



Centro Universitario de la Defensa en la Escuela Naval Militar

TRABAJO FIN DE MÁSTER

Aplicación de técnicas de inteligencia artificial a la gestión documental organizativa

Máster Universitario en Dirección TIC para la Defensa

ALUMNO: Pablo García Martín

DIRECTORES: Norberto Fernández García
Francisco Troncoso Pastoriza

CURSO ACADÉMICO: 2025-2026

Universida_{de}Vigo



Centro Universitario de la Defensa en la Escuela Naval Militar

TRABAJO FIN DE MÁSTER

*Aplicación de técnicas de inteligencia artificial a la gestión
documental organizativa*

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

UniversidadeVigo

RESUMEN

La gestión documental en entornos de defensa enfrenta heterogeneidad de formatos, requisitos estrictos de seguridad y soberanía del dato, y complejidad operativa que demanda comprensión semántica sobre documentación técnica y normativa distribuida.

Este trabajo diseña una arquitectura de gestión documental basada en inteligencia artificial y valida su viabilidad en un sistema de gestión de municiones para la Armada Española. La propuesta integra procesamiento de *lenguaje* natural, recuperación aumentada por generación (RAG) y búsqueda semántica mediante *embeddings*

Se propone una arquitectura híbrida que separa componentes generativos de módulos deterministas para validaciones críticas: las consultas sobre inventario y restricciones de seguridad se resuelven mediante SQL contra bases de datos relacionales, mientras que la comprensión de normativa técnica emplea búsqueda vectorial semántica.

El trabajo desarrolla una revisión del estado del arte en técnicas de IA aplicables a gestión documental, especifica una arquitectura de referencia basada en microservicios con requisitos de seguridad para entornos clasificados, y valida estos conceptos en un prototipo operativo implementado con Python/Streamlit y modelos de lenguaje ejecutados *on-premise* mediante Ollama. El prototipo demuestra empíricamente la viabilidad del patrón híbrido RAG+SQL y valida el uso de LLMs en entornos de alta criticidad sin comprometer la soberanía tecnológica.

Como trabajo de dirección TIC, la contribución se centra en el diseño arquitectónico conceptual validado con la implementación práctica de un prototipo, además se proporciona una guía para el escalado de dicho a fase de producción, con métricas de evaluación específicas.

PALABRAS CLAVE

Inteligencia artificial, gestión documental, RAG, procesamiento de lenguaje natural, arquitectura de microservicios, búsqueda semántica

AGRADECIMIENTOS

Quiero expresar mi agradecimiento, a mis tutores por la orientación y el rigor aportados durante todo el proceso, así como por las observaciones que han permitido mejorar la calidad de este trabajo. Agradezco también a los compañeros que, desde el ámbito profesional, han facilitado contexto, criterio y apoyo para enfocar adecuadamente el caso de estudio. Por último, doy las gracias a mi familia y mi pareja por su paciencia, comprensión y ánimo constante, especialmente en las fases de mayor exigencia.

CONTENIDO

Contenido	1
Índice de Figuras	4
Índice de Tablas.....	6
1 Introducción y objetivos.....	7
1.1 Gestión documental organizativa.....	7
1.2 Gestión documental en entornos de alta criticidad	8
1.3 Caso específico de estudio	8
1.4 Objetivos	9
1.5 Alcance.....	9
2 Estado del arte: técnicas de IA aplicadas a la gestión documental	11
2.1 Conceptos clave de Inteligencia Artificial	11
2.1.1 Conceptos clave del Machine Learning	13
2.1.2 Paradigmas de aprendizaje.....	14
2.1.3 Aplicación de IA en entornos profesionales	14
2.1.4 Aplicación de técnicas de IA a la gestión documental organizativa.....	15
2.2 Extracción y preprocesamiento de información.....	17
2.2.1 Técnicas de OCR	17
2.2.2 Extracción de información estructurada	18
2.2.3 Preprocesamiento de datos logísticos	21
2.3 Procesamiento de lenguaje natural.....	22
2.3.1 Representación de texto	22
2.3.2 PLN Profundo	24
2.3.3 Generación de Lenguaje Natural.....	27
2.3.4 Ingeniería de prompts.....	29
2.3.5 Sistemas Multiagentes.....	31
2.4 Recuperación de Información	32
2.4.1 Motores de búsqueda inteligentes y ajuste fino	32
2.4.2 Búsqueda Semántica	33
2.4.3 Generación Aumentada por Recuperación (RAG)	33
2.5 Almacenamiento y Bases de Datos	36
2.5.1 Bases de Datos Relacionales.....	36
2.5.2 Bases de datos NoSQL.....	37
2.5.3 Bases de Datos Vectoriales	39

3	Arquitectura de la solución	44
3.1	Paradigma arquitectónico: monolito y microservicios.....	44
3.1.1	Elección arquitectónica	45
3.1.2	Principios de diseño fundamentales	48
3.1.3	Patrones de comunicación entre microservicios	50
3.1.4	Escalabilidad	51
3.2	Preprocesamiento	51
3.3	Almacenamiento y bases de datos	52
3.3.1	Estrategias de indexado en bases de datos vectoriales.....	53
3.3.2	Base de datos relacional	53
3.4	LLMs y RAG	54
3.4.1	Elección de LLM	54
3.4.2	Arquitectura de Generación Aumentada por Recuperación (RAG)	55
3.4.3	Representaciones semánticas mediante Embeddings	56
3.4.4	Ingeniería de prompts y control de generación	56
3.4.5	Limitaciones y estrategias de mitigación	57
3.5	Proceso de Ingesta.....	58
3.5.1	Control de acceso y autenticación.....	58
3.5.2	Fragmentación semántica de documentos (Chunking)	59
3.5.3	Clasificación automática de contenido	59
3.5.4	Generación de representaciones vectoriales	59
3.5.5	Enriquecimiento con metadatos	60
3.5.6	Extracción estructurada de información	60
3.6	Proceso de Recuperación	60
3.6.1	Orquestador central (LLM Router).....	61
3.6.2	Agente de recuperación semántica.....	61
3.6.3	Agente de consultas estructuradas	61
3.6.4	Módulos Deterministas de Validación y Optimización	62
3.6.5	Generador con Anclaje Contextual	62
3.7	Flujo de procesamiento	62
3.7.1	Consulta	62
3.7.2	Ingesta de documentos	64
3.8	Seguridad de LLMs, y sistemas RAG.....	66
3.8.1	Superficie de amenazas y vulnerabilidades específicas de LLM.....	66
3.8.2	Mecanismos de autenticación y cifrado en sistemas de IA generativa	67
3.8.3	Despliegue on-premise versus cloud: soberanía y cumplimiento.....	68

3.8.4	Marcos de gobernanza y estándares de IA	69
4	Caso de estudio: logística en defensa.....	70
4.1	Panorámica general	70
4.1.1	Situación actual	70
4.1.2	Solución propuesta.....	72
4.2	Prototipo de validación (PoC).....	79
4.2.1	Alcance del prototipo	79
4.2.2	Mapeo de componentes.....	80
4.2.3	Integración de componentes y flujos end-to-end	82
4.2.4	Tecnologías utilizadas en la PoC	82
4.3	Estrategia de escalado: hacia una arquitectura operativa.....	83
4.3.1	De monolito a microservicios: descomposición funcional.	85
4.3.2	Orquestación distribuida (Sagas) y gestión transaccional.....	87
4.3.3	Estrategias de observabilidad y resiliencia.	88
4.3.4	Interfaces de operación y experiencia de usuario	90
4.4	Marco de evaluación: metodología y pruebas.....	91
4.4.1	Métricas de recuperación y generación.....	91
4.4.2	Plan de pruebas funcionales y de seguridad.....	94
4.4.3	Análisis de riesgos y KPIs de impacto operativo.....	95
5	Conclusiones	98
5.1	Contribuciones principales.....	99
5.2	Limitaciones y consideraciones	100
5.3	Líneas futuras de investigación y desarrollo.....	101
5.4	Reflexión final.....	102
6	Bibliografía	104
	Anexo I. Prototipo SIGLM: casos de uso.....	110
	Anexo II. Código prototipo SIGLM.....	124
	Anexo III: Glosario	179

ÍNDICE DE FIGURAS

Figura 1-1. Alcance y flujo de trabajo del TFM.....	10
Figura 2-1. Línea del tiempo con principales hitos de la IA (elaboración propia).....	12
Figura 2-2. Jerarquía y subconjuntos en el campo de la IA.	13
Figura 2-3. Comparación de arquitectura: Red Neuronal Simple vs. Red de Aprendizaje Profundo [12].....	14
Figura 2-4: Estrategias de entrenamiento para Word2Vec [19].	23
Figura 2-5. Visualización de arquitectura <i>feed-forward</i> y recursiva [22]	25
Figura 2-6. Arquitectura de los transformadores [8].	26
Figura 2-7. Evolución del número de parámetros en modelos de lenguaje basados en Transformers [26].....	28
Figura 2-8. Esquema de funcionamiento de Generación Aumentada por Recuperación (RAG) [36].	34
Figura 3-1. Representación comparativa de alto nivel entre arquitectura monolítica y microservicios [42].....	45
Figura 3-2. Representación de la comunicación síncrona y asíncrona (elaboración propia).	51
Figura 3-3. Diagrama de flujo de consultas (elaboración propia).	63
Figura 3-4. Diagrama de flujo de inserción de documentos (elaboración propia).	65
Figura 4-1. Arquitectura simplificada del <i>backend</i> del SIGLM mostrando los microservicios principales (elaboración propia).....	86
Figura 4-2. Patrón Saga para orquestación de ingesta documental (elaboración propia).	88
Figura 4-3. Arquitectura modular del <i>frontend</i> SIGLM basada en Angular (elaboración propia)...	90
Figura A-1. Ejemplo de consulta genérica al LLM sin RAG.....	111
Figura A-2. Consulta sobre municiones, por restricciones requiere RAG, pero no hay documentación cargada.	111
Figura A-3. Búsqueda RAG de información sobre municiones. En el panel izquierdo se observa cómo realiza la búsqueda.	113
Figura A-4. Respuesta a la consulta RAG sobre municiones.....	114
Figura A-5. Respuesta a consulta RAG sobre normativa.....	114
Figura A-6. Tras una orden mediante chat de inserción en SQL, el sistema captura los datos recibidos, y trata de buscar por RAG los faltantes.....	116
Figura A-7. Inserción de datos en base SQL mediante chatbot.	117
Figura A-8. Movimiento rechazado por restricciones de seguridad.	117
Figura A-9. Extracción de munición de base SQL, una satisfactoria y otra fallida.	118
Figura A-10. Detección automática de documentos con movimientos de munición.	121
Figura A-11. Ejecutar movimiento desde documento.....	121

ÍNDICE DE TABLAS

Tabla 3-1. Etapas para el preprocesamiento de documentación. 51

Tabla 3-2. Fases de preprocesamiento para cada tipo de documento. 52

Tabla 3-3. Potenciales riesgos de los LLM y estrategias de mitigación 58

Tabla 4-1. Beneficios Esperados e Indicadores de Impacto del SIGLM 77

Tabla 4-2. Tecnologías utilizadas en la PoC, frente a las propuestas para la arquitectura en producción.....84

Tabla 4-3. Descomposición funcional desde la PoC monolítica a la arquitectura propuesta de microservicios. 85

Tabla 4-4. Métricas clave de recuperación y generación 92

Tabla 4-5. Matriz de Riesgos Críticos y Estrategias de Mitigación 95

Tabla 4-6. KPIs Operativos y Métricas de Impacto 96

Tabla 5-1. Hoja de ruta desde componente conceptual, a prototipo y propuesta de escalado a producción.....100

1 INTRODUCCIÓN Y OBJETIVOS

La Inteligencia Artificial (IA) y el *Machine Learning* (ML) han transformado el mundo en los últimos años. Las nuevas tecnologías de IA generativa, están teniendo un impacto directo en la gestión de documentos y en la optimización de operaciones organizativas. En este entorno cambiante, la presión por decidir con rapidez y rigor aumenta la demanda de sistemas capaces de analizar, comprender y extraer valor de grandes volúmenes de datos. La IA ya no se concibe únicamente como automatización de tareas rutinarias: su objetivo es elevar la eficiencia, reducir los tiempos de respuesta y mejorar la calidad de la decisión, dotando a las organizaciones de herramientas para convertir información dispersa en conocimiento accionable.

En este contexto, resulta necesario explorar el uso de IA en la gestión documental de organizaciones del ámbito de la Defensa. La combinación de exigencias regulatorias, sensibilidad de la información y complejidad operativa hace que los procesos tradicionales —a menudo manuales, fragmentados y poco escalables— se queden cortos. Tecnologías como el reconocimiento óptico de caracteres (*Optical Character Recognition*, OCR), el procesamiento del lenguaje natural (PLN), los modelos de lenguaje de gran tamaño (*Large Language Model*, LLMs) y la IA generativa permiten automatizar la clasificación y el enriquecimiento documental, extraer entidades clave, habilitar búsqueda semántica y apoyar la validación frente a normas y catálogos maestros.

En entornos de alta criticidad estas herramientas pueden reducir errores, optimizar tiempos de respuesta y facilitar la toma de decisiones. La presente memoria aborda, por tanto, el diseño y aplicación de una solución basada en técnicas de IA orientadas a transformar este proceso documental crítico.

1.1 Gestión documental organizativa

La gestión documental organizativa comprende el conjunto de políticas, procesos y tecnologías que gobiernan el ciclo de vida de los documentos y registros de una entidad, desde su creación o captura hasta su conservación y disposición. Su finalidad es garantizar autenticidad, integridad, disponibilidad, confidencialidad y trazabilidad, alineando la documentación con los objetivos de la organización, la normativa aplicable y las necesidades de auditoría. En la práctica, esto implica definir metadatos y taxonomías, establecer flujos de revisión y aprobación, asegurar el control de versiones y aplicar controles de acceso y evidencias verificables de las acciones realizadas sobre cada documento.

En los últimos años, la gestión documental evoluciona desde repositorios “pasivos” hacia enfoques cognitivos, donde técnicas de IA permiten comprender el contenido, extraer información operativa y facilitar consultas en lenguaje natural. Esta evolución aporta ventajas claras —reducción de tiempos de búsqueda, mejora de calidad de datos y automatización de verificaciones—, pero en entornos sensibles exige reforzar soberanía del dato, trazabilidad de decisiones y compatibilidad con sistemas legados.

Las capacidades de IA se aplican de forma transversal: (i) en la ingesta, mediante OCR y comprensión de documentos; (ii) en la descripción, con clasificación automática y extracción de entidades relevantes; (iii) en la recuperación, con búsqueda semántica e híbrida; y (iv) en el aseguramiento, detectando incoherencias y apoyando verificaciones de cumplimiento con mecanismos de “humano en el bucle” cuando la confianza no alcanza umbrales definidos. Para entornos auditables, la combinación de semántica, reglas deterministas y registro de evidencias permite respuestas fundamentadas y replicables, reforzando la gobernanza de la información.

1.2 Gestión documental en entornos de alta criticidad

Los entornos de alta criticidad (defensa, aeroespacial, energía o sanidad) imponen requisitos documentales que elevan la complejidad y condicionan la arquitectura: heterogeneidad documental, seguridad/soberanía del dato y complejidad operativa. En estos contextos, la gestión documental no es un apoyo administrativo, sino un componente habilitador de la seguridad, la continuidad de misión y la capacidad de decisión.

Heterogeneidad documental. La información convive en formatos y calidades muy dispares: papel digitalizado, PDFs estructurados y no estructurados, hojas de cálculo, bases de datos heredadas y texto libre. Esto obliga a incorporar mecanismos robustos de normalización, clasificación y extracción que mantengan coherencia semántica y trazabilidad desde el origen hasta el uso operativo.

Seguridad y soberanía del dato. El tratamiento de información sensible exige control estricto de localización, acceso y auditoría. En consecuencia, el despliegue suele requerir infraestructuras propias (*on-premise*), segregación por niveles de clasificación, controles granulares basados en mínimo privilegio y registros de auditoría completos e inmutables, con mecanismos criptográficos que garanticen integridad y no repudio. Además, el diseño debe contemplar desde el inicio el cumplimiento de marcos normativos aplicables.

Complejidad operativa. A diferencia de escenarios corporativos convencionales, aquí los documentos alimentan procesos transaccionales y decisiones con restricciones del dominio. Es necesario sincronizar información documental con bases de datos en evolución continua, validar reglas y restricciones deterministas antes de autorizar operaciones y, cuando procede, integrar capacidades de optimización para asignación y planificación bajo múltiples limitaciones. Todo ello debe ofrecer explicaciones comprensibles y supervisión humana en operaciones sensibles.

En conjunto, estas condiciones justifican el papel habilitador de la IA —OCR avanzado, PLN, búsqueda híbrida y RAG— siempre integrada con controles deterministas y gobierno del dato, de forma que la automatización incremente eficiencia sin comprometer seguridad, trazabilidad ni fiabilidad operativa.

1.3 Caso específico de estudio

Para validar la arquitectura propuesta, este TFM se apoya en un caso de estudio particularmente exigente: la gestión logística de municiones en la Armada Española. La elección de este dominio no es casual. Se trata de un entorno donde convergen, en su máxima expresión, todos los desafíos identificados previamente: la documentación abarca desde normativa OTAN multilingüe y manuales técnicos hasta albaranes manuscritos digitalizados, registros de inspección y comunicaciones operativas; la información está frecuentemente clasificada, exigiendo despliegue exclusivamente *on-premise* bajo control total de la organización; y las operaciones están sometidas a restricciones físico-químicas estrictas —incompatibilidades entre grupos de riesgo, límites de NEQ (*Net Explosive Quantity*), distancias de seguridad (QD), condiciones ambientales— que hacen inviable cualquier margen de error.

Lo que distingue a este caso no es solo su complejidad, sino su criticidad. Un fallo en la gestión documental no se traduce únicamente en ineficiencia administrativa: puede comprometer afectar a la disponibilidad operativa de unidades, generar situaciones de riesgo por almacenamiento incompatible o

impedir la trazabilidad. La pérdida de un albarán de transferencia, la consulta errónea de una especificación técnica o la validación incorrecta de una compatibilidad entre municiones pueden tener consecuencias que trascienden lo operativo.

