall()]

```

```

        # Buscar columnas de rendimiento
        cols_rendimiento = [c for c in columnas if any(x in c.lower() for x in ['token', 'time',
'duration', 'latency'])]
        if cols_rendimiento:
            resultado["columnas_rendimiento_en_chats"] = cols_rendimiento

    except Exception as e:
        resultado["error"] = str(e)

    return resultado

def analizar_rag(conn):
    """Analiza uso de RAG/documentos."""
    cursor = conn.cursor()

    resultado = {
        "rag_habilitado": False,
        "total_documentos": 0,
        "total_colecciones": 0,
        "documentos_por_tipo": {},
        "colecciones": []
    }

    try:
        # Buscar tablas de documentos/RAG
        cursor.execute("""
            SELECT name FROM sqlite_master
            WHERE type='table'
            AND (name LIKE '%document%' OR name LIKE '%file%' OR name LIKE '%collection%' OR name
LIKE '%knowledge%');
        """)
        tablas_rag = [t[0] for t in cursor.fetchall()]

        if tablas_rag:
            resultado["rag_habilitado"] = True
            resultado["tablas_rag"] = tablas_rag

            for tabla in tablas_rag:
                tabla_escaped = f'"{tabla}"'
                cursor.execute(f"SELECT COUNT(*) FROM {tabla_escaped};")
                count = cursor.fetchone()[0]

                if "document" in tabla.lower() or "file" in tabla.lower():
                    resultado["total_documentos"] += count

                cursor.execute(f"PRAGMA table_info({tabla_escaped});")
                columnas = [c[1] for c in cursor.fetchall()]

                if "type" in columnas or "content_type" in columnas or "filename" in columnas:
                    col = next((c for c in ["type", "content_type", "filename"] if c in
columnas), None)
                    if col:
                        cursor.execute(f"SELECT {col}, COUNT(*) FROM {tabla_escaped} GROUP BY
{col};")
                        resultado["documentos_por_tipo"].update(dict(cursor.fetchall()))

                    elif "collection" in tabla.lower():
                        resultado["total_colecciones"] = count
                        cursor.execute(f"SELECT * FROM {tabla_escaped} LIMIT 20;")
                        resultado["colecciones"] = [dict(zip([d[0] for d in cursor.description], row))
for row in cursor.fetchall()]

            except Exception as e:
                resultado["error"] = str(e)

    return resultado

```

```

def analizar_auditoria(conn):
    """Analiza logs de auditoría."""
    cursor = conn.cursor()

    resultado = {
        "auditoria_disponible": False,
        "total_eventos": 0,
        "eventos_por_tipo": {},
        "actividad_por_dia": {},
        "ultimos_eventos": []
    }

    try:
        # Buscar tablas de auditoría
        cursor.execute("""
            SELECT name FROM sqlite_master
            WHERE type='table'
            AND (name LIKE '%audit%' OR name LIKE '%log%' OR name LIKE '%event%' OR name LIKE
'%history%');
        """)
        tablas_audit = [t[0] for t in cursor.fetchall()]

        if tablas_audit:
            resultado["auditoria_disponible"] = True
            resultado["tablas_auditoria"] = tablas_audit

            for tabla in tablas_audit:
                tabla_escaped = f'"{tabla}"'
                cursor.execute(f"SELECT COUNT(*) FROM {tabla_escaped};")
                resultado["total_eventos"] += cursor.fetchone()[0]

                cursor.execute(f"PRAGMA table_info({tabla_escaped});")
                columnas = [c[1] for c in cursor.fetchall()]

                # Eventos por tipo
                if "type" in columnas or "action" in columnas or "event_type" in columnas:
                    col_tipo = next((c for c in ["type", "action", "event_type"] if c in columnas),
None)

                    if col_tipo:
                        cursor.execute(f"SELECT {col_tipo}, COUNT(*) FROM {tabla_escaped} GROUP BY
{col_tipo};")

                        resultado["eventos_por_tipo"].update(dict(cursor.fetchall()))

                # Últimos eventos
                cursor.execute(f"SELECT * FROM {tabla_escaped} ORDER BY rowid DESC LIMIT 10;")
                cols = [d[0] for d in cursor.description]
                for row in cursor.fetchall():
                    resultado["ultimos_eventos"].append(dict(zip(cols, row)))

            except Exception as e:
                resultado["error"] = str(e)

        return resultado

def generar_resumen(analisis):
    resumen = []
    resumen.append("=" * 70)
    resumen.append("ANÁLISIS DE BASE DE DATOS OPEN WEBUI - RESUMEN")
    resumen.append("=" * 70)
    resumen.append(f"Fecha de análisis: {datetime.now().strftime('%Y-%m-%d %H:%M:%S')}")
    resumen.append("")