Desde el punto de vista arquitectónico, este escenario constituye un "banco de pruebas natural" para la solución propuesta. Si la combinación de RAG, bases de datos estructuradas y validación determinista es capaz de operar con fiabilidad en este contexto —donde la heterogeneidad documental, la seguridad del dato y la precisión operativa son no negociables— su aplicabilidad a dominios menos exigentes quedaría implícitamente validada. Además, el diseño se ha concebido de forma modular y parametrizable: aunque el prototipo se centra en municiones, la arquitectura permite su transferencia a otros sectores críticos mediante la adaptación del modelo de dominio, las reglas de validación y las ontologías específicas.

El documento avanza de forma progresiva: tras revisar el estado del arte de las tecnologías empleadas (capítulo 2), se presenta la arquitectura genérica (capítulo 3) y su aplicación concreta al caso de estudio (capítulo 4), donde se detallan los flujos operativos, los criterios de evaluación y los beneficios esperados en términos de seguridad, eficiencia y trazabilidad.

1.4 Objetivos

El objetivo general de este trabajo es diseñar una arquitectura de gestión documental organizativa basada en IA, validar su viabilidad técnica mediante un prototipo funcional y establecer las pautas para su escalado en entornos de alta criticidad.

Para ello, se busca analizar y sistematizar las principales técnicas de IA aplicadas a la gestión documental (OCR y extracción, búsqueda híbrida y *embeddings*, RAG y su evaluación, y prácticas de gobernanza y seguridad), con el fin de fundamentar las decisiones de diseño de la arquitectura propuesta. A partir de este marco teórico, se desarrolla un prototipo funcional que demuestra empíricamente la viabilidad del enfoque híbrido RAG-SQL en un entorno controlado. Este prototipo, desarrollado en Python con Streamlit y LLMs locales vía Ollama, permite validar los flujos críticos de datos, la orquestación inteligente de consultas y la integración efectiva entre componentes semánticos y deterministas.

Complementariamente, se definen las pautas de escalado desde la arquitectura monolítica del prototipo hacia una solución distribuida basada en microservicios, especificando componentes, tecnologías y requisitos de infraestructura necesarios para un despliegue en producción en entornos de defensa. Finalmente, se establece un marco de evaluación que combina métricas técnicas de recuperación y generación con indicadores clave de desempeño (KPIs) operativos, proporcionando criterios cuantificables para una validación futura en entornos reales.

1.5 Alcance

El alcance de este trabajo se estructura siguiendo una lógica de investigación aplicada que progresa desde la revisión teórica hasta la propuesta arquitectónica, su validación mediante prototipo funcional y las recomendaciones para escalado operativo. Como trabajo de dirección TIC orientado a la toma de decisiones estratégicas en entornos de defensa, este TFM abarca tanto el diseño arquitectónico y la fundamentación técnica como la implementación de una prueba de concepto que demuestre la viabilidad del enfoque propuesto.

El trabajo incluye una revisión exhaustiva del estado del arte en técnicas de IA aplicables a gestión documental, abarcando OCR, extracción de información estructurada, modelos de *embeddings* y representaciones semánticas, técnicas de búsqueda híbrida (léxica y vectorial), arquitecturas de Generación Aumentada por Recuperación (RAG), y técnicas de optimización aplicables a problemas logísticos con restricciones complejas.

Se desarrolla el diseño de una arquitectura de referencia *on-premise* para gestión documental y apoyo a decisiones en entornos de alta criticidad, especificando modelo de datos, estrategias de gobernanza, mecanismos de seguridad y auditoría, y protocolos de control de acceso multinivel. La arquitectura incluye la definición detallada de flujos de ingesta y consulta, estableciendo la separación clara entre componentes generativos (LLMs) y módulos deterministas responsables de validaciones críticas (compatibilidad, cálculos de seguridad, restricciones regulatorias).

Para validar la viabilidad técnica del diseño propuesto, se ha desarrollado un prototipo funcional (PoC). Este prototipo, pretende demostrar empíricamente tres aspectos críticos: la capacidad de un LLM para orquestrar consultas heterogéneas, la efectividad del patrón híbrido de almacenamiento RAG-SQL, y la viabilidad del procesamiento OCR integrado en flujos automatizados de ingesta documental. A partir de los resultados del prototipo, se establece una hoja de ruta de escalado que especifica la transición desde la arquitectura monolítica hacia una solución distribuida basada en microservicios.

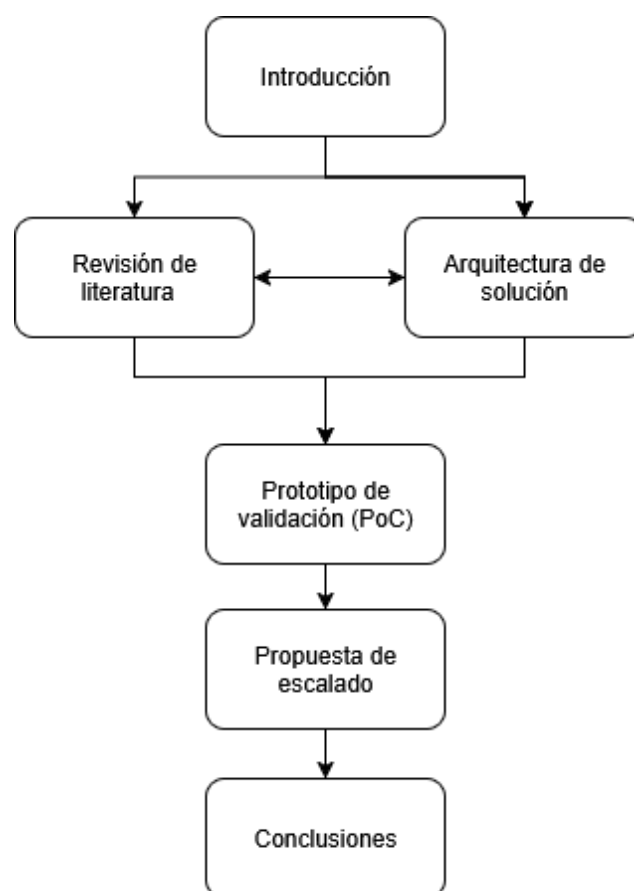


Figura 1-1. Alcance y flujo de trabajo del TFM

El prototipo desarrollado demuestra la viabilidad técnica del enfoque y las pautas de escalado proporcionan una hoja de ruta clara y ejecutable una potencial transición.

Como resultado, el trabajo entrega un diseño arquitectónico validado empíricamente mediante prototipo funcional, acompañado de recomendaciones específicas para una posible implantación en entornos de defensa. Este enfoque proporciona a los responsables de dirección TIC la información estratégica necesaria para la toma de decisiones fundamentadas sobre la adopción de tecnologías de IA en contextos de alta criticidad y seguridad nacional, respaldada por evidencia empírica del funcionamiento real del sistema propuesto.

2 ESTADO DEL ARTE: TÉCNICAS DE IA APLICADAS A LA GESTIÓN DOCUMENTAL

La gestión documental en el ámbito logístico militar exige tratar volúmenes significativos de información heterogénea —fichas técnicas, manuales, registros, normativa y documentación de seguridad— producida en épocas y formatos diversos. La Inteligencia Artificial (IA) aporta el conjunto de técnicas que transforma esa masa documental en conocimiento estructurado y accionable. En este trabajo se abordará el ciclo completo: digitalización y extracción (OCR), análisis semántico (PLN), recuperación y fundamentación de respuestas (RAG) y almacenamiento especializado (SQL y bases vectoriales). Antes de entrar en cada bloque técnico, conviene fijar un marco sintético sobre qué entendemos por IA y cómo ha evolucionado hasta su estado del arte actual.

2.1 Conceptos clave de Inteligencia Artificial

Por IA entenderemos la capacidad de sistemas computacionales para realizar tareas que, si las ejecutara una persona, requerirían inteligencia: razonar, aprender, resolver problemas, tomar decisiones e interactuar con el entorno. La IA es un campo intrínsecamente interdisciplinar que converge computación, matemática, estadística, lingüística, filosofía y psicología, cuyo desarrollo ha alternado aproximaciones simbólicas y conexionistas desde su nacimiento formal en la conferencia de Dartmouth en 1956 [1].

La tradición simbólica, dominante entre las décadas de 1950 y 1980, representó el conocimiento mediante reglas y símbolos formales, originando los sistemas expertos con una base de conocimiento estructurada en reglas "si-entonces" y un motor de inferencia capaz de derivar conclusiones lógicas [2]. Durante esta época se estableció la representación simbólica y el razonamiento basado en reglas como piedras angulares de la IA, influenciando décadas de desarrollo posterior. Los principios que establecieron, desde la representación explícita del conocimiento hasta las estrategias de búsqueda heurística, continúan informando los sistemas de IA modernos incluso cuando el campo ha evolucionado hacia enfoques basados en aprendizaje automático y redes neuronales [3].

La corriente conexionista, inspirada en el procesamiento distribuido del cerebro humano, modeló el aprendizaje como ajuste de parámetros en redes neuronales artificiales. El renacimiento de las redes neuronales en los años ochenta, vino de la mano del redescubrimiento del algoritmo de retropropagación. En 1986, Rumelhart, Hinton y Williams publicaron el artículo seminal "*Learning Representations by Back-Propagating Errors*" en *Nature*, describiendo cómo la retropropagación ajusta repetidamente los pesos de las conexiones en la red para minimizar una medida de la diferencia entre el vector de salida real y el deseado, permitiendo que unidades ocultas internas representen características importantes del dominio de la tarea [4].

Yann LeCun demostró que cuando la retropropagación se aplicaba a redes neuronales convolucionales estas se volvían muy efectivas en reconocer números manuscritos. Durante su tiempo en Toronto como investigador postdoctoral de Hinton, mostró este resultado crucial que llevaría al desarrollo de LeNet-5 en 1998, una arquitectura que anticipaba elementos esenciales de las redes profundas modernas. El artículo "*Gradient-Based Learning Applied to Document Recognition*" [5], diversos métodos aplicados al reconocimiento de caracteres manuscritos y los comparó en una tarea estándar de reconocimiento de dígitos manuscritos, demostrando que las redes neuronales convolucionales superaban todas las otras técnicas.

El punto de inflexión transformador llegó en 2012 cuando Alex Krizhevsky, Ilya Sutskever y Geoffrey Hinton presentaron *AlexNet* en el *ImageNet Large Scale Visual Recognition Challenge* (ILSVRC). *AlexNet* logró una tasa de error top-5 del 15.3%, una mejora dramática sobre el 26.2% del segundo clasificado que asombró a la comunidad de visión por computadora [6]. Las innovaciones técnicas cruciales de *AlexNet* incluyeron el uso de funciones de activación ReLU que aceleraron el entrenamiento seis veces comparado con redes equivalentes usando tanh, técnicas sofisticadas de aumento de datos, y el uso de *dropout* como método de regularización. La arquitectura consistía en ocho capas: cinco convolucionales seguidas de tres completamente conectadas, totalizando aproximadamente 60 millones de parámetros y 650,000 neuronas.

Tras *AlexNet*, la década siguiente vio avances en arquitecturas de redes profundas. *ResNet* introdujo conexiones residuales que permitieron entrenar arquitecturas de cientos de capas superando el problema del gradiente evanescente. En la referencia [7] se presenta un marco de aprendizaje residual para facilitar el entrenamiento de redes sustancialmente más profundas que las usadas anteriormente, evaluando redes residuales con una profundidad de hasta 152 capas—8 veces más profundas que *VGGNet* pero con menor complejidad, logrando 3.57% de error en el conjunto de prueba de *ImageNet*.

En procesamiento de lenguaje natural, la introducción de arquitecturas basadas en transformadores revolucionó la comprensión y generación de texto. El artículo fundamental "*Attention Is All You Need*" de Vaswani propuso el *Transformer*, que se explicará en el apartado 2.3.2.3, una nueva arquitectura basada únicamente en mecanismos de atención, prescindiendo completamente de recurrencia y convoluciones. El modelo logró 28.4 BLEU en la tarea de traducción inglés-alemán de WMT 2014, mejorando los mejores resultados existentes en más de 2 BLEU [8].

BERT (*Bidirectional Encoder Representations from Transformers*), introducido por Devlin [9] representó un avance fundamental al diseñar representaciones bidireccionales profundas preentrenadas a partir de texto sin etiquetar mediante condicionamiento conjunto en contexto izquierdo y derecho en todas las capas. BERT empleó un objetivo de "modelo de lenguaje enmascarado" (MLM) donde el modelo enmascara aleatoriamente algunos tokens de la entrada y el objetivo es predecir el id de vocabulario original basándose únicamente en su contexto.

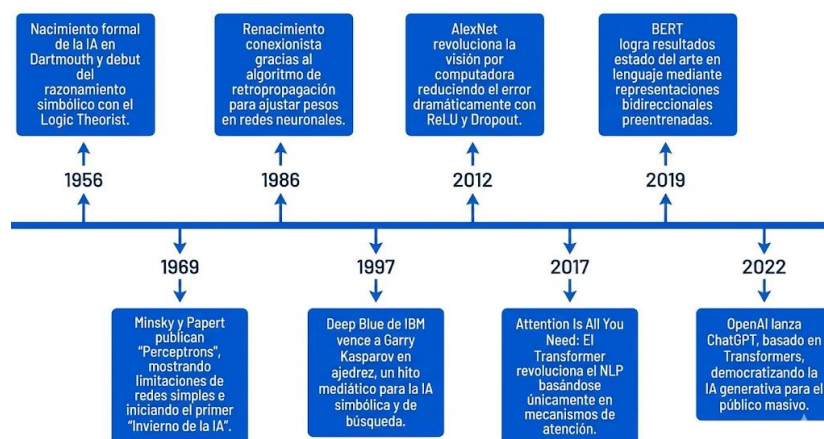


Figura 2-1. Línea del tiempo con principales hitos de la IA (elaboración propia)

2.1.1 Conceptos clave del Machine Learning

Machine Learning (ML) es el subcampo de la IA que busca que los sistemas mejoren su desempeño con la experiencia sin programar explícitamente todas las reglas. Tom Mitchell proporcionó en 1997 una definición formal y precisa: "Se dice que un programa de computadora aprende de la experiencia E con respecto a alguna clase de tareas T y medida de rendimiento P, si su rendimiento en las tareas en T, medido por P, mejora con la experiencia E" [10]. Esta definición establece un marco formal para comprender el aprendizaje automático, enfatizando la mejora en el rendimiento de tareas a través del aprendizaje.

En términos operativos, el aprendizaje automático combina tres elementos fundamentales que definen su proceso iterativo de mejora: un proceso de decisión que mapea entradas a salidas (un clasificador que asigna categorías, un predictor que estima valores futuros, un generador que produce nuevo contenido), una función de error o pérdida que cuantifica la discrepancia entre la salida producida por el modelo y la salida deseada o verdadera, y un proceso de actualización u optimización que ajusta los parámetros internos del modelo para reducir ese error en iteraciones sucesivas [10], [11].

Cuando las representaciones y los modelos son profundos, compuestos por múltiples capas de procesamiento no lineal apiladas, hablamos de *Deep Learning* (DL). Se define el aprendizaje profundo como una forma de aprendizaje automático que permite a las computadoras aprender de la experiencia y entender el mundo en términos de una jerarquía de conceptos [11]. Como la computadora recopila conocimiento de la experiencia, no hay necesidad de que un operador humano especifique formalmente todo el conocimiento que la computadora necesita. La jerarquía de conceptos permite a la computadora aprender conceptos complicados construyéndolos a partir de otros más simples.

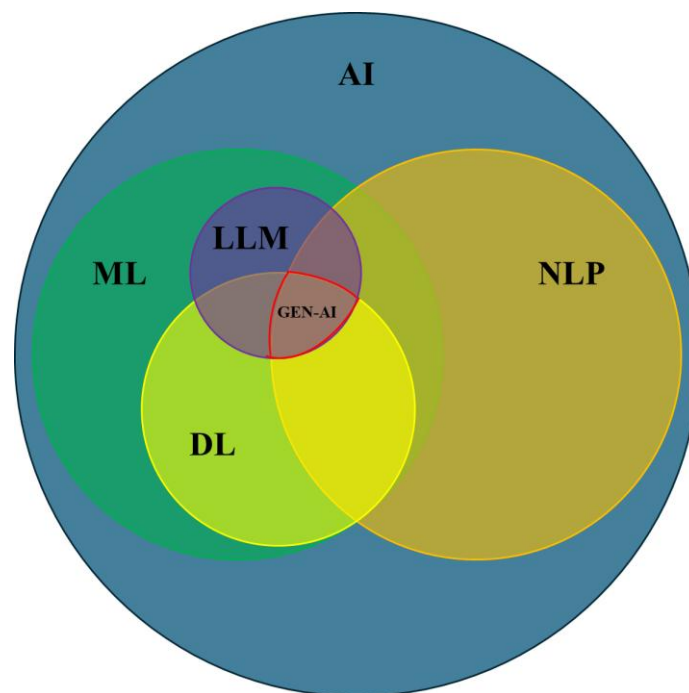


Figura 2-2. Jerarquía y subconjuntos en el campo de la IA.

El término "*deep*" en *deep learning* se refiere al número de capas en una red neuronal artificial (ver Figura 2-3). Cada capa en una red profunda extrae características progresivamente más abstractas y complejas usando transformaciones no lineales sucesivas: las capas iniciales pueden detectar bordes y texturas básicas en imágenes o fonemas en audio, las capas intermedias reconocen formas geométricas, partes de objetos o palabras, y las capas superiores identifican objetos completos, escenas o conceptos semánticos de alto nivel [11]. Esta jerarquía de representaciones aprendidas automáticamente es lo que

distingue el aprendizaje profundo del aprendizaje automático tradicional, donde las características relevantes debían ser diseñadas manualmente por expertos del dominio.

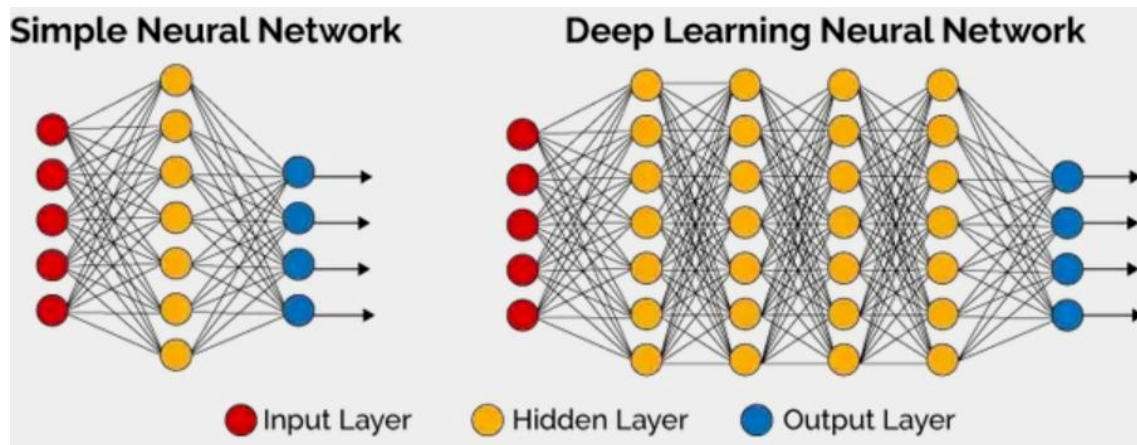


Figura 2-3. Comparación de arquitectura: Red Neuronal Simple vs. Red de Aprendizaje Profundo [12]

Entre las variantes principales de arquitecturas profundas destacan las Redes Neuronales Convolucionales (CNN) y las Redes Neuronales Recurrentes (RNN), cada una optimizada para tipos específicos de datos y tareas. Las CNN, fundamentadas en el trabajo de [5], están diseñadas específicamente para tareas perceptuales como procesamiento de imágenes, analizando patrones de píxeles mediante filtros convolucionales que se deslizan sobre la imagen identificando características espaciales locales. Las RNN, por otro lado, incorporan conexiones recurrentes que permiten que los datos fluyan tanto hacia adelante como hacia atrás en el tiempo, manteniendo un estado oculto interno que actúa como memoria de corto plazo de la secuencia procesada [11].

2.1.2 Paradigmas de aprendizaje

En función de la disponibilidad y naturaleza de los datos distinguimos cuatro paradigmas, relevantes en distinto grado para este trabajo [10], [11]:

- Supervisado: aprende a partir de pares entrada-salida etiquetados (p. ej., clasificar tipos documentales o detectar campos extraídos correctamente). Es el enfoque preferente cuando disponemos de verdad terreno fiable.
- No supervisado: descubre estructura latente sin etiquetas (agrupación de documentos por tema, detección de duplicados o versiones). Útil para curación documental y control de calidad del corpus.
- Semi-supervisado: combina un subconjunto pequeño de datos etiquetados con un volumen grande sin etiquetar. Reduce el coste de anotación en dominios especializados como el militar.
- Aprendizaje por refuerzo: optimiza decisiones por prueba-error en interacción con un entorno. En el contexto de este TFM aparece indirectamente en técnicas de alineación de LLM (p. ej., RLHF) y en la orquestación multiagente cuando se exploran planes y se evalúan recompensas (seguridad, precisión, tiempo).

2.1.3 Aplicación de IA en entornos profesionales

La inteligencia artificial ha demostrado su valor práctico en numerosos sectores. En el comercio minorista, por ejemplo, los algoritmos de predicción de demanda ayudan a mantener niveles óptimos de inventario, mientras que los sistemas de recomendación personalizan la experiencia de compra de cada cliente. El sector financiero aprovecha modelos predictivos para detectar fraudes en tiempo real, analizando millones de transacciones y detectando patrones sospechosos mucho más rápido que cualquier auditor humano. También se utiliza para evaluar la solvencia de solicitantes de crédito, procesando múltiples fuentes de datos de forma más consistente que los métodos tradicionales.

En logística, la IA puede optimizar rutas de transporte, gestiona la asignación de recursos como vehículos y almacenes. También es posible utilizar visión artificial para tareas como recuentos de inventario o detección de daños. Tecnologías como RFID e IoT permiten seguimiento en tiempo real, mientras que los gemelos digitales facilitan la simulación de escenarios complejos. Además, los sistemas automáticos son capaces de verificar el cumplimiento normativo revisando documentación técnica y etiquetas, detectar anomalías en inventarios y predecir necesidades de mantenimiento de equipos.

El ámbito militar también ha adoptado estas tecnologías. Como señala la referencia [2], la IA se aplica en reconocimiento de objetivos mediante visión artificial, procesamiento de inteligencia de múltiples fuentes y optimización de cadenas de suministro bajo condiciones adversas. El ejército estadounidense emplea sistemas de IA para predecir tareas de mantenimiento analizando datos de sensores, operar vehículos autónomos en zonas peligrosas y gestionar grandes volúmenes de datos logísticos.

La logística militar se beneficia particularmente de sistemas de IA para previsión de demanda de suministros, seguimiento en tiempo real de activos y personal, y ciberdefensa. El ejército estadounidense ha adoptado sistemas de IA en múltiples áreas logísticas, incluyendo predicción de tareas de mantenimiento mediante análisis de datos de sensores de equipamiento, operaciones de reabastecimiento de combustible sin conductor en entornos peligrosos mediante vehículos autónomos, y almacenamiento masivo de datos logísticos con capacidades de IBM Watson para procesamiento y análisis.

Sin embargo, la integración de IA en contextos militares presenta desafíos técnicos, éticos y estratégicos significativos que requieren atención cuidadosa. Los riesgos técnicos incluyen potencial de mal funcionamiento en condiciones no previstas, vulnerabilidad a hackeo y otros ciberataques, y dependencia de infraestructura tecnológica. Las implicaciones éticas y legales son especialmente críticas en relación con armas autónomas letales, planteando preguntas sobre delegación de autoridad para tomar vidas humanas, *accountability* cuando sistemas autónomos cometen errores, y cumplimiento con derecho internacional humanitario.

2.1.4 Aplicación de técnicas de IA a la gestión documental organizativa

La gestión documental organizativa abarca el ciclo de vida completo de la información con requisitos estrictos de integridad, trazabilidad, seguridad y cumplimiento normativo [2]. La IA permite transformar repositorios estáticos de archivos pasivos en sistemas cognitivos activos capaces de convertir documentos heterogéneos en datos fiables, estructurados según esquemas formales y reutilizables que alimentan analítica avanzada.

2.1.4.1 Captura y normalización

En la fase de captura y normalización, la ingesta inteligente combina OCR profundo basado en redes neuronales convolucionales avanzadas que superan ampliamente precisión de métodos tradicionales, fundamentadas en los avances de LeCun [5] en reconocimiento de documentos mediante aprendizaje basado en gradientes, junto con análisis de estructura de página que identifica regiones funcionales para documentos escaneados preservando estructura lógica.

La limpieza aborda ruido, rotaciones, tablas complejas y formularios, generando salidas estructuradas (JSON/Parquet) con metadatos normalizados (autoría, fecha, versión, clasificación, sensibilidad). Para asegurar la calidad de los datos, utilizamos procesos automatizados que revisan que el formato sea correcto, las unidades coincidan y que los valores pertenezcan a listas permitidas.

2.1.4.2 Clasificación y enrutamiento

Modelos de clasificación temática y de tipología documental (p. ej., ficha técnica, informe de inspección, normativa) permiten enrutar automáticamente a flujos específicos (según el tipo de documento). Los modelos utilizan un entrenamiento estándar, pero se pueden complementar con

Modelos de Lenguaje capaces de aprender a partir de unas pocas muestras cuando la información es escasa o muy variada.

2.1.4.3 Extracción semántica

La extracción semántica mediante técnicas de NER (*Named Entity Recognition*) identifica entidades clave y sus interacciones: material con código y descripción, lote con identificador único, proveedor con nombre y certificaciones, fechas críticas, ubicaciones físicas. Se aplican reglas de negocio para consolidar mediante resolución de entidades, desambiguación y deduplicación antes de persistir en sistemas maestros relacionales SQL y en representaciones de conocimiento como grafos que modelan ontologías de dominio.

2.1.4.4 Búsqueda y recuperación avanzada

La búsqueda y recuperación avanzada se fundamenta en indexación híbrida que combina métodos léxicos tradicionales con búsqueda vectorial semántica basada en *embeddings* densos generados por modelos de lenguaje. Esta arquitectura híbrida habilita recuperación contextual sofisticada para sistemas RAG (*Retrieval-Augmented Generation*).

Lewis et al. [13] introdujeron RAG como una receta de ajuste fino de propósito general que combina memoria paramétrica preentrenada (el LLM) con memoria no paramétrica (índice vectorial denso de corpus documental). Los modelos de lenguaje grandes preentrenados han demostrado almacenar conocimiento factual en sus parámetros, sin embargo, su habilidad de acceder y manipular precisamente conocimiento es aún limitada. RAG explora una aproximación donde la memoria paramétrica es un modelo seq2seq preentrenado y la memoria no paramétrica es un índice vectorial denso accedido mediante un *retriever* neuronal preentrenado.

Los fragmentos recuperados incluyen procedencia, versiones y niveles de clasificación, lo que permite respuestas generativas ancladas en evidencia con citas y enlaces a la fuente, mejorando precisión y auditabilidad.

2.1.4.5 Automatización documental

La automatización documental aprovecha capacidades de IA generativa usando LLM para producir múltiples tipos de contenido derivado útil: resúmenes ejecutivos, comparativas entre versiones, extractos temáticos, borradores conformes a plantillas. Estas generaciones automáticas incorporan verificaciones de coherencia numérica, consistencia de referencias cruzadas, y cumplimiento de normas de estilo corporativo, registrando meticulosamente todas las interacciones humanas con huellas digitales.

2.1.4.6 Gobernanza, seguridad y cumplimiento

El pilar de gobernanza, seguridad y cumplimiento implementa controles técnicos y organizativos robustos. Control de acceso basado en roles (RBAC) o basado en atributos (ABAC), clasificación y etiquetado de sensibilidad, cifrado en reposo con AES-256 y en tránsito mediante TLS, linaje de datos que registra exhaustivamente accesos y modificaciones, versionado estricto que mantiene histórico completo de evolución documental preservando todas las revisiones.

2.1.4.7 Indicadores y mejora continua

La operación del sistema se monitoriza utilizando indicadores clave de rendimiento cuantitativos: cobertura de clasificación, precisión y *recall* de extracción de entidades, tasa de *grounding* correcto en RAG, latencia por etapa del pipeline. Estos indicadores cuantitativos alimentan ciclos estructurados de MLOps que sistematizan mejora continua con reentrenos periódicos, ajuste fino eficiente de parámetros (PEFT), actualización incremental de vocabularios controlados, y mejora continua basada en retroalimentación de usuarios.

2.1.4.8 Arquitectura de referencia

Una arquitectura de referencia robusta combina varios componentes especializados: un *data lake* documental que almacena documentos en su forma bruta original y formas derivadas procesadas, bases de datos SQL relacionales para datos maestros de entidades críticas y transaccionales, *vector stores* especializadas optimizadas para recuperación semántica eficiente mediante búsqueda por similitud de *embeddings*, bases de datos de grafos para modelar relaciones complejas entre entidades, y una capa de orquestación superior que implementa arquitecturas RAG coordinando recuperador con generador y sistemas multiagente [13].

El valor estratégico de estos sistemas radica en transformar documentación dispersa, estática y de difícil acceso en un activo dinámico de conocimiento organizacional que se actualiza continuamente, unificado proporcionando punto de acceso único, y accionable sustentando decisiones operativas con evidencia trazable directamente a fuentes autorizadas.

Este marco conceptual justifica las elecciones tecnológicas del resto del capítulo: usar OCR profundo para digitalizar con robustez; aplicar PLN con representaciones contextuales y LLM; fundamentar respuestas mediante RAG con bases vectoriales; y mantener datos maestros y trazabilidad en SQL. En las secciones siguientes (2.2–2.6) desarrollamos estas piezas con el nivel técnico y el énfasis operativo que requiere la gestión logística de municiones.

2.2 Extracción y preprocesamiento de información

La extracción y el preprocesamiento de información representan etapas fundamentales en los sistemas de IA orientados a la gestión documental, particularmente en entornos logísticos complejos. Estos procesos facilitan la conversión de datos no estructurados o semiestructurados —provenientes de documentos físicos, digitales o multimodales— en representaciones estructuradas y procesables, aptas para su integración en algoritmos de aprendizaje automático y procesamiento del lenguaje natural. En el ámbito logístico, donde la documentación exhibe una diversidad de formatos, idiomas y niveles de degradación, la robustez de estas técnicas incide directamente en la precisión, eficiencia y escalabilidad del sistema global [11].

2.2.1 Técnicas de OCR

El OCR representa un pilar fundamental en la transformación digital de documentos, al convertir texto incrustado en imágenes o escaneos en formato editable y procesable por sistemas computacionales.

Históricamente, las técnicas de OCR se han sustentado en enfoques tradicionales basados en reglas heurísticas y algoritmos clásicos de procesamiento de imágenes, que siguen un flujo secuencial diseñado para entornos controlados. Este pipeline inicia con el preprocesamiento de la imagen, donde se aplican filtros para mejorar la calidad, tales como la binarización para convertir la imagen en blanco y negro, la eliminación de ruido mediante técnicas de suavizado, la corrección de inclinaciones a través de algoritmos de detección de líneas, y la normalización del contraste para resaltar el texto. Posteriormente, la segmentación identifica regiones de interés, separando líneas, palabras y caracteres individuales mediante métodos como el análisis de componentes conectados o proyecciones histogramáticas horizontales y verticales. La extracción de características calcula descriptores cuantitativos, incluyendo momentos invariantes para formas geométricas, perfiles de proyección para contornos, y patrones de píxeles para texturas. La clasificación asigna cada segmento a una categoría utilizando comparaciones con plantillas o clasificadores supervisados como k-vecinos más cercanos (k-NN) o máquinas de vectores de soporte (SVM). Finalmente, el post-procesamiento corrige errores mediante diccionarios léxicos y modelos probabilísticos de lenguaje que evalúan la coherencia contextual del texto extraído. Estos métodos destacan por su eficiencia computacional y su rendimiento óptimo en documentos con texto impreso claro, fondos uniformes y fuentes estandarizadas, pero revelan limitaciones significativas en escenarios reales, donde factores como iluminación variable, degradación por exposición ambiental

o ruido introducido por escaneos antiguos comprometen su precisión y aplicabilidad, especialmente en contextos militares con documentación expuesta a condiciones adversas en buques o almacenes.

Frente a estas restricciones, las técnicas de OCR impulsadas por aprendizaje profundo permiten extraer representaciones jerárquicas directamente de datos crudos sin requerir una ingeniería manual exhaustiva de características. En el núcleo de estos avances se encuentran las redes neuronales convolucionales (CNN), que actúan como espina dorsal para la extracción de rasgos visuales. Arquitecturas como ResNet, introducidas por He et al. (2016), potencian esta funcionalidad mediante conexiones residuales, incrementando la resiliencia ante variaciones estilísticas y degradaciones comunes en documentos logísticos. El artículo seminal de He et al. presentó un marco de aprendizaje residual para facilitar el entrenamiento de redes sustancialmente más profundas que las usadas anteriormente, evaluando redes residuales con una profundidad de hasta 152 capas y logrando resultados superiores en *benchmarks* de reconocimiento visual.

Las redes neuronales recurrentes (RNN), junto con sus variantes como LSTM y GRU, se integran en modelos híbridos como CRNN (*Convolutional Recurrent Neural Network*), que capturan dependencias secuenciales en el texto. Esta arquitectura novedosa integra extracción de características, modelado de secuencias y transcripción en un marco unificado, poseyendo propiedades distintivas: es entrenable de extremo a extremo en contraste con la mayoría de algoritmos existentes cuyos componentes se entrenan y ajustan por separado, maneja naturalmente secuencias de longitudes arbitrarias sin involucrar segmentación de caracteres o normalización de escala horizontal, no está confinado a ningún léxico predefinido y logra rendimientos notables tanto en tareas de reconocimiento de texto con léxico como sin él [14].

Los mecanismos de atención y los transformadores han sido incorporados en arquitecturas especializadas en OCR. Vaswani et al. (2017) propusieron el *Transformer* en su artículo fundamental "*Attention Is All You Need*" [8], una nueva arquitectura basada únicamente en mecanismos de atención, prescindiendo completamente de recurrencia y convoluciones. El modelo logró 28.4 BLEU en la tarea de traducción inglés-alemán de WMT 2014, mejorando los mejores resultados existentes en más de 2 BLEU. Esta arquitectura ha sido adaptada para tareas de OCR en arquitecturas como TrOCR, que dirigen el foco hacia regiones relevantes de la imagen, habilitando el procesamiento de disposiciones no lineales o multimodales, y permitiendo un *fine-tuning* eficiente a partir de conjuntos de datos preentrenados masivos para adaptarse a dominios específicos como la logística naval.

En 2025, el estado del arte en OCR integra modelos de lenguaje visual (VLM) y grandes modelos de lenguaje multimodales. Los últimos modelos generativos mejoran los flujos de OCR tradicionales al procesar conjuntamente imágenes y texto, mejorando la precisión en la extracción de datos complejos mediante una comprensión semántica contextual. Herramientas de OCR contemporáneas han evolucionado incorporando técnicas de aprendizaje por transferencia, permitiendo adaptaciones rápidas a documentos heterogéneos sin requerir grandes volúmenes de datos anotados.

No obstante, estos progresos no están exentos de desafíos. Los modelos de OCR profundos exigen conjuntos de datos masivos para el entrenamiento y recursos computacionales intensivos, particularmente problemáticos en fases de inferencia sobre el terreno. Además, presentan una opacidad interpretativa que complica la validación de resultados en contextos regulados como la gestión de municiones, donde la trazabilidad y explicabilidad de cada decisión son imperativas. En la práctica, los enfoques híbridos surgen como una solución óptima, equilibrando precisión, eficiencia y escalabilidad.