    # Esquema
    resumen.append("ESTRUCTURA DE LA BASE DE DATOS")
    resumen.append("-" * 40)
    resumen.append(f"Total de tablas: {len(analisis['esquema'])}")

```

```

    resumen.append(f"Total de registros: {sum(t['registros'] for t in
analisis['esquema'].values():,)}")
    resumen.append("")

# Usuarios
u = analisis["usuarios"]
resumen.append("ESTADÍSTICAS DE USUARIOS")
resumen.append("-" * 40)
resumen.append(f"Total usuarios registrados: {u['total_usuarios']}")
resumen.append(f"Usuarios activos última semana: {u['usuarios_activos_ultima_semana']}")
resumen.append(f"Usuarios activos último mes: {u['usuarios_activos_ultimo_mes']}")
if u['distribucion_rols']:
    resumen.append(f"Distribución por roles: {u['distribucion_rols']}")
resumen.append("")

# Conversaciones
c = analisis["conversaciones"]
resumen.append("ESTADÍSTICAS DE CONVERSACIONES")
resumen.append("-" * 40)
resumen.append(f"Total conversaciones: {c['total_conversaciones']}")
resumen.append(f"Promedio por usuario: {c['promedio_por_usuario']}")
if c['conversaciones_por_dia']:
    dias = list(c['conversaciones_por_dia'].keys())
    if dias:
        resumen.append(f"Rango de fechas: {min(dias)} a {max(dias)}")
resumen.append("")

# Modelos
m = analisis["modelos"]
resumen.append("USO DE MODELOS")
resumen.append("-" * 40)
resumen.append(f"Modelos disponibles: {len(m['modelos_disponibles'])}")
if m['modelos_mas_usados']:
    resumen.append(f"Modelos más usados: {'', '.join(m['modelos_mas_usados'][:5])}")
if m['uso_por_modelo']:
    total_inferencias = sum(m['uso_por_modelo'].values())
    resumen.append(f"Total inferencias registradas: {total_inferencias:,}")
resumen.append("")

# RAG
r = analisis["rag"]
resumen.append("SISTEMA RAG")
resumen.append("-" * 40)
resumen.append(f"RAG habilitado: {'Sí' if r['rag_habilitado'] else 'No'}")
if r['rag_habilitado']:
    resumen.append(f"Total documentos: {r['total_documentos']}")
    resumen.append(f"Total colecciones: {r['total_colecciones']}")
    if r['documentos_por_tipo']:
        resumen.append(f"Tipos de documentos: {r['documentos_por_tipo']}")
resumen.append("")

# Auditoría
a = analisis["auditoria"]
resumen.append("AUDITORÍA Y LOGS")
resumen.append("-" * 40)
resumen.append(f"Auditoría disponible: {'Sí' if a['auditoria_disponible'] else 'No'}")
if a['auditoria_disponible']:
    resumen.append(f"Total eventos registrados: {a['total_eventos']}")
    if a['eventos_por_tipo']:
        resumen.append(f"Tipos de eventos: {list(a['eventos_por_tipo'].keys())}")
resumen.append("")

resumen.append("=" * 70)

return "\n".join(resumen)

def main():

```

```

    parser = argparse.ArgumentParser(description="Analizador de Base de Datos Open WebUI")
    parser.add_argument("database", help="Ruta a la base de datos SQLite de Open WebUI")
    parser.add_argument("--output", "-o", help="Archivo de salida para el informe JSON",
default="Open WebUI_analysis.json")
    parser.add_argument("--resumen", "-r", help="Generar solo resumen de texto",
action="store_true")
    args = parser.parse_args()

    print(f"Analizando base de datos: {args.database}")
    print("-" * 50)

    conn = conectar_db(args.database)

    analisis = {
        "metadata": {
            "base_datos": args.database,
            "fecha_analisis": datetime.now().isoformat(),
            "tamaño_bytes": Path(args.database).stat().st_size
        },
        "esquema": obtener_esquema(conn),
        "usuarios": analizar_usuarios(conn),
        "conversaciones": analizar_conversaciones(conn),
        "modelos": analizar_modelos(conn),
        "rendimiento": analizar_rendimiento(conn),
        "rag": analizar_rag(conn),
        "auditoria": analizar_auditoria(conn)
    }

    conn.close()

    # Generar resumen de texto
    resumen = generar_resumen(analisis)
    print(resumen)

    if not args.resumen:
        # Guardar JSON completo
        with open(args.output, 'w', encoding='utf-8') as f:
            json.dump(analisis, f, indent=2, ensure_ascii=False, default=str)
        print(f"\nInforme JSON completo guardado en: {args.output}")