2.2.2 Extracción de información estructurada

Una vez que el texto ha sido extraído mediante técnicas de OCR, surge la necesidad de identificar y organizar la información relevante contenida en los documentos, transformando el texto libre en datos estructurados que puedan ser almacenados en bases de datos, sometidos a análisis estadísticos o empleados como insumos para sistemas de soporte a la decisión. En el contexto de la gestión logística militar, esta extracción estructurada adquiere una dimensión crítica, ya que permite automatizar la

captura de elementos clave como identificadores de material, fechas de caducidad o ubicaciones de almacenamiento, minimizando errores humanos y acelerando procesos operativos en entornos de alta criticidad. Este proceso no solo eleva la calidad de los datos disponibles para modelos de lenguaje grandes (LLM) y arquitecturas de Recuperación Aumentada por Generación (RAG), sino que también asegura que las consultas generativas produzcan resultados precisos y accionables, como predicciones sobre la vida útil de equipamientos o evaluaciones de conformidad normativa.

2.2.2.1 Técnicas de *Named Entity Recognition* (NER)

El Reconocimiento de Entidades Nombradas (NER, por sus siglas en inglés) es una tarea esencial dentro del PLN, enfocada en la identificación y clasificación de elementos textuales en categorías predefinidas tales como personas, organizaciones, ubicaciones, fechas, cantidades o códigos. En el ámbito logístico militar, el NER facilita la extracción automática de información vital de los documentos. Por ejemplo, a partir de informes y fichas se pueden extraer identificadores de material (NIIN, números de lote, calibres, vida útil, equivalentes NATO, grupos de riesgo), cantidades y unidades de medida, fechas asociadas a caducidad, fabricación o inspecciones programadas, ubicaciones de almacenamiento o destino, nombres de proveedores o fabricantes, así como estados operativos o clasificaciones de seguridad. Estas capacidades resultan especialmente valiosas en escenarios donde documentación heterogénea —proveniente de manuales históricos en papel o informes operativos digitales— debe procesarse rápidamente para respaldar decisiones en tiempo real. La información estructurada extraída mediante NER puede luego integrarse en sistemas LLM para generar informes sintetizados, o en arquitecturas RAG para recuperar contextos normativos específicos bajo consulta.

Las técnicas clásicas de NER se basan en enfoques lingüísticos y de aprendizaje automático supervisado, ofreciendo una base sólida, aunque limitada en generalización. Los sistemas basados en reglas emplean expresiones regulares, *gazetteers* —listas de entidades conocidas— y patrones gramaticales para detectar entidades; por instancia, patrones específicos pueden reconocer códigos alfanuméricos de municiones o formatos estandarizados de fechas, proporcionando resultados deterministas e interpretables sin necesidad de datos de entrenamiento extensos, si bien requieren mantenimiento manual constante para adaptarse a variaciones de dominio. Por otro lado, los modelos probabilísticos secuenciales, como los *Conditional Random Fields* (CRF) y *Hidden Markov Models* (HMM), modelan dependencias entre etiquetas de palabras adyacentes, incorporando características contextuales tales como palabras vecinas, prefijos, sufijos y capitalización para asignar categorías con mayor precisión probabilística. Los CRF, en particular, han demostrado utilidad en entornos logísticos al permitir la integración de características arbitrarias sin asumir independencia entre observaciones, aunque su rendimiento se ve afectado en datasets con ruido inherente a documentos degradados [15].

El avance hacia técnicas modernas de NER impulsadas por aprendizaje profundo ha revolucionado el campo. En 2019 se introdujo BERT (*Bidirectional Encoder Representations from Transformers*) [9]. BERT representó un avance fundamental al diseñar representaciones bidireccionales profundas preentrenadas a partir de texto sin etiquetar mediante condicionamiento conjunto en contexto izquierdo y derecho en todas las capas. BERT empleó un objetivo de "modelo de lenguaje enmascarado" (MLM) donde el modelo enmascara aleatoriamente algunos tokens de la entrada y el objetivo es predecir el id de vocabulario original basándose únicamente en su contexto. El modelo BERT preentrenado puede ajustarse finamente con solo una capa de salida adicional para crear modelos estado del arte para un amplio rango de tareas, obteniendo nuevos resultados en once tareas de procesamiento de lenguaje natural.

Arquitecturas como BiLSTM-CRF combinan redes LSTM bidireccionales —que capturan contexto tanto anterior como posterior en la secuencia— con una capa CRF para modelar transiciones entre etiquetas, permitiendo aprender representaciones contextuales profundas sin ingeniería de características manual. En el estado del arte de 2025, la integración de grandes modelos de lenguaje (LLM) ha impulsado innovaciones significativas en NER. Li et al. [16] propusieron un marco unificado basado en

comprensión de lectura automática (MRC) para NER que reformula la tarea como pregunta-respuesta, logrando mejoras significativas en datasets de referencia. En la referencia [17] se introduce GLiNER (*Generalist Model for NER using Bidirectional Transformer*), un paradigma novedoso que integra información contextual global en NER y puede identificar tipos de entidades arbitrarios en tiempo de inferencia sin reentrenamiento.

Herramientas comerciales, como *Google Cloud Natural Language API* o *IBM Watson NLU*, ofrecen NER con soporte multilingüe y personalización, mejorando la precisión en dominios logísticos mediante entrenamiento en datasets especializados. No obstante, el principal desafío en aplicaciones logísticas radica en la especificidad del dominio: modelos generales requieren adaptación para reconocer entidades únicas, lo que puede implicar la creación de corpus anotados o aprendizaje por transferencia desde dominios afines, asegurando así que los datos estructurados alimenten eficazmente LLM y RAG para consultas precisas.

2.2.2.2 Extracción de tablas y formularios

Una porción significativa de la información logística se presenta en tablas y formularios estandarizados, lo que plantea desafíos únicos al requerir una comprensión integrada de la estructura espacial y el contenido semántico del documento.

El proceso inicia con la detección y segmentación de tablas, donde enfoques clásicos buscan líneas horizontales y verticales, patrones de celdas y otros indicios para delimitar bordes de tabla y separaciones de celdas. Métodos tradicionales aplican detección de bordes, técnicas de agrupamiento (*clustering*) y análisis de la distribución de caracteres para identificar la presencia de una tabla incluso si carece de bordes explícitos.

Por otra parte, métodos modernos utilizan redes neuronales convolucionales entrenadas para detectar estructuras tabulares de manera *end-to-end*, superando las limitaciones de los heurísticos en documentos con diseños complejos. En 2025, servicios especializados como Amazon Textract o Google Cloud Document AI han avanzado en el manejo de formatos complicados, extrayendo tablas de PDFs e imágenes con alta precisión con modelos de ML que se adaptan a diferentes *layouts*.

Una vez localizada la tabla, continúa la comprensión de su estructura interna, identificando filas, columnas, celdas combinadas y encabezados. Pueden emplearse heurísticas que analizan la distribución espacial de los textos, o bien modelos de aprendizaje profundo especializados. Por ejemplo, *Graph Neural Networks* (GNN) han sido aplicadas para modelar las relaciones entre celdas adyacentes, infiriendo la estructura lógica de la tabla a partir de la topología de sus elementos. Asimismo, enfoques basados en mecanismos de atención alinean elementos visuales con posiciones tabulares esperadas, mejorando la robustez frente a variaciones en documentos logísticos con formatos poco convencionales. Entre los avances recientes destaca el modelo *Arctic-TILT* de *Snowflake Document AI*, especializado en extracción de información tabular compleja, que permite procesar informes financieros o *datasheets* con tablas incrustadas de manera altamente precisa.

La extracción del contenido de las tablas típicamente implica aplicar OCR a nivel de celda para obtener el texto de cada campo y luego asociar esos valores con sus respectivos encabezados o etiquetas. El resultado se representa en estructuras de datos, como archivos JSON o tablas relacionales, optimizadas para posteriores búsquedas o consultas RAG sobre datos logísticos. Para los formularios, el proceso puede beneficiarse de conocimiento previo sobre sus plantillas: si se dispone del formato esperado de ciertos formularios (por ejemplo, un parte de inspección de municiones), se pueden usar plantillas predefinidas para mapear cada campo de entrada a una etiqueta estructurada, reduciendo ambigüedades. Los modelos modernos de comprensión visual de documentos, como los de Microsoft *Azure Form Recognizer*, integran información de texto, componentes visuales y posicionamiento relativo para extraer automáticamente campos variables sin depender totalmente de plantillas fijas. En entornos militares, la extracción confiable de tablas y formularios asegura la trazabilidad y precisión de

los datos, al fusionar información de múltiples fuentes (p. ej., combinar inventarios de diferentes sistemas) para respaldar decisiones críticas de forma consistente.

2.2.2.3 Limpieza y normalización de datos

La información extraída con OCR y técnicas de extracción estructurada inevitablemente incorpora errores e inconsistencias que deben rectificarse antes de su empleo en etapas subsiguientes, garantizando la integridad de los datos en sistemas logísticos militares donde precisiones mínimas pueden impactar operaciones. La corrección de errores OCR aborda imperfecciones en documentos degradados o manuscritos mediante diccionarios y modelos de lenguaje que comparan palabras con términos válidos, utilizando probabilísticos contextuales para correcciones; modelos de DL secuencia-a-secuencia, como transformadores [8], mapean texto erróneo a correcto, mientras que post-procesamiento dominio-específico valida contra listas controladas de códigos y nomenclaturas en logística militar.

La normalización de formatos convierte datos a estándares uniformes: fechas de múltiples representaciones a YYYY-MM-DD, cantidades con conversión de unidades y manejo de convenciones numéricas, o códigos con eliminación de inconsistencias, empleando técnicas como escalado para variables numéricas.

La detección de valores ausentes o inválidos implica marcar campos no extraídos para revisión, con validación que identifica rangos anómalos o inconsistencias; resolución de ambigüedades usa desambiguación contextual, puntuaciones de confianza y validación cruzada entre fuentes. En contextos logísticos, estas prácticas preparan datos para LLM y RAG, minimizando sesgos y elevando la fiabilidad de respuestas generadas.

2.2.3 Preprocesamiento de datos logísticos

Los sistemas logísticos contemporáneos deben integrar información proveniente de fuentes heterogéneas, cada una con sus propios formatos, niveles de estructuración y requisitos de procesamiento. El preprocesamiento adecuado de estos datos resulta crucial para garantizar la calidad y utilidad de la información en las fases posteriores del pipeline de IA, como la generación de *embeddings* para bases de datos vectoriales en RAG o la alimentación de modelos de lenguaje para análisis avanzados.

En esta etapa se incluyen tareas como la integración de datos (unificación de distintas bases de datos o fuentes en una estructura común), la transformación de atributos (creación de nuevas variables derivadas, codificación de categorías, anonimización de información sensible) y la reducción de dimensionalidad si fuera necesaria para eliminar redundancias. En el dominio logístico militar, a menudo se debe combinar información procedente de sistemas *legacy* (por ejemplo, un sistema antiguo de inventario) con datos actuales de sensores o plataformas modernas, armonizando campos y resolviendo discrepancias. El preprocesamiento también engloba la normalización semántica, que consiste en homogeneizar términos y unidades: por ejemplo, garantizar que todos los documentos usen la misma nomenclatura para un calibre o el mismo formato para expresar la cantidad neta explosiva (NEQ) de una munición.

Otro aspecto importante es la indexación y fragmentación de documentos extensos. Para facilitar su posterior recuperación, documentos muy largos (como manuales completos) pueden dividirse en secciones o párrafos indexados, manteniendo referencias cruzadas para reconstruir contexto cuando se usen en un sistema RAG. Asimismo, se generan metadatos enriquecidos que describen cada documento (fecha, fuente, clasificación de seguridad, etc.), lo cual es vital para filtrar resultados en consultas posteriores o aplicar controles de acceso.

En resumen, el preprocesamiento de datos logísticos garantiza que, tras la extracción inicial, la información esté lista para usar en los sistemas de IA. Las técnicas descritas de limpieza, normalización e integración de datos conforman un puente entre la digitalización de documentos y la aplicación de algoritmos avanzados. A medida que estas etapas preparatorias se realizan con rigor, las fases siguientes

de análisis y generación de conocimiento (mediante PLN, LLM, RAG, etc.) pueden operar sobre cimientos de datos sólidos, confiables y coherentes.

2.3 Procesamiento de lenguaje natural

Una vez digitalizados y estructurados los documentos, el siguiente paso consiste en aplicar técnicas de procesamiento del lenguaje natural (PLN) para comprender y explotar el contenido textual. El propósito central del PLN es permitir que una máquina comprenda e interactúe con el texto escrito por humanos. Detrás de esta meta general subyace una enorme complejidad: el lenguaje natural es rico en matices, ambigüedades y variaciones de estilo, lo que hace que su procesamiento automático sea desafiante. En el contexto de la logística de municiones, las técnicas de PLN permiten analizar reportes técnicos, normas de seguridad o comunicaciones operativas, extrayendo de ellos información valiosa y facilitando respuestas automatizadas o resúmenes que apoyen la toma de decisiones.

A lo largo de las últimas décadas, el PLN ha evolucionado desde métodos basados en reglas gramaticales y modelos estadísticos sencillos hasta enfoques de aprendizaje profundo con modelos neuronales de gran escala [11]. A continuación, se describen las principales técnicas de representación y comprensión del lenguaje utilizadas en la actualidad, con énfasis en su aplicación en sistemas de documentación logística.

2.3.1 Representación de texto

Un primer paso para que las máquinas procesen lenguaje es transformar las palabras en una representación numérica manejable (*embeddings*). Tradicionalmente, se emplearon modelos simplificados como la bolsa de palabras (*Bag of Words*), donde cada documento se representa por las palabras que contiene, ignorando el orden, pero contabilizando las frecuencias. Este enfoque conduce a vectores muy dispersos (de dimensionalidad igual al vocabulario total) e ignora relaciones semánticas importantes (por ejemplo, no distingue entre sinónimos ni capta el contexto). Mejorando esta idea básica, el modelo de Bolsa de Palabras Ponderada mediante TF-IDF introduce un peso para cada término según su frecuencia local en el documento y su rareza global en el corpus, atenuando la influencia de palabras demasiado comunes y destacando términos distintivos.

A pesar de ser simples y efectivos para ciertas tareas, estos esquemas tradicionales presentan limitaciones significativas. La representación de un documento como un vector de presencia/ausencia o frecuencia de términos no captura el significado ni las relaciones sintácticas: “munición caducada” y “caducidad de munición” tendrían la misma representación, perdiendo información de orden y contexto. Para superar estas deficiencias surgieron los modelos de espacio vectorial semántico, que buscan asignar a palabras y documentos vectores en un espacio continuo donde la cercanía refleje similitud de significado. Técnicas clásicas como el *Latent Semantic Analysis* (LSA) aplicaban factorizaciones matriciales a la matriz documento-término para descubrir dimensiones latentes que resumieran conceptos subyacentes, mientras que métodos probabilísticos como *Latent Dirichlet Allocation* (LDA) modelaron los documentos como mezclas de temas latentes.

2.3.1.1 Word embeddings y representaciones distribuidas

La siguiente de incrustaciones de texto se desarrolla utilizando la capacidad de generalización del aprendizaje profundo. Estos se fundamentan en la hipótesis distribucional, que postula que términos con significados parecidos tienden a aparecer en entornos textuales similares. Aunque el Análisis Semántico Latente (LSA, por sus siglas en inglés) fue pionero en aplicar esta idea al PLN, no logró capturar adecuadamente las conexiones semánticas entre vocablos. Gracias al progreso en la capacidad computacional y al refinamiento teórico de esta hipótesis, han surgido modelos de ML capaces de mapear palabras a vectores en espacios multidimensionales, asignando representaciones similares a términos con sentidos análogos.

Los primeros modelos, basados en redes neuronales poco profundas, facilitaron representaciones densas de palabras. Uno de los métodos más renombrados y eficaces es Word2Vec. Desarrollado en 2013, Word2Vec [18] utiliza un modelo de red neuronal que busca capturar los significados y contextos de las palabras en un espacio vectorial de alta dimensión. Su premisa central es que cada palabra está influida por sus vecinas; esto define el contexto C , compuesto por las k palabras adyacentes a la palabra objetivo w . Word2Vec ofrece dos variantes arquitectónicas principales:

- CBOW (Bolsa de Palabras Continua): el modelo predice la palabra actual a partir de una ventana de contexto circundante C de k palabras. El orden de las palabras del contexto no influye en la predicción (suposición de bolsa de palabras), y la eventual presencia de repetición de palabras se ignora.
- Skip-Gram: el modelo usa la palabra actual para predecir la ventana de contexto circundante C de k palabras. La arquitectura *skip-gram* pondera más fuertemente las palabras de contexto cercanas que las distantes.

El modelo consta de una sola capa oculta que produce la representación vectorial de las palabras; estos vectores se inicializan con valores aleatorios y se actualizan gradualmente durante el entrenamiento. Es esencial señalar que el procedimiento de entrenamiento no requiere anotaciones. Ambas estrategias de entrenamiento están representadas en la Figura 2-4.

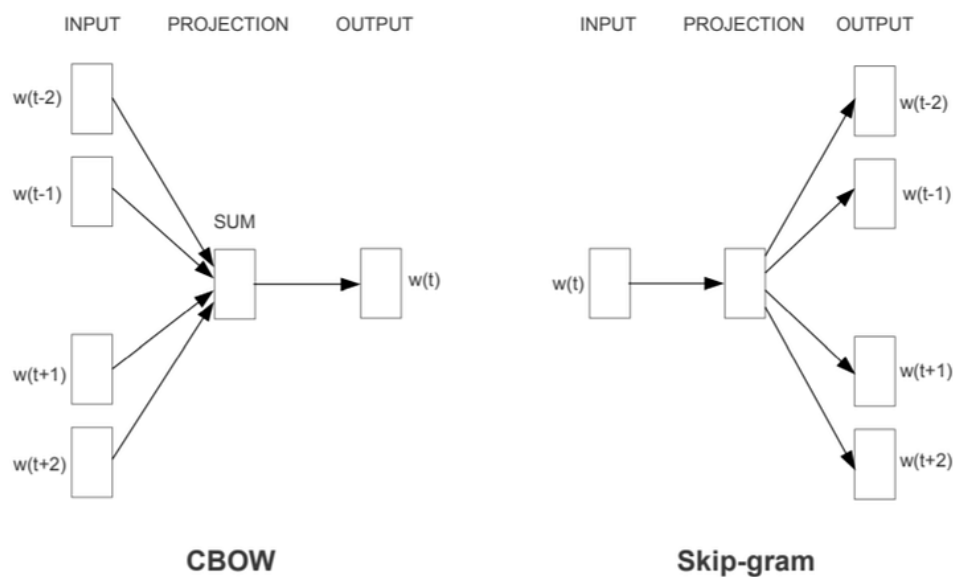


Figura 2-4: Estrategias de entrenamiento para Word2Vec [19].

Estas incrustaciones de palabras contienen información semántica importante que permite la detección de relaciones entre palabras. Para entender cuán similares son dos palabras (ω_i, ω_j) , se puede medir la similitud entre sus representaciones vectoriales $(\vec{\omega}_i, \vec{\omega}_j)$. Diferentes medidas, como la similitud euclidiana, de Manhattan o coseno, pueden usarse para realizar esta tarea. Cada una tiene sus ventajas y desventajas en relación con la búsqueda semántica. La similitud coseno de dos vectores calcula el coseno del ángulo entre ellos:

$$\cos(\omega_i, \omega_j) = \frac{\vec{\omega}_i \cdot \vec{\omega}_j}{|\vec{\omega}_i| |\vec{\omega}_j|}$$

Este método es ampliamente utilizado porque permanece constante independientemente de la longitud de los dos vectores, es computacionalmente eficiente para vectores dispersos, y el rango de

valores que toma va de -1 a 1, lo que facilita determinar si dos palabras son similares (cosenos cercanos a 1), no relacionadas (cosenos cercanos a 0) u opuestas (cosenos cercanos a -1).

Estos *embeddings* preentrenados se convirtieron en bloques básicos para muchas aplicaciones de PLN, ya que permitían inicializar modelos con conocimiento semántico previo. Sin embargo, tenían aún limitaciones: un vector por palabra no distingue distintas acepciones (por ejemplo, el término "banco" tendría un único vector pese a sus múltiples significados). Además, estos modelos estáticos no incorporaban contexto más allá de una ventana local.

2.3.1.2 Representaciones contextuales y embeddings de última generación

La evolución natural de los *embeddings* llevó a representaciones contextuales, donde la representación de una palabra varía según el contexto en el que aparece. Los primeros acercamientos incluyeron modelos como ELMo [20], que empleó redes neuronales recurrentes profundas para generar vectores dependientes de la oración completa. Pero el verdadero hito vino con los modelos basados en transformadores, en particular con BERT [9], que introdujo *embeddings* contextuales bidireccionales capaces de capturar matices de significado condicionados tanto por la izquierda como por la derecha del término. Con BERT y sus sucesores (RoBERTa, ALBERT, etc.), cada palabra en un texto tiene un vector diferente ajustado a su contexto específico, resolviendo en gran medida la ambigüedad polisémica.

Estas representaciones de última generación permitieron un salto cualitativo en tareas como etiquetado semántico, resumen automático o respuesta a preguntas, al proveer a los algoritmos de una comprensión más fina del lenguaje. En el dominio logístico, contar con *embeddings* contextuales entrenados en textos técnicos (por ejemplo, documentación de mantenimiento, normas militares) mejora considerablemente la capacidad de los sistemas para entender consultas especializadas o extractos normativos complejos.

2.3.1.3 Modelo de espacio vectorial enriquecido

La codificación utilizada en la representación clásica de bolsa de palabras planteaba problemas fundamentales al no considerar la semántica ni la sintaxis. Los avances descritos en *embeddings* distribuidos y contextuales han enriquecido el tradicional modelo de espacio vectorial hasta transformarlo por completo: ahora cada documento puede representarse, por ejemplo, como la combinación (promedio, suma ponderada o concatenación) de los *embeddings* de sus palabras, obteniendo vectores densos que capturan su temática.

Además, existen técnicas para generar directamente un vector de documento completo, ya sea entrenando modelos tipo *Paragraph2Vec* o usando la representación producida por el token especial [CLS] en modelos como BERT para un texto dado. Estas representaciones vectoriales de documentos alimentan los componentes de recuperación de información y RAG (como veremos más adelante), permitiendo comparar semánticamente consultas en lenguaje natural con documentos técnicos en el almacén de conocimiento.

2.3.2 PLN Profundo

Con el auge del aprendizaje profundo, las metodologías para comprender el lenguaje natural experimentaron transformaciones significativas. Los modelos basados en redes neuronales han superado a los enfoques tradicionales en la mayoría de tareas de PLN, gracias a su capacidad para extraer características de alto nivel y modelar relaciones complejas de forma automática a partir de grandes conjuntos de datos [11].

2.3.2.1 Redes neuronales convolucionales aplicadas al lenguaje

Aunque las CNN se asociaron inicialmente a visión por computador, también encontraron aplicaciones en PLN, especialmente en tareas de clasificación de texto o detección de patrones locales en secuencias (como identificar frases clave). Las redes convolucionales aplicadas a secuencias de texto usan filtros unidimensionales que recorren la oración para extraer n-gramas relevantes de distintas

longitudes. Modelos pioneros como el de [21] demostraron que una CNN sencilla con capas de *pooling max* podía lograr resultados sobresalientes en clasificación de sentimientos o categorización de tópicos. En el contexto logístico, una CNN entrenada sobre reportes de incidentes podría detectar automáticamente si un párrafo describe un fallo de munición, una medida de seguridad, etc., gracias a sus filtros que actúan como detectores de expresiones específicas.

Las ventajas de las CNN en PLN incluyen su capacidad de paralelización (no procesan secuencialmente carácter a carácter, sino en paralelo por n-grama) y su efectividad con conjuntos de datos medianos. Sin embargo, no capturan dependencias de larga distancia tan bien como otras arquitecturas, ya que su campo receptivo es limitado a menos que se añadan muchas capas.

2.3.2.2 RNN

Las redes neuronales recurrentes (RNN) son modelos capaces de reconocer y generar secuencias de datos. A diferencia de las redes neuronales tradicionales, las RNN reutilizan sus neuronas para cada elemento de la secuencia como se observa en la Figura 2-5. Esto las hace capaces de capturar dependencias a lo largo de la secuencia, recordando información previa relevante. Esta naturaleza secuencial las hace ideales para el procesamiento de texto, ya que pueden considerar las palabras anteriores al generar la siguiente.

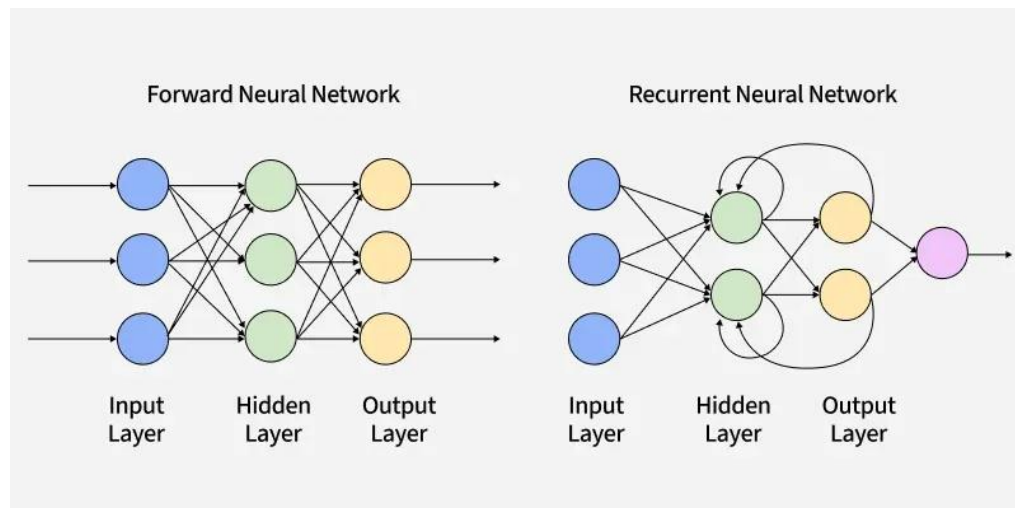


Figura 2-5. Visualización de arquitectura *feed-forward* y recursiva [22]

En la Figura 2-5, es evidente que la RNN de la derecha, a diferencia de la red neuronal de avance directo (*feed-forward*), depende de la salida anterior para calcular el valor subsiguiente. Esto contrasta con la red neuronal de avance directo, donde la entrada y la salida son independientes.

Sin embargo, las RNN básicas presentan una limitación crítica cuando trabajan con secuencias largas: el problema del desvanecimiento del gradiente. Durante el entrenamiento mediante retropropagación a través del tiempo, las señales de error se multiplican en cada paso temporal, atenuándose exponencialmente hasta volverse prácticamente nulas. En contextos militares, donde un manual puede extenderse cientos de páginas, esta limitación resulta especialmente problemática. Una RNN simple podría olvidar que el procedimiento descrito en la página 45 hace referencia a precauciones mencionadas en la página 2, perdiendo coherencia contextual esencial.

Para mitigar esto se introdujeron variantes como LSTM (*Long Short-Term Memory*) y GRU (*Gated Recurrent Unit*) [23], que incorporan puertas de entrada, olvido y salida para controlar el flujo de información, logrando retener datos relevantes durante más tiempo. Estas arquitecturas permitieron que las redes recurrentes manejaran contextos más amplios. Por ejemplo, en un procedimiento operativo extenso, una LSTM puede aprender que una referencia a "el artefacto" en el párrafo actual se refiere a

"la mina naval M-12" mencionada diez frases antes, algo fuera del alcance de un modelo no recurrente básico.

Hasta aproximadamente 2017, las RNN (especialmente las LSTM bidireccionales) representaban la mejor solución para muchas tareas de PLN, incluyendo el etiquetado secuencial o la traducción automática, debido a su capacidad para codificar secuencias de texto en vectores de tamaño fijo que preservaban bastante bien la información contextual. Sin embargo, presentaban inconvenientes: el entrenamiento secuencial limitaba la paralelización y las dependencias muy largas seguían siendo difíciles de aprender completamente.

2.3.2.3 Transformadores

La introducción de los transformadores [8] revolucionó el PLN al resolver gran parte de las limitaciones de las RNN. Los transformadores se basan completamente en mecanismos de atención, que permiten a cada posición de la secuencia valorar la importancia de todas las demás posiciones, capturando dependencias a cualquier distancia sin recurrir al procesamiento recurrente. Al eliminar la recurrencia, los transformadores lograron dos ventajas clave: paralelización total (toda la secuencia se procesa en conjunto, aprovechando mejor la capacidad de cómputo moderno) y capacidad de contexto prácticamente ilimitada (en teoría, pueden modelar dependencias de orden muy superior, limitadas solo por la longitud de secuencia que se decida procesar).

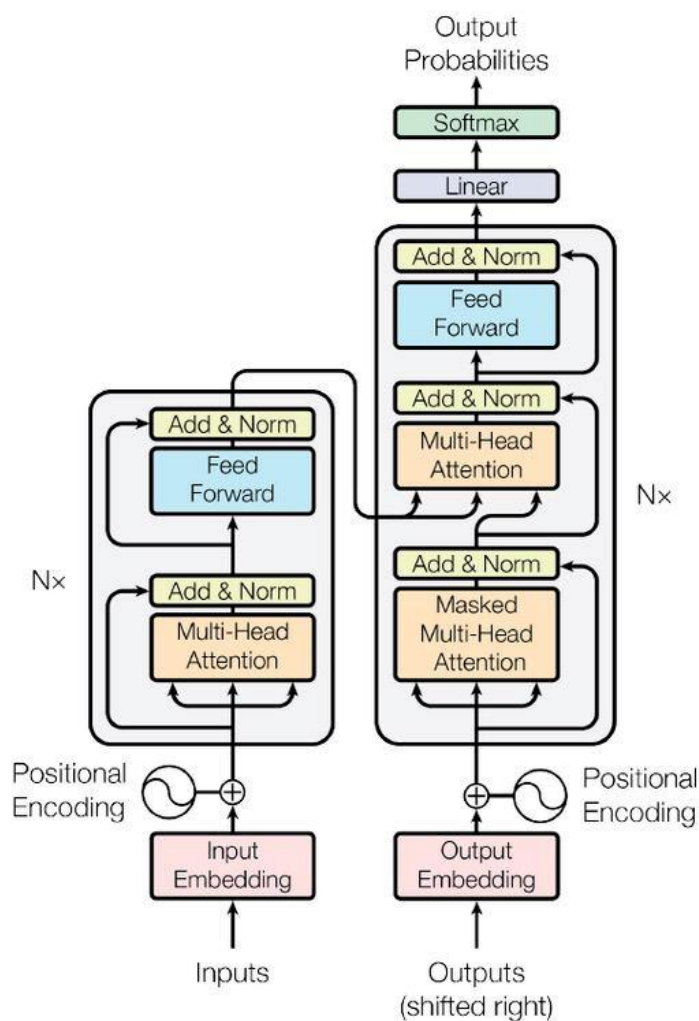


Figura 2-6. Arquitectura de los transformadores [8].

El Transformador original, fue diseñado como un modelo de secuencia a secuencia (seq2seq) para traducción automática. Su arquitectura, ilustrada en la Figura 2-6 , consta de dos componentes principales: un codificador (*encoder*) que procesa la secuencia de entrada completa y genera representaciones contextualizadas, y un decodificador (*decoder*) que genera la secuencia de salida palabra por palabra, atendiendo tanto a su propio contexto generado como a las representaciones del codificador. Esta arquitectura *encoder-decoder* permite transformar una secuencia de un dominio (por ejemplo, una oración en español) en una secuencia de otro dominio (la misma oración traducida al inglés).

El mecanismo de atención constituye el núcleo del *transformer*, permitiendo que cada elemento de la secuencia atienda a todos los demás elementos de forma ponderada. A diferencia de las arquitecturas recurrentes previas que procesaban el texto secuencialmente con memoria limitada, la atención multi-cabeza (*multi-head attention*) [8] permite que el modelo capture múltiples tipos de relaciones simultáneamente: una cabeza de atención puede enfocarse en dependencias sintácticas (por ejemplo, concordancia sujeto-verbo), mientras otra captura relaciones semánticas (correferencias pronominales) o patrones numéricos. Esta capacidad de procesamiento paralelo no solo acelera dramáticamente el entrenamiento, sino que también permite modelar dependencias de largo alcance que las redes recurrentes difícilmente capturaban.

El éxito del *transformer* original desencadenó una proliferación de variantes arquitectónicas especializadas. La comunidad investigadora descubrió rápidamente que no todas las tareas requieren ambos componentes del *transformer*, y que eliminando selectivamente el codificador o el decodificador se podían obtener modelos más eficientes para casos de uso específicos. Esta evolución dio lugar a una taxonomía de tres familias arquitectónicas claramente diferenciadas.

Codificadores puros (Encoder-only): Estos modelos se preentrenan típicamente mediante tareas de enmascaramiento de palabras (*masked language modeling*): se ocultan aleatoriamente algunos tokens de la entrada y el modelo debe predecirlos basándose en el contexto circundante. Esta tarea de entrenamiento obliga al modelo a desarrollar representaciones profundas del significado contextual de cada palabra, capturando ambigüedades léxicas, relaciones sintácticas y matices semánticos [9].

Decodificadores puros (Decoder-only): Los modelos *decoder-only* generan texto de forma autorregresiva y unidireccional: predicen el siguiente token basándose únicamente en los tokens previos, sin acceso a contexto futuro. La familia GPT (*Generative Pre-trained Transformer*) de OpenAI popularizó esta arquitectura, demostrando que modelos entrenados simplemente para predecir la siguiente palabra en textos masivos desarrollan capacidades sorprendentes de comprensión y generación [24].

Modelos Encoder-Decoder: Los modelos que mantienen la arquitectura completa del *transformer* original combinan las fortalezas de ambos componentes. El codificador procesa bidireccionalmente la entrada completa, generando representaciones contextualizadas, mientras el decodificador genera la salida autorregresivamente, atendiendo a las representaciones del codificador con atención cruzada.

La tendencia reciente favorece claramente los decodificadores puros para aplicaciones de propósito general. El paradigma de *in-context learning*, donde se proporcionan instrucciones y ejemplos dentro del *prompt* para guiar el comportamiento del modelo sin reentrenamiento, ha resultado extraordinariamente efectivo con decodificadores. Este fenómeno, denominado *prompt engineering*, permite adaptar el modelo a nuevas tareas simplemente reformulando la entrada, democratizando el acceso a capacidades avanzadas sin necesidad de *expertise* en *machine learning*.

2.3.3 Generación de Lenguaje Natural

La arquitectura de decodificadores descrita en la sección anterior ha transformado radicalmente las capacidades de generación de lenguaje natural (*Natural Language Generation*, NLG). Mientras que los primeros sistemas de generación se basaban en plantillas rígidas y gramáticas formales, los modelos

basados en *transformer* aprenden de los patrones estadísticos presentes en corpus masivos de texto, desarrollando capacidades emergentes que van más allá de su objetivo de entrenamiento explícito.

La evolución de los modelos *transformer* ha estado marcada por un crecimiento exponencial en el número de parámetros. Kaplan et al. demostraron en 2020 que el rendimiento de los modelos de lenguaje escala predeciblemente con tres factores: tamaño del modelo, cantidad de datos de entrenamiento, y recursos computacionales empleados [25]. Esta observación desencadenó una carrera por modelos cada vez más grandes, pasando de BERT con 345 millones de parámetros en 2018, a GPT-3 con 175 mil millones en 2020, hasta modelos propietarios contemporáneos con parámetros estimados en el rango del billón (Figura 2-7).