        # Guardar resumen de texto
        resumen_file = args.output.replace('.json', '_resumen.txt')
        with open(resumen_file, 'w', encoding='utf-8') as f:
            f.write(resumen)
        print(f"Resumen de texto guardado en: {resumen_file}")

if __name__ == "__main__":
    main()

```

ANEXO IV: FINE-TUNING DE MISTRAL

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Fine-tuning ligero (QLoRA) del modelo Mistral-7B-Instruct-v0.2
"""

import os
os.environ["PYTORCH_ALLOC_CONF"] = "expandable_segments:True"

from datasets import load_dataset
from transformers import (
    AutoModelForCausalLM,
    AutoTokenizer,
    TrainingArguments,
    Trainer,
    BitsAndBytesConfig,
)
from peft import LoraConfig, get_peft_model, prepare_model_for_kbit_training
import torch

# -----
# CONFIGURACIÓN PRINCIPAL
# -----
MODEL_NAME = "mistralai/Mistral-7B-Instruct-v0.2"
DATA_PATH = "dataset_ciber_300.jsonl" # tu dataset
OUTPUT_DIR = "./mistral-ciber-lora"

EPOCHS = 2
BATCH_SIZE = 1
GRAD_ACCUM = 16
LR = 2e-4
MAX_LENGTH = 1024

# -----
# CARGA DE DATOS
# -----
print("Cargando dataset...")
dataset = load_dataset("json", data_files=DATA_PATH)["train"]
dataset = dataset.train_test_split(test_size=0.1, seed=42)
train_ds, eval_ds = dataset["train"], dataset["test"]

# -----
# TOKENIZADOR
# -----
tokenizer = AutoTokenizer.from_pretrained(MODEL_NAME)
if tokenizer.pad_token is None:
    tokenizer.pad_token = tokenizer.eos_token

def format_prompt(example):
    prompt = (
        f"### Instrucción:\n{example['instruction']}\n\n"
        f"### Entrada:\n{example['input']}\n\n"
        f"### Respuesta:\n{example['output']}"
    )
    t = tokenizer(
        prompt,
        max_length=MAX_LENGTH,
        truncation=True,
        padding="max_length",
    )
    t["labels"] = t["input_ids"].copy()
    return t

print("Tokenizando ejemplos...")
train_tok = train_ds.map(format_prompt)
```

```

eval_tok = eval_ds.map(format_prompt)
train_tok.set_format("torch")
eval_tok.set_format("torch")

# -----
# CONFIGURACIÓN QLoRA 4-bit
# -----
bnb_config = BitsAndBytesConfig(
    load_in_4bit=True,
    bnb_4bit_use_double_quant=True,
    bnb_4bit_quant_type="nf4",
    bnb_4bit_compute_dtype=torch.bfloat16,
)

# -----
# CARGA DEL MODELO BASE
# -----
print("Cargando modelo Mistral-7B-Instruct-v0.2 en 4 bits...")
model = AutoModelForCausalLM.from_pretrained(
    MODEL_NAME,
    quantization_config=bnb_config,
    device_map="auto",
    torch_dtype=torch.bfloat16,
)

# Muy importante para QLoRA:
model = prepare_model_for_kbit_training(model)

# -----
# CONFIGURACIÓN LoRA
# -----
peft_config = LoraConfig(
    r=16,
    lora_alpha=32,
    lora_dropout=0.05,
    target_modules=[
        "q_proj", "k_proj", "v_proj", "o_proj",
        "gate_proj", "up_proj", "down_proj",
    ],
    task_type="CAUSAL_LM",
)

model = get_peft_model(model, peft_config)

# Desactivar cache para gradient checkpointing
model.config.use_cache = False
# Activar gradientes en las entradas
model.enable_input_require_grads()
# Ahorrar VRAM
model.gradient_checkpointing_enable()

# (Opcional) Ver cuántos parámetros se van a entrenar
model.print_trainable_parameters()

# -----
# ARGUMENTOS DE ENTRENAMIENTO
# -----
print("Configurando entrenamiento LoRA...")
args = TrainingArguments(
    output_dir=OUTPUT_DIR,
    per_device_train_batch_size=BATCH_SIZE,
    per_device_eval_batch_size=BATCH_SIZE,
    gradient_accumulation_steps=GRAD_ACCUM,
    learning_rate=LR,
    num_train_epochs=EPOCHS,
    bf16=True,
    logging_steps=10,