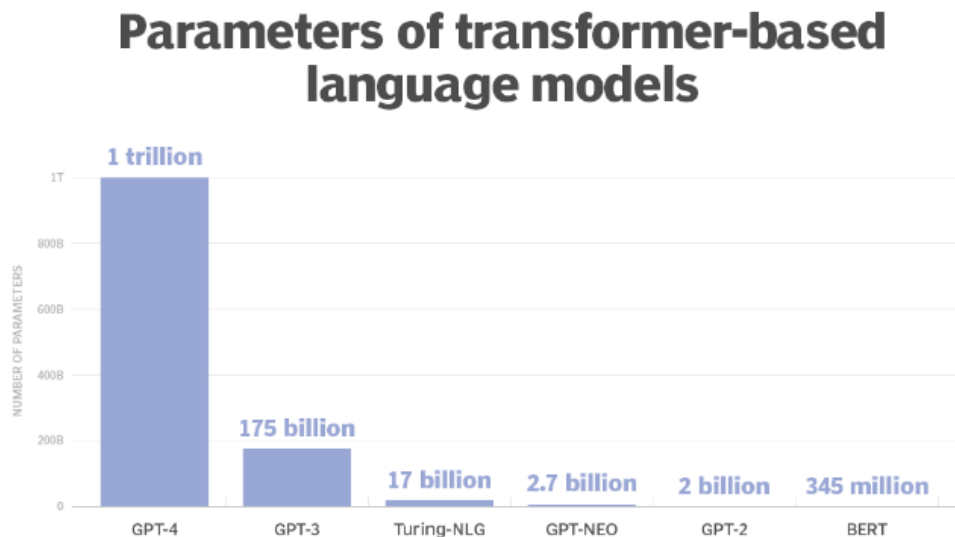


Figura 2-7. Evolución del número de parámetros en modelos de lenguaje basados en Transformers [26]

El preentrenamiento de modelos *decoder-only* mediante objetivos simples de predicción de siguiente palabra utilizando millones de parámetros produce representaciones internas sorprendentemente ricas. Esta capacidad de seguir instrucciones arbitrarias en lenguaje natural ha democratizado el acceso a tecnologías avanzadas de NLP, eliminando la necesidad de conjuntos de datos etiquetados extensos o *expertise in machine learning* para adaptar el modelo a nuevas tareas.

Sin embargo, esta versatilidad tiene un precio crítico: los modelos generativos producen texto basándose en patrones estadísticos aprendidos durante el entrenamiento, no en verificación de hechos o acceso a bases de conocimiento estructuradas. Esto resulta en el fenómeno conocido como alucinación [27], donde el modelo genera afirmaciones plausibles, pero factualmente incorrectas con confianza indistinguible de respuestas veraces.

Las alucinaciones son especialmente problemáticas en dominios técnicos especializados. Un modelo puede afirmar con aparente certeza que el NEQ unitario de un cartucho específico es 0,25 kg cuando el valor real es 1,8 kg, basándose en haber visto valores similares en su entrenamiento, pero sin verificar la especificación exacta del modelo concreto consultado. En logística militar, donde decisiones operacionales críticas dependen de la precisión factual de la información (cálculos de capacidad de polvorines, compatibilidad de grupos de riesgo, procedimientos de seguridad), esta limitación es inaceptable.

La comunidad investigadora ha desarrollado varias aproximaciones para abordar estas limitaciones, con grados variables de éxito:

El ajuste fino supervisado sobre datos del dominio específico puede mejorar la precisión factual en ese dominio [28], pero requiere datasets anotados extensos, introduce el riesgo de sobreajuste que

degrada capacidades generales, y no resuelve el problema de conocimiento congelado (el modelo seguirá desconociendo información posterior al *fine-tuning*).

El *prompt engineering* avanzado mediante *chain-of-thought* [29] puede mejorar el razonamiento complejo, instruyendo al modelo a explicitar su proceso de razonamiento paso a paso, pero sigue limitado por el conocimiento paramétrico del modelo y no garantiza corrección factual.

La aproximación que ha demostrado mayor efectividad en entornos de producción es la Generación Aumentada por Recuperación (RAG) [13], que aborda frontalmente las limitaciones mencionadas mediante la integración de conocimiento externo dinámico en tiempo de inferencia. Como se detallará en la sección 2.4.3, RAG mantiene el conocimiento especializado fuera del modelo, en bases documentales actualizables, eliminando la dependencia del conocimiento paramétrico para información factual específica mientras preserva las capacidades generativas del LLM para sintetizar y comunicar esa información de forma accesible.

En los siguientes apartados se analizará cómo funcionan estas aproximaciones.

2.3.4 Ingeniería de prompts

La ingeniería de *prompts* se ha convertido en un método clave para mejorar el rendimiento de modelos de lenguaje grandes preentrenados (LLMs) y modelos de visión-lenguaje (VLMs). Consiste en crear instrucciones específicas para la tarea, denominadas *prompts*, que guían la salida del modelo sin modificar sus parámetros. Este enfoque mejora la adaptabilidad de LLMs y VLMs, permitiéndoles manejar una amplia variedad de tareas y dominios de manera efectiva.

A diferencia de métodos tradicionales que requieren reentrenamiento o ajuste fino extenso, la ingeniería de *prompts* ajusta el comportamiento del modelo mediante instrucciones cuidadosamente elaboradas, ofreciendo una alternativa más eficiente y flexible. Permite que estos modelos destaquen en diversas aplicaciones sin modificaciones significativas. La investigación continua en esta área introduce técnicas innovadoras, destacando la creciente importancia de la ingeniería de *prompts* en la configuración de la adaptabilidad y utilidad de la IA en todos los sectores.

2.3.4.1 Aprendizaje en contexto (ICL)

El aprendizaje en contexto es un método en el que los modelos de lenguaje pueden aprender tareas al recibir solo unos pocos ejemplos como demostraciones. El aprendizaje en contexto, introducido en el artículo original de GPT-3 [30], permite que los modelos de lenguaje realicen tareas proporcionando solo unos pocos ejemplos. En este enfoque, se crea un *prompt* que contiene pares entrada-salida que demuestran la tarea. Luego, se añade una entrada de prueba al final del *prompt* y el modelo predice los siguientes tokens basándose en los ejemplos proporcionados. Para realizar predicciones precisas, el modelo debe comprender la distribución de entrada (por ejemplo, noticias financieras o generales), la distribución de salida (por ejemplo, sentimiento positivo/negativo o temas), las relaciones entrada-salida (por ejemplo, clasificación de sentimiento o tema) y el formato.

Prompting de cero disparos (*zero-shot prompting*): El *prompting* de cero disparos elimina la necesidad de grandes cantidades de datos de entrenamiento, dependiendo en su lugar de *prompts* bien diseñados para dirigir al modelo hacia la realización de nuevas tareas. En este enfoque, se proporciona al modelo una descripción de la tarea dentro del *prompt*, pero no se incluyen datos etiquetados con ejemplos específicos de entrada-salida. El modelo utiliza entonces su conocimiento *preentrenado* para hacer predicciones sobre la nueva tarea basándose únicamente en el *prompt* dado.

Prompting de pocos disparos (*few-shot prompting*): El *prompting* de pocos disparos implica proporcionar al modelo un pequeño número de ejemplos entrada-salida para ayudarlo a comprender una tarea específica, a diferencia del *prompting* de cero disparos, que no ofrece ejemplos. Unos pocos ejemplos de alta calidad pueden mejorar significativamente el rendimiento del modelo en tareas complejas. Sin embargo, incluir estos ejemplos requiere tokens adicionales, lo que puede convertirse en

una limitación para entradas largas. La selección y disposición de los ejemplos en el *prompt* también son cruciales, ya que pueden influir en el comportamiento del modelo, con sesgos potenciales. Aunque el *prompting* de pocos disparos potencia las capacidades de modelos grandes, el diseño cuidadoso del *prompt* es esencial para optimizar el rendimiento y minimizar sesgos no deseados.

Prompting basado en instrucciones: El *Instruction Prompting* se centra en la capacidad de los modelos de IA generativa, particularmente los modelos de lenguaje grandes (LLMs), para seguir instrucciones escritas en lenguaje natural. Este enfoque permite que los modelos manejen tareas nuevas y no vistas previamente interpretando y ejecutando estas instrucciones sin necesidad de datos de entrenamiento específicos de la tarea.

2.3.4.2 Mejorar las capacidades de razonamiento

Los Modelos de Lenguaje Grandes (LLMs) a menudo tienen dificultades con el razonamiento complejo, lo que limita su potencial. Sin embargo, la investigación ha demostrado que, cuando se les solicita de manera efectiva, los modelos suficientemente grandes pueden desarrollar capacidades de razonamiento para abordar tales desafíos [31].

Cadena de pensamiento (CoT): El razonamiento *Chain-of-Thought* (CoT) permite a los LLMs realizar razonamiento paso a paso, mejorando su capacidad para manejar tareas complejas mediante aprendizaje en contexto. Solicitar al modelo que genere pasos de razonamiento intermedios mejora significativamente su rendimiento en tareas aritméticas, de sentido común y de razonamiento simbólico. El *prompting* CoT descompone problemas en pasos más pequeños, proporciona un proceso de razonamiento interpretable y es ampliamente aplicable a tareas como problemas matemáticos, razonamiento de sentido común y manipulación simbólica. Puede implementarse fácilmente en modelos grandes preentrenados incluyendo ejemplos CoT en *prompts* de pocos disparos, convirtiéndolo en un método valioso para potenciar las capacidades de razonamiento [29].

Árbol de pensamientos (ToT): El enfoque *Tree of Thoughts* (ToT) se basa y generaliza el método *Chain of Thoughts* (CoT), permitiendo que los modelos de lenguaje exploren unidades coherentes de texto, denominadas pensamientos, que actúan como pasos intermedios en la resolución de problemas. Este nuevo marco para inferencia de modelos de lenguaje que generaliza sobre el popular enfoque *Chain of Thought* para *prompting*, habilitando la exploración sobre unidades coherentes de texto (pensamientos) que sirven como pasos intermedios hacia la resolución de problemas. ToT mantiene activamente un árbol estructurado de estos pensamientos, donde cada pensamiento representa una secuencia de lenguaje significativa. Esta estructura permite al modelo autoevaluar el progreso evaluando cómo cada idea intermedia contribuye a resolver el problema mediante un proceso de razonamiento deliberado expresado en lenguaje natural. Los experimentos mostraron que ToT mejora significativamente las habilidades de resolución de problemas de los modelos de lenguaje en tres tareas novedosas que requieren planificación o búsqueda no trivial: *Game of 24*, *Creative Writing*, y *Mini Crosswords* [32].

ReAct: El enfoque ReAct permite generar trazas de razonamiento y acciones específicas de la tarea de manera intercalada, creando una sinergia más fuerte entre ambas. Este enfoque explora el uso de LLMs para generar tanto trazas de razonamiento como acciones específicas de la tarea de manera intercalada, permitiendo una mayor sinergia entre ambas. Las trazas de razonamiento ayudan al modelo a desarrollar, monitorear y ajustar planes de acción mientras maneja excepciones. Las acciones, por su parte, permiten al modelo interactuar con fuentes externas como bases de conocimiento o entornos para recopilar información adicional. ReAct se ha aplicado a una amplia gama de tareas de lenguaje y toma de decisiones, demostrando un rendimiento superior en comparación con métodos de vanguardia, al tiempo que mejora la interpretabilidad humana y la confianza en el modelo. Concretamente, en respuesta a preguntas (HotpotQA) y verificación de hechos (Fever), ReAct supera problemas de alucinación y propagación de errores prevalentes en el razonamiento *chain-of-thought* mediante interacción con una API simple de Wikipedia, generando trayectorias de resolución de problemas similares a las humanas que son más interpretables que las baselines sin trazas de razonamiento [33].

2.3.5 Sistemas Multiagentes

Cuando un sistema de IA necesita resolver tareas complejas, delegar responsabilidades en agentes especializados puede resultar más eficaz que depender de un único modelo monolítico. En el contexto de la gestión documental, esto cobra especial relevancia. En lugar de centrarnos solo en potenciar un modelo individual, ahora el énfasis está en combinar diferentes capacidades

Los sistemas multiagente se proponen como una estructura donde múltiples agentes —frecuentemente instancias del propio LLM con especializaciones y enfoques distintos— colaboran para descomponer problemas, consultar fuentes externas, verificar hipótesis y consensuar decisiones. El interés académico y aplicado por esta aproximación crece por dos motivos complementarios: por un lado, amplía la cobertura funcional sin necesidad de entrenamientos monolíticos; por otro, permite introducir controles y garantías que son difíciles de imponer a un único modelo generativo operando en “caja negra”.

En su definición más extendida, un sistema multiagente es un entorno en el que agentes autónomos basados en LLM (o en combinaciones de LLM y programas deterministas) persiguen objetivos compartidos mediante comunicación estructurada, memoria común y protocolos de coordinación. Frente al agente único de propósito general, la organización multiagente introduce separación de responsabilidades (p. ej., comprensión, recuperación, síntesis, verificación, planificación), con beneficios en interpretabilidad, escalabilidad y robustez. La literatura reciente señala tres motivaciones recurrentes: (i) descomposición del trabajo intelectual para reducir la carga de razonamiento por contexto; (ii) acceso a herramientas externas —buscadores, bases de datos, optimizadores— mediante agentes puentes; y (iii) control de calidad a través de críticas cruzadas, auto-consistencia y validadores.

Aunque no existe un estándar universal, pueden reconocerse patrones convergentes. La mayoría de propuestas adoptan una capa de orquestación que traduce una intención en un plan jerárquico de subtarefas. Este orquestador puede ser un LLM con políticas de planificación (*plan-and-solve*, *reflect-and-refine*) o un planificador programático que enrutará a los agentes adecuados en función de reglas y señales de confianza. Los agentes se implementan como roles con entradas y salidas tipadas (p. ej., mensajes JSON validados), y se comunican mediante un bus de mensajería que conserva el historial para auditoría y para retroalimentar la memoria compartida.

La arquitectura multiagente típica incluye un orquestador que analiza la intención del usuario y decide qué agentes activar. Este orquestador puede ser otro LLM con instrucciones específicas o un sistema basado en reglas que enruta según patrones reconocidos en la consulta. Los agentes se diseñan con responsabilidades claras: unos acceden a documentación vectorizada para resolver dudas normativas, otros ejecutan consultas exactas contra bases de datos relacionales, y algunos actúan como verificadores que comprueban la coherencia de las respuestas antes de presentarlas al usuario.

Una aportación distintiva del enfoque multiagente es la formalización de los intercambios. Lejos del diálogo libre, los mensajes se estructuran y validan con esquemas que describen campos obligatorios, dominios de valores y unidades. Tal formalización habilita varias propiedades valiosas: composición fiable (los agentes consumen exactamente lo que esperan), verificación automática (precondiciones y postcondiciones), y trazabilidad (cada transición de estado queda explicada por un mensaje firmado y versionado). Esta “gramática de la colaboración” no solo mejora la robustez, sino que reduce la ambigüedad típica del lenguaje natural en tareas sensibles.

La coordinación multiagente se ha explorado con múltiples estrategias. Los esquemas de debate piden a agentes con perspectivas distintas que propongan y critiquen soluciones sucesivas; la auto consistencia busca converger repitiendo el razonamiento con semillas o itinerarios independientes y eligiendo la respuesta modal o mejor puntuada; las políticas de crítica introducen un “revisor” con instrucciones orientadas a encontrar fallos lógicos, lagunas de evidencia o inconsistencias de formato; por último, los circuitos de reflexión (*reflection loops*) documentan supuestos, evalúan errores y relanzan

el proceso incorporando la diagnosis. Cada estrategia aporta beneficios distintos: el debate favorece la cobertura de alternativas, la auto consistencia reduce variabilidad estocástica, la crítica especializada eleva la calidad y la reflexión estabiliza el aprendizaje procedimental del sistema.

Una limitación conocida de los LLMs es su tendencia a alucinar hechos y a degradarse fuera de distribución. Por ello, el estado del arte converge hacia diseños donde la generación se ancla a información verificable mediante RAG y a cálculos exactos mediante el uso de herramientas especializadas. En un sistema multiagente, este anclaje se formaliza: un agente de recuperación aplica filtros de metadatos (fecha, fuente, jurisdicción) y devuelve fragmentos con procedencia; el agente generador debe citar o referirse a esa evidencia en su síntesis; y un agente verificador comprueba la coherencia inferencial entre evidencia y conclusión. De manera complementaria, agentes de herramientas invocan funciones deterministas (consultas SQL, solucionadores, validadores de unidades y dimensiones) para sustituir con cálculo exacto lo que, en un modelo puramente lingüístico, sería especulación.

2.4 Recuperación de Información

En un entorno con ingentes cantidades de datos documentales, la recuperación de información (IR, por sus siglas en inglés) consiste en localizar material relevante dentro de grandes colecciones para satisfacer una necesidad específica de información. En la gestión logística de municiones, esto se traduce en ser capaces de encontrar rápidamente la norma, ficha técnica o registro operacional que responda a una consulta determinada (por ejemplo, localizar todos los documentos que mencionen un lote de munición defectuoso o extraer la última directiva sobre almacenamiento de cierto explosivo). La recuperación de información se vuelve crítica cuando se integran sistemas de generación de lenguaje (LLM) con bases de conocimiento: los modelos necesitan acceder al documento adecuado en el momento preciso para brindar respuestas precisas y actualizadas.

Tradicionalmente, los sistemas IR se han basado en índices invertidos y coincidencia de palabras clave (algoritmos clásicos como TF-IDF, BM25), donde la relevancia de un documento se estima según la presencia y frecuencia de términos de la consulta. Si bien estos métodos son rápidos y eficaces para textos bien formateados, tienen dificultades para comprender sinónimos o contexto semántico. Por ejemplo, una búsqueda de "vigencia del propulsor X" podría no arrojar un documento que hable de "fecha de caducidad del propelente X" si no comparten palabras exactas.

El estado del arte ha avanzado hacia métodos de recuperación semántica. Empleando *embeddings* vectoriales, tanto las consultas como los documentos pueden proyectarse a un espacio vectorial común donde la similitud se mide por cercanía geométrica (coseno, Euclídea, etc.). Esto permite recuperar documentos conceptualmente relevantes, aunque no compartan palabras textuales. Por ejemplo, un sistema que represente "vigencia del propulsor X" y "caducidad del propelente X" en vectores cercanos podrá enlazar la consulta con el documento correcto.

2.4.1 Motores de búsqueda inteligentes y ajuste fino

Los motores de búsqueda modernos combinan lo mejor de ambos mundos: utilizan índices tradicionales para filtrar rápidamente candidatos (p. ej., cientos de documentos que contengan al menos algún término de la consulta) y luego reordenan esos candidatos mediante modelos más sofisticados, como redes neuronales que estiman relevancia contextual.

Modelos como ColBERT [34] emplean BERT para obtener representaciones semánticas de consultas y pasajes, refinando los resultados con gran precisión. ColBERT introduce una arquitectura de interacción tardía que codifica independientemente la consulta y el documento usando BERT y luego emplea un paso de interacción económico pero potente que modela su similitud de grano fino. Al retrasar y sin embargo retener esta interacción de grano fino, ColBERT puede aprovechar la expresividad de los modelos de lenguaje profundos mientras simultáneamente gana la capacidad de precomputar representaciones de documentos offline. Los autores demostraron que ColBERT es dos órdenes de

magnitud más rápido que modelos basados en BERT existentes, todo mientras impacta mínimamente la calidad de re-ranking y superando cada *baseline* no-BERT. ColBERT emplea BERT para obtener representaciones semánticas de consultas y pasajes, refinando los resultados con gran precisión.

Un aspecto crítico es la capacidad de ajustar finamente estos sistemas de recuperación a los datos específicos de la organización. Es aquí donde entra el *Parameter-Efficient Fine Tuning* (PEFT): en vez de reentrenar un modelo de lenguaje entero para las peculiaridades de los datos logísticos, se ajustan solo algunos parámetros (por ejemplo, capas adicionales especializadas en terminología militar) que se pueden entrenar con un conjunto limitado de ejemplos de búsquedas reales y sus documentos relevantes. Este ajuste fino eficiente reduce el coste computacional y permite iterar rápidamente conforme se recopilan nuevas interacciones de los usuarios con el sistema de búsqueda.

Además, la evaluación continua de la calidad de recuperación es práctica estándar: mediante métricas como *precision@k*, *recall*, MRR, etc., se valida que el motor esté encontrando lo que debe. En logística militar, donde ciertas consultas podrían ser raras pero críticas (p. ej., procedimientos de emergencia para un tipo específico de munición), se pone especial atención en optimizar la recuperación incluso en esos casos extremos, a menudo incorporando sinónimos y jerga del dominio en los modelos.

2.4.2 Búsqueda Semántica

Con la adopción amplia de representaciones vectoriales, ha surgido un ecosistema de bases de datos vectoriales (ver Sección 2.4.3) diseñadas para soportar de forma eficiente las operaciones de búsqueda por similitud de *embeddings*. En lugar de índices invertidos de palabras, estos sistemas indexan los vectores de los documentos (o pasajes de documentos) y, dada la representación vectorial de una consulta, pueden retornar los *k* vectores más cercanos (y por tanto los *k* documentos más relevantes semánticamente) en cuestión de milisegundos aun con millones de elementos indexados.

Técnicas avanzadas de indexación vectorial, como FAISS (*Facebook AI Similarity Search*) o HNSW (*Hierarchical Navigable Small World*), permiten realizar aproximaciones muy eficientes a la búsqueda *k*-NN (vecinos más cercanos) en espacios de alta dimensión, manteniendo un balance entre velocidad y precisión configurable. Estas herramientas abordan el problema de una mejor utilización de GPUs para búsqueda de similitud, proponiendo un diseño novedoso para *k-selection* que opera hasta el 55% del rendimiento pico teórico, habilitando una implementación de vecino más cercano que es 8.5x más rápida que el estado del arte previo en GPU. Aplicaron esto en diferentes escenarios de búsqueda de similitud, proponiendo diseños optimizados para búsqueda de fuerza bruta, aproximada y de dominio comprimido basada en cuantización de producto. En todos estos *setups*, superaron el estado del arte por amplios márgenes. FAISS ha sido integrada en plataformas de búsqueda de nueva generación dentro del ámbito empresarial y académico, permitiendo realizar búsquedas eficientes en conjuntos de datos de escala de miles de millones de vectores [35].

Estas herramientas han sido integradas en plataformas de búsqueda de nueva generación dentro del ámbito empresarial. En la práctica de la logística, esto implica que, ante una consulta compleja, el sistema puede apoyarse en la similitud de *embeddings* para encontrar fragmentos de manuales u órdenes con contenido relacionado, aunque expresado con palabras distintas. Por ejemplo, una búsqueda sobre "procedimiento de eliminación de munición obsoleta" podría recuperar un apartado dentro de una directiva que habla de "protocolo de desmilitarización de explosivos antiguos", incluso si no comparte palabras exactas, gracias a que sus *embeddings* captan que ambos tratan de desechar material viejo de forma segura.

2.4.3 Generación Aumentada por Recuperación (RAG)

Los LLMs, como la serie GPT, la serie LLama y otros modelos como Gemini, han logrado un éxito notable en el procesamiento del lenguaje natural, demostrando un rendimiento superior en diversos puntos de referencia. A pesar de estos avances, los LLMs tienen limitaciones, particularmente en el manejo de consultas específicas de un dominio o altamente especializadas. Un problema común es la

generación de información incorrecta, o "alucinaciones", especialmente cuando las consultas van más allá de los datos de entrenamiento del modelo o requieren información actualizada. Estas debilidades hacen que el despliegue de LLMs como soluciones de caja negra en entornos de producción del mundo real sea poco práctico sin salvaguardas adicionales. Un enfoque prometedor para mitigar estas limitaciones es la Generación Aumentada por Recuperación (RAG, por sus siglas en inglés), que integra la recuperación de datos externos en el proceso generativo, mejorando así la capacidad del modelo para proporcionar respuestas precisas y relevantes.

RAG, introducido por Lewis a mediados de 2020 [13], representa un enfoque novedoso dentro del campo de los LLMs, mejorando las tareas generativas. Específicamente, RAG implica un paso inicial de recuperación donde los LLMs consultan una fuente de datos externa para obtener información relevante antes de proceder a responder preguntas o generar texto, como se observa en el esquema de la Figura 2-8. Este proceso no solo informa la fase de generación posterior, sino que también asegura que las respuestas estén fundamentadas en evidencia recuperada, mejorando significativamente la precisión y relevancia de la salida. La recuperación dinámica de información desde bases de conocimiento durante la fase de inferencia permite a RAG abordar problemas como la generación de contenido incorrecto, comúnmente referido como "alucinaciones". La integración de RAG en los LLMs ha experimentado una rápida adopción y se ha convertido en una tecnología fundamental para refinar las capacidades de los chatbots y hacer que los LLMs sean más viables para aplicaciones prácticas.

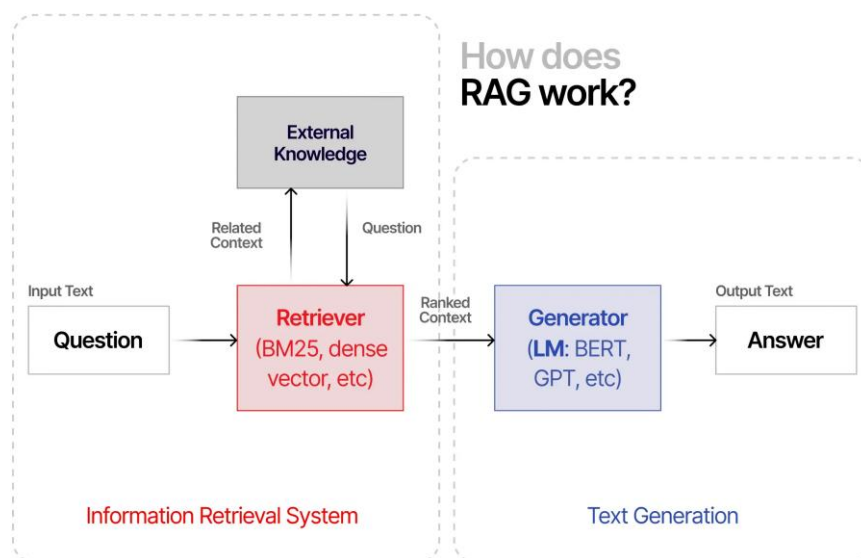


Figura 2-8. Esquema de funcionamiento de Generación Aumentada por Recuperación (RAG) [36].

Por lo tanto, un sistema de RAG se comporta como un modelo de secuencia a secuencia (seq2seq), que acepta una secuencia como entrada y produce una secuencia como salida. La única diferencia es que, antes de enviar la entrada al modelo, el recuperador encuentra información relevante para esa entrada y la envía junto con la entrada para producir una mejor salida. De esta manera, el conocimiento no paramétrico complementa el conocimiento ya presente en el modelo [13].

La inclusión de conocimiento externo permite una fácil modificación de la información dentro del modelo. Este enfoque también mejora la precisión del modelo al proporcionar un medio confiable para verificar su conocimiento. En consecuencia, el modelo puede acceder a información relevante para mejorar su precisión.

Este método aumenta la versatilidad del sistema, permitiéndole evitar la necesidad de reentrenamiento cuando se trata de tareas complejas basadas en conocimiento.

Un flujo típico de RAG consta de tres pasos principales:

2.4.3.1 Recuperación de contextos relevantes

Dada la consulta del usuario (por ejemplo, "¿Qué mantenimiento preventivo se recomienda para los torpedos modelo X?"), el sistema realiza una búsqueda en su base documental (manuales de mantenimiento, órdenes técnicas, etc.) para extraer los fragmentos más pertinentes (por ejemplo, el apartado del manual del torpedo X que habla de mantenimiento preventivo). Esta búsqueda emplea las técnicas de IR antes descritas, a menudo mediante *embeddings* y bases vectoriales para capturar correspondencias semánticas. Los métodos de recuperación pueden incluir tanto búsqueda léxica tradicional (BM25) como búsqueda densa mediante similitud de *embeddings* (utilizando sistemas como FAISS o HNSW), o incluso enfoques híbridos que combinan ambas estrategias para maximizar tanto la precisión como el *recall*.

2.4.3.2 Construcción del *prompt* combinado

Los fragmentos de texto recuperados se combinan con la consulta original para conformar un *prompt* extendido que se proporcionará al modelo de lenguaje. Este *prompt* típicamente incluye una formulación explícita del mandato (por ejemplo: "Usando la información a continuación, responde la pregunta...") seguida del contenido de los documentos relevantes y finalmente la pregunta del usuario. El objetivo es contextualizar al LLM con información factual específica antes de que genere la respuesta. Este paso de construcción del *prompt* es crucial porque determina cómo el modelo interpretará y utilizará la información recuperada. Las estrategias avanzadas de *prompt engineering* incluyen la estructuración jerárquica de la información, la inclusión de metadatos de los documentos (fecha, fuente, nivel de autoridad), y la priorización de fragmentos según su relevancia y fiabilidad.

2.4.3.3 Síntesis (generación de la respuesta)

Finalmente, el LLM toma el *prompt* combinado y produce una respuesta en lenguaje natural. Durante esta generación, el modelo mezcla su conocimiento interno con los datos proporcionados en el contexto. Idealmente, extrae los puntos clave de los documentos recuperados y los reformula para responder a la pregunta del usuario de manera completa y coherente. En sistemas avanzados, esta fase puede incluir mecanismos de verificación de consistencia, donde el modelo evalúa si la información recuperada es suficiente y coherente antes de generar la respuesta final, o incluso puede desencadenar búsquedas adicionales si detecta lagunas de información.

La arquitectura RAG presenta varias ventajas fundamentales para aplicaciones críticas como la logística militar. En primer lugar, mantiene las respuestas actualizadas, ya que se apoya en datos externos que pueden ser tan recientes como la última actualización de la base de datos, eliminando la necesidad de reentrenar el modelo cada vez que se publica una nueva directiva o manual. En segundo lugar, aporta transparencia y trazabilidad: es posible mostrar al usuario los documentos fuente usados para la respuesta, lo que permite verificar la información y aumenta la confianza en el sistema. En tercer lugar, permite abarcar información detallada de nicho sin necesidad de entrenar el modelo de lenguaje en cada dato específico, lo que es especialmente valioso cuando se trata de información clasificada o altamente especializada que no puede incluirse en el entrenamiento del modelo base. En cuarto lugar, facilita el control de calidad y la auditoría, ya que cada respuesta puede rastrearse hasta sus documentos fuente, permitiendo identificar y corregir información errónea o desactualizada en la base de conocimiento.

Esto es especialmente beneficioso en logística militar, donde mucho conocimiento es altamente específico, clasificado o cambia con el tiempo (nuevas directivas, nuevos equipos, actualizaciones de procedimientos de seguridad, modificaciones en normativas internacionales, etc.). La capacidad de actualizar la base de conocimiento sin reentrenar el modelo representa un ahorro considerable de recursos computacionales y permite una respuesta ágil a cambios operativos.

Por otro lado, también conlleva retos técnicos y operativos significativos que deben abordarse cuidadosamente. Requiere una orquestación cuidadosa entre los componentes de recuperación y generación. Si la búsqueda falla en encontrar el documento adecuado debido a una mala formulación de la consulta, *embeddings* de baja calidad, o una indexación deficiente, el LLM generará una respuesta posiblemente incorrecta o vaga basándose únicamente en su conocimiento paramétrico, que puede estar desactualizado o ser insuficiente para el dominio específico.

Por otra parte está el tema de la fusión de información: si se recuperan múltiples documentos que contienen información contradictoria o que requieren síntesis compleja, el sistema debe ser capaz de identificar estas contradicciones, priorizarlas según la autoridad de la fuente, y sintetizarlas de manera coherente sin introducir inconsistencias.

Finalmente, el balance entre la cantidad de contexto recuperado y la ventana de contexto disponible del LLM representa un compromiso constante: demasiado contexto puede saturar la capacidad del modelo y diluir la información relevante, mientras que muy poco contexto puede resultar en respuestas incompletas o inexactas.

La adopción de RAG en escenarios como la Defensa es un cambio radical en la forma en que el personal accede a la información: en lugar de hojear manuales extensos, navegar por sistemas documentales complejos, o depender exclusivamente de expertos humanos con disponibilidad limitada, el personal operativo podría realizar preguntas en lenguaje natural a un asistente inteligente que, fundamentado en la documentación oficial, entregue respuestas precisas con referencias explícitas a las fuentes.

Esto aumentaría significativamente la eficiencia operativa al reducir el tiempo de búsqueda de información. Sin embargo, el éxito de estos sistemas depende de varios factores: la calidad y actualización constante de la base documental, la validación rigurosa del sistema mediante evaluaciones, la supervisión continua para detectar y corregir errores, y la integración adecuada con los flujos de trabajo existentes para maximizar la adopción por parte del personal.

En el contexto militar específico, consideraciones adicionales incluyen: el manejo apropiado de documentos clasificados con diferentes niveles de acceso, la capacidad de operar en entornos con conectividad limitada mediante despliegues *on-premise*, la auditoría exhaustiva de todas las consultas y respuestas para cumplimiento regulatorio, y la resiliencia ante posibles ataques adversarios que intenten manipular el sistema mediante consultas maliciosas o envenenamiento de la base de conocimiento.

2.5 Almacenamiento y Bases de Datos

El manejo eficaz de la información extraída y generada por los sistemas de IA requiere una infraestructura de almacenamiento de datos robusta y flexible. En el contexto de la logística militar, coexisten datos estructurados (por ejemplo, inventarios, catálogos) con datos no estructurados (documentos, informes de texto libre), así como nuevas formas de datos vectoriales provenientes de *embeddings*. Para atender estas necesidades, el estado del arte abarca desde bases de datos relacionales tradicionales hasta novedosas bases de datos vectoriales, pasando por diversas variantes de NoSQL, cada una aportando ventajas particulares.

2.5.1 Bases de Datos Relacionales

Las bases de datos relacionales (SQL) han sido la piedra angular del almacenamiento de datos durante décadas. Se basan en un modelo estructurado de tablas con filas y columnas, donde la información se organiza siguiendo un esquema predefinido.

Un concepto clave en el diseño relacional es la normalización, un proceso sistemático para estructurar los esquemas de base de datos con el objetivo de minimizar la redundancia de datos y eliminar anomalías de inserción, actualización o borrado. Mediante la normalización (primera forma normal, segunda forma normal, tercera forma normal, etc.), los datos se distribuyen en múltiples tablas relacionadas por claves foráneas, evitando duplicaciones como tener el nombre de un proveedor repetido en 20 filas distintas cuando podría estar en una sola tabla de proveedores. En sistemas logísticos militares, la normalización hasta cierto nivel garantiza consistencia —por ejemplo, la información de una determinada munición (peso, calibre, fabricante, especificaciones técnicas, etc.) se almacena una única vez en una tabla maestra y no se repite en cada registro de inventario donde aparece esa munición—. Sin embargo, un esquema demasiado normalizado puede impactar el rendimiento de consultas complejas que requieran múltiples uniones (*joins*), especialmente cuando se necesita agregar información de varias tablas para generar reportes.

Los sistemas gestores de bases de datos relacionales (SGBD) modernos, como Oracle, MySQL, PostgreSQL o Microsoft SQL Server, han incorporado numerosas optimizaciones (índices *B-tree* y de dispersión (*hash*), almacenamiento en caché inteligente de consultas, planes de ejecución avanzados basados en estadísticas, procesamiento paralelo de consultas) que las mantienen vigentes y eficientes para gran parte de las cargas de trabajo. Además, estándares como SQL permiten manejar las bases de datos de forma declarativa, especificando qué datos se necesitan en lugar de cómo obtenerlos, y características como transacciones ACID (Atomicidad, Consistencia, Aislamiento, Durabilidad) brindan garantías de consistencia muy adecuadas en entornos críticos (por ejemplo, asegurando que una transferencia de municiones de un almacén a otro registre la salida y entrada de forma atómica, evitando estados inconsistentes donde el material desaparece de un almacén sin aparecer en el otro).

No obstante, con la creciente diversidad de datos, las bases relacionales han debido complementarse con enfoques alternativos para tipos de datos menos estructurados o para escalabilidad horizontal más allá de lo que suele soportar un solo nodo SQL tradicional.

2.5.2 Bases de datos NoSQL

El movimiento NoSQL surgió para atender requerimientos que las bases de datos relacionales convencionales no manejaban eficientemente, especialmente en aplicaciones web de gran escala que demandaban alta disponibilidad, particionamiento de datos masivos, y flexibilidad esquemática. En lugar de un modelo único, NoSQL agrupa una variedad de sistemas de almacenamiento, cada uno optimizado para ciertos patrones de uso. En logística, si bien el núcleo transaccional suele seguir en SQL, es posible aprovechar bases NoSQL para almacenar documentación, métricas de sensores o registros de actividad con mayor flexibilidad.