```

```
        save_strategy="epoch",
        warmup_ratio=0.05,
        lr_scheduler_type="cosine",
        report_to=[],
    )

    trainer = Trainer(
        model=model,
        args=args,
        train_dataset=train_tok,
        eval_dataset=eval_tok,
    )

    torch.cuda.empty_cache()
    print("Iniciando entrenamiento...")
    trainer.train()

    print("Guardando adaptador LoRA...")
    model.save_pretrained(OUTPUT_DIR)
    tokenizer.save_pretrained(OUTPUT_DIR)
    print("Entrenamiento completado. Adaptador guardado en:", OUTPUT_DIR)
```

ANEXO V: EVALUACIÓN DEL MODELO ADAPTADO

```

from transformers import AutoTokenizer, AutoModelForCausalLM
from peft import PeftModel
import torch

# Rutas
BASE_MODEL = "mistralai/Mistral-7B-Instruct-v0.2"
ADAPTER_DIR = "./mistral-ciber-lora"

# Carga modelo base y adaptador
print("Cargando modelo base y adaptador LoRA...")
tokenizer = AutoTokenizer.from_pretrained(BASE_MODEL)
model = AutoModelForCausalLM.from_pretrained(
    BASE_MODEL,
    device_map="auto",
    torch_dtype=torch.bfloat16,
)
model = PeftModel.from_pretrained(model, ADAPTER_DIR)

# Función helper
def chat(prompt):
    inputs = tokenizer(prompt, return_tensors="pt").to("cuda")
    output = model.generate(
        **inputs,
        max_new_tokens=256,
        temperature=0.3,
        do_sample=False,
    )
    print(tokenizer.decode(output[0], skip_special_tokens=True))

# Ejemplo 1 – Clasificación MITRE
prompt = """Instrucción:
Clasifica la siguiente actividad sospechosa según MITRE ATT&CK. Devuelve táctica y técnica.

### Entrada:
El atacante usa PowerShell para ejecutar scripts descargados desde una URL remota y establecer
persistencia.

### Respuesta: """
chat(prompt)

```

ANEXO VI: INTEGRACIÓN DE LORA CON MISTRAL

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Fusiona el adaptador LoRA con el modelo base Mistral-7B-Instruct-v0.2
para obtener un modelo completo "mistral-ciber-merged".
"""

from transformers import AutoModelForCausalLM, AutoTokenizer
from peft import PeftModel

# Rutas
BASE_MODEL = "mistralai/Mistral-7B-Instruct-v0.2"
ADAPTER_DIR = "./mistral-ciber-lora"
OUTPUT_DIR = "./mistral-ciber-merged"

print("Cargando modelo base y adaptador...")
tokenizer = AutoTokenizer.from_pretrained(BASE_MODEL)

# Carga del modelo base
model = AutoModelForCausalLM.from_pretrained(
    BASE_MODEL,
    device_map="auto",
    torch_dtype="bfloat16"
)

# Carga del adaptador LoRA
model = PeftModel.from_pretrained(model, ADAPTER_DIR)

print("Fusionando adaptador LoRA con el modelo base...")
merged_model = model.merge_and_unload()

print("Guardando modelo fusionado...")
merged_model.save_pretrained(OUTPUT_DIR)
tokenizer.save_pretrained(OUTPUT_DIR)

print("Modelo final guardado en:", OUTPUT_DIR)
```

ANEXO VII: ACRÓNIMOS

AIP – Artificial Intelligence Platform
API – Application Programming Interface
BERT – Bidirectional Encoder Representations from Transformers
BGE – BAAI General Embedding
BM25 – Best Matching 25 (algoritmo de ranking)
CESGA – Centro de Supercomputación de Galicia
CUDA – Compute Unified Device Architecture
ENS – Esquema Nacional de Seguridad
FAISS – Facebook AI Similarity Search
FP16/32 – Floating Point 16/32-bit
GGUF – GPT-Generated Unified Format
GPT – Generative Pre-trained Transformer
GPU – Graphics Processing Unit
GTE – General Text Embeddings
HPC – High Performance Computing
IA – Inteligencia Artificial
LLM – Large Language Model
LoRA – Low-Rank Adaptation
MCP – Model Context Protocol
MoE – Mixture of Experts
OCR – Optical Character Recognition
OWASP – Open Web Application Security Project
PEFT – Parameter-Efficient Fine-Tuning
QLoRA – Quantized Low-Rank Adaptation
RAG – Retrieval-Augmented Generation
RBAC – Role-Based Access Control
RGPD – Reglamento General de Protección de Datos
SaaS – Software as a Service
SSD – Solid State Drive
TPU – Tensor Processing
VRAM – Video Random Access Memory