Entre las principales categorías de NoSQL destacan:

- **Almacenes de documentos:** En estos, cada entrada es esencialmente un documento autodescriptivo (generalmente en formato JSON, BSON o similar) que puede contener estructuras anidadas y *arrays*. MongoDB es un ejemplo popular de esta categoría. Son útiles cuando los datos que se almacenan varían en estructura de un caso a otro o evolucionan con el tiempo, ya que no requieren un esquema fijo predefinido. Cada documento puede tener campos diferentes, permitiendo gran flexibilidad. En logística militar, un almacén de documentos podría guardar, por ejemplo, los expedientes completos de inspección de un lote de munición, incluyendo campos estándar (fecha, inspector, resultado) y otros específicos de esa inspección particular (observaciones especiales, mediciones adicionales, fotografías embebidas), sin obligar a definir columnas para cada posible observación especial que podría surgir. Esta flexibilidad es particularmente valiosa cuando los procedimientos de inspección evolucionan o cuando diferentes tipos de munición requieren verificaciones distintas.

- **Almacenes clave-valor:** Representan el modelo más simple, almacenando pares clave (única) - valor (opaco para el sistema, puede ser texto, binario, JSON serializado). Son altamente escalables y rápidos para lecturas/escrituras directas por clave, aunque carecen de capacidades de consulta complejas. Sistemas como Redis, Amazon DynamoDB o Riak pertenecen a esta categoría. Pueden emplearse para cachés de respuestas recientes (por ejemplo, guardar temporalmente el resultado de una consulta compleja para evitar recalcularla), sesiones de usuario en aplicaciones web, o incluso para indexar rápidamente la disponibilidad de cierto material dado su identificador, sin la sobrecarga de una consulta SQL completa. En escenarios de alta concurrencia, estos sistemas pueden manejar millones de operaciones por segundo.
- **Bases de datos de familia de columnas (Wide-Column Stores):** Inspiradas en Bigtable de Google, almacenan datos en tablas esparcidas donde las columnas se agrupan en familias, y cada fila puede tener diferentes conjuntos de columnas. Apache Cassandra y HBase son ejemplos representativos. Proveen alta escalabilidad horizontal y rendimiento en escrituras masivas distribuidas, por lo que se han usado en sistemas de registro de eventos y aplicaciones que manejan series temporales. En un escenario naval, podrían aplicarse para registrar telemetrías continuas (por ejemplo, temperatura y humedad de varios polvorines a lo largo del tiempo, presión atmosférica, niveles de gases), donde cada nueva lectura se inserta como una columna adicional en una familia identificada por la ubicación y el sensor, permitiendo luego consultar eficientemente la serie temporal por emplazamiento o agregar estadísticas por períodos de tiempo. La naturaleza distribuida de estos sistemas permite escalar horizontalmente añadiendo nodos al clúster.
- **Bases de datos de grafos:** Optimizadas para datos altamente interconectados, donde los elementos (nodos) tienen muchas relaciones (aristas) entre sí, y donde las consultas frecuentemente necesitan atravesar estas relaciones. Neo4j, Amazon Neptune y ArangoDB son representantes conocidos. Emplean lenguajes de consulta especializados como Cypher o Gremlin que facilitan expresar recorridos complejos. En la logística de defensa, podrían modelar, por ejemplo, la red completa de proveedores, contratos, entregas y dependencias entre componentes, permitiendo consultas complejas como "¿qué proveedores alternativos están indirectamente conectados a este componente a través de la cadena de suministro?" o "¿cuál es el camino más corto entre dos entidades en la red logística?". Aunque este uso es menos típico en la gestión directa de municiones, podría ser relevante para analizar dependencias de suministro, detectar puntos únicos de fallo en la cadena logística, o mapear relaciones entre equipos y sus partes de repuesto.
- **Bases de datos de series temporales:** Especializadas en almacenar y consultar datos con *timestamp* asociado, optimizando específicamente para patrones de acceso donde las consultas se hacen por rangos de tiempo y las escrituras son predominantemente secuenciales. Ejemplos incluyen InfluxDB, TimescaleDB (basada en PostgreSQL), Prometheus y OpenTSDB. Estas bases suelen ofrecer compresión de datos por periodos (aprovechando patrones temporales), funciones agregadas eficientes en el rango temporal (promedios, máximos, mínimos, desviaciones estándar por día/semana/mes), y políticas de retención que automáticamente eliminan datos antiguos según criterios predefinidos. Pueden alertar automáticamente cuando métricas exceden umbrales definidos.

Cada uno de estos tipos NoSQL renuncia a alguna propiedad ACID o flexibilidad universal de consulta a cambio de eficiencia en casos específicos (el conocido teorema CAP y los *trade-offs* entre consistencia, disponibilidad y tolerancia a particiones). En el sistema global de una organización, suelen

coexistir con las bases relacionales en arquitecturas políglota: por ejemplo, se mantiene un sistema SQL central para datos maestros críticos (catálogo de material, registros de personal, datos contractuales), mientras se usa una base de documentos para almacenar documentos digitalizados procesados con OCR y anotados con NER, una base de series temporales para datos de sensores ambientales en tiempo real, y quizá un almacén clave-valor como capa de caché para acelerar consultas frecuentes. La clave está en integrar estos componentes de manera que las aplicaciones puedan extraer lo mejor de cada uno sin duplicar innecesariamente datos ni perder control de consistencia global, implementando patrones como *Event Sourcing* o CQRS cuando sea apropiado.

2.5.3 Bases de Datos Vectoriales

Las bases de datos vectoriales emergieron a finales de la década de 2010 como una categoría especializada, en respuesta a la proliferación de modelos de ML que generan *embeddings* de alta dimensión para representar datos (texto, imágenes, audio, video, etc.). A diferencia de las bases de datos tradicionales que recuperan registros por coincidencia exacta de valores o rangos numéricos simples, las bases vectoriales están optimizadas para búsquedas por similitud: dado un vector de consulta, deben encontrar rápidamente los vectores almacenados más cercanos según una métrica de distancia (coseno, Euclidiana, Manhattan, producto interno, etc.).

La hipótesis distribucional, articulada originalmente por lingüistas como Zellig Harris en 1954 [37] y John Rupert Firth en 1957 [38], establece el principio fundamental de que "conocerás una palabra por la compañía que tiene", sugiriendo que palabras apareciendo en contextos similares poseen significados similares. Los *embeddings* vectoriales formalizan matemáticamente esta hipótesis mediante el aprendizaje de representaciones vectoriales donde la similitud distribucional en el espacio original se traduce en proximidad geométrica en el espacio vectorial. Cuantitativamente, si $\text{sim}(w_i, w_j)$ representa alguna medida de similitud semántica entre palabras w_i y w_j en el espacio original, el objetivo es que la similitud coseno entre sus vectores correspondientes aproxime esta similitud, es decir, $\text{sim}(w_i, w_j) \approx \cos(v_i, v_j)$ donde $\cos$ se define como el producto punto normalizado de vectores.

Estas bases de datos son un pilar fundamental en sistemas de recuperación semántica y RAG [13], pues los documentos, frases o párrafos se almacenan como vectores, y consultas representadas como vectores permiten identificar información relevante, aunque no haya coincidencia textual literal. Este enfoque ha revolucionado la búsqueda de información al permitir recuperar documentos conceptualmente similares basándose en su significado semántico en lugar de solo en palabras clave coincidentes.

La evolución tecnológica de las bases vectoriales puede resumirse en tres generaciones:

- **Primera generación (2005–2015 aprox.):** Antes de la explosión del DL, las búsquedas semánticas se hacían con métodos de coocurrencia y factorización matricial como LSA (*Latent Semantic Analysis*) mencionado anteriormente, o modelos probabilísticos de tópicos como LDA (*Latent Dirichlet Allocation*). Herramientas específicas eran limitadas y experimentales; a veces se usaban extensiones de motores de búsqueda como Apache Solr con vectores densos de conteos, o incluso se intentaba usar extensiones espaciales como PostGIS de PostgreSQL para representar vectores como puntos en espacios de alta dimensión, con resultados subóptimos. No existían bases vectoriales dedicadas, sino aproximaciones ad-hoc que no escalaban bien. El rendimiento no era óptimo para alta dimensionalidad (*curse of dimensionality*), y las búsquedas exhaustivas eran computacionalmente prohibitivas para conjuntos de datos moderadamente grandes.

- **Segunda generación (2016–2020):** Con Word2Vec [18], doc2vec y luego BERT [9] generando *embeddings* contextuales de alta calidad, surgió la necesidad urgente de estructuras de índice especializadas. Algoritmos de *Approximate Nearest Neighbor Search* como Annoy (*Approximate Nearest Neighbors Oh Yeah*, desarrollado por Spotify), FAISS [35], y implementaciones de HNSW [39] se difundieron ampliamente entre investigadores de ML. Aun así, su integración era manual y requería conocimientos especializados: eran bibliotecas en memoria sin todas las características de una base de datos completa (transacciones, control de concurrencia, seguridad, replicación, respaldo automático). En esta época aparecen los primeros motores vectoriales dedicados experimentales y *plugins* para sistemas existentes, como Elasticsearch/OpenSearch integrando capacidades vectoriales, y el proyecto Milvus en sus primeras versiones.
- **Tercera generación (2021 en adelante):** Con la consolidación del uso industrial de vectores impulsada por el auge de los LLMs y aplicaciones de IA generativa, emergen soluciones completas y maduras tipo Pinecone, Weaviate, Milvus 2.0, Qdrant, Chroma y otros, diseñados desde cero para almacenar, indexar y consultar billones de vectores a escala distribuida en producción. Wang et al. (2021) presentaron Milvus en su artículo "*Milvus: A Purpose-Built Vector Data Management System*" [40], describiendo un sistema diseñado específicamente para gestionar eficientemente datos vectoriales a gran escala. Los autores demostraron que Milvus es hasta dos órdenes de magnitud más rápido que los competidores mientras proporciona más funcionalidades, soportando interfaces de aplicación fáciles de usar (SDKs en Python/Java/Go/C++ y APIs RESTful), optimizando para plataformas de cómputo heterogéneas con CPUs y GPUs modernas, habilitando procesamiento avanzado de consultas más allá de simple búsqueda de similitud vectorial, manejando datos dinámicos para actualizaciones rápidas mientras asegura procesamiento eficiente de consultas, y distribuyendo datos a través de múltiples nodos para lograr escalabilidad y disponibilidad. Estos sistemas incorporan no solo índices eficientes (grafos de vecinos como HNSW, árboles, *product quantization*, IVF (*Inverted File Index*), DiskANN (para datos en disco, etc.), sino también características empresariales esenciales como metadatos con filtrado vectorial (para filtrar búsquedas por categorías, fechas, o cualquier atributo), versionado de *embeddings* (permitiendo actualizar modelos sin perder histórico), control de acceso granular (RBAC), multicliente para aislar datos de diferentes usuarios o departamentos, y conectores con lenguajes de consulta tipo SQL, GraphQL o APIs REST para integrarse fácilmente en aplicaciones empresariales modernas.

Una base vectorial típica mantiene en memoria RAM (o en un almacenamiento optimizado como SSD con técnicas de memoria mapeada) estructuras de índice sofisticadas que reducen drásticamente el número de cálculos necesarios para encontrar vecinos cercanos. Dado que comparar directamente un vector de 768 dimensiones (tamaño típico de *embedding* BERT-base) con un millón de otros vectores requeriría 768 millones de operaciones de multiplicación y suma, lo cual sería computacionalmente costoso y lento, estos índices construyen particiones espaciales inteligentes, grafos de proximidad, o estructuras de cuantización que permiten descartar la mayoría de candidatos rápidamente mediante heurísticas y concentrarse en los más prometedores. Por ejemplo, HNSW construye un grafo jerárquico navegable donde la búsqueda comienza en capas superiores esparcidas y desciende progresivamente a capas más densas, logrando complejidad logarítmica.

En la gestión logística militar, una base vectorial permite, por ejemplo, almacenar todos los párrafos de manuales técnicos, procedimientos operativos y normativa regulatoria en un espacio vectorial semántico unificado. Cuando un técnico o analista formula una pregunta o busca un concepto específico (por ejemplo, "procedimiento de emergencia para munición dañada por agua"), se obtiene en milisegundos el conjunto de fragmentos más cercanos conceptualmente, incluso si los documentos

originales usan terminología diferente (como "protocolo de respuesta a munición mojada" o "directiva para material afectado por humedad"). Esto se integra, como vimos en la sección de RAG, en la etapa de recuperación de sistemas de generación aumentada. Otra aplicación valiosa es la detección de duplicados o versiones similares de un documento: documentos con vectores muy cercanos en el espacio semántico podrían indicar que, aunque tengan títulos distintos o provengan de fuentes diferentes, cubren materia redundante o casi idéntica, ayudando a consolidar y organizar la documentación institucional.

2.5.3.1 Diseño e indexación en bases vectoriales

El diseño de una base vectorial debe balancear cuidadosamente precisión versus rendimiento, y también considerar el uso de memoria versus capacidad de almacenamiento. A diferencia de las búsquedas tradicionales donde se espera recuperar resultados exactos, en búsquedas vectoriales generalmente se admite un resultado aproximado (no necesariamente los k vecinos verdaderamente más cercanos, pero sí prácticamente indistinguibles para el uso final y con alta probabilidad de solapamiento con los verdaderos k más relevantes) a cambio de velocidad y escalabilidad. Esta aproximación es aceptable en la mayoría de aplicaciones de IA porque los *embeddings* mismos son aproximaciones del significado, y pequeñas variaciones en el ranking de resultados muy similares raramente afectan la experiencia del usuario final.

Algoritmos como HNSW [39] construyen un grafo jerárquico de proximidad entre vectores durante la indexación: las consultas entonces navegan ese grafo mediante búsqueda *greedy* en múltiples capas, evitando recorrer exhaustivamente todos los puntos. HNSW logra complejidad de búsqueda logarítmica y ha demostrado excelente balance entre velocidad y *recall* en *benchmarks*. Otros enfoques como *Product Quantization* (PQ) reducen la dimensionalidad efectiva codificando el espacio vectorial en subespacios con libros de códigos preaprendidos (*codebooks*), de modo que las comparaciones se hacen en un espacio comprimido mucho más pequeño, sacrificando algo de precisión por enorme ahorro de memoria. IVF (*Inverted File Index*) particiona el espacio en regiones mediante *clustering* (típicamente *k-means*), y durante la búsqueda solo se exploran los *clusters* más cercanos al vector de la consulta. DiskANN permite indexar billones de vectores en disco SSD, usando grafos Vamana optimizados y técnicas de precarga para mantener buen rendimiento sin requerir toda la base en RAM.

Un desafío adicional importante es la actualización dinámica del índice: en entornos logísticos operacionales, se incorporan nuevos documentos constantemente (nuevas directivas, reportes diarios, actualizaciones de manuales), y ocasionalmente se eliminan o actualizan documentos existentes. Los índices deben permitir inserciones incrementales y, a veces, borrados eficientes sin tener que reconstruir por completo la estructura (lo que podría ser inviable a menudo, especialmente con billones de vectores). Las soluciones modernas emplean índices incrementales o estructuras mixtas (combinando un índice estático grande optimizado para búsqueda con uno pequeño mutable para cambios recientes, que se fusionan periódicamente mediante procesos de compactación), permitiendo continuar sirviendo consultas mientras se actualizan los datos. Sistemas como Milvus implementan arquitecturas de escritura diferida y separación de computación y almacenamiento para manejar eficientemente estas actualizaciones.

2.5.3.2 Implementaciones de Bases de Datos Vectoriales

Entre las implementaciones de bases vectoriales especializadas, cada una ofrece particularidades y está optimizada para diferentes casos de uso:

- FAISS (*Facebook AI Similarity Search*): es una biblioteca de propósito general desarrollada por Meta AI, implementada en C++ con envoltorios oficiales en Python, muy utilizada tanto para experimentación académica como en producción industrial. FAISS implementa numerosos algoritmos de indexación (*flat brute-force*, IVF con variantes, PQ, HNSW, y combinaciones híbridas) y está altamente optimizada para aprovechar SIMD y GPUs. Sin embargo, FAISS es esencialmente una biblioteca de indexación, no una base de datos completa, por lo que requiere

integrar su funcionalidad de búsqueda con sistemas de almacenamiento y gestión de datos propios.

- Milvus: un proyecto de código abierto que evolucionó rápidamente desde su lanzamiento en 2019, graduándose en la *LF AI & Data Foundation* en 2021. Soporta múltiples tipos de índices (IVF, HNSW, ANNOY, DiskANN, con cuantización) y se integra perfectamente con Docker y Kubernetes para despliegues en la nube. Permite almacenamiento persistente en disco de grandes volúmenes de vectores, consultas distribuidas a través de múltiples nodos, y ha sido diseñado con arquitectura nativa en la nube que separa computación de almacenamiento. En 2021 se demostró que Milvus supera a competidores por uno o dos órdenes de magnitud en rendimiento [40]. Milvus 2 incorpora características empresariales como múltiples niveles de consistencia, integración multinube, consola administrativa, y capacidades de consulta temporal para consultar datos históricos. Es ampliamente usado en producción por cientos de organizaciones en diversos campos.
- Pinecone: servicio gestionado completamente en la nube que provee una API sencilla y abstrae toda la complejidad de índices, fragmentación de datos, y replicación; asegura alta disponibilidad mediante gestión automatizada de infraestructura. Es popular por su simplicidad de adopción (literalmente minutos para empezar), pero en entornos sensibles como el militar, su uso dependería de las garantías de seguridad, cumplimiento normativo, y ubicación física de datos que pueda ofrecer. Algunos proveedores ofrecen despliegues *on-premise* dedicados o en nubes privadas para clientes gubernamentales.
- Elasticsearch / OpenSearch: originalmente motores de búsqueda full-text basados en Lucene, han incorporado soporte nativo para vectores densos desde versiones recientes. Esto es útil cuando se quiere combinar en la misma consulta filtros por texto tradicional (usando BM25 o análisis lingüístico) y similitud por vector (búsqueda semántica), permitiendo consultas híbridas sofisticadas. Sin embargo, su rendimiento en búsquedas vectoriales puras suele ser inferior al de motores diseñados específicamente para vectores, y la sobrecarga de mantener índices Lucene puede ser significativo cuando no se necesitan las capacidades de búsqueda de texto completo.
- Weaviate: un sistema de código abierto que destaca por su integración nativa con GraphQL para consultas, módulos de vectorización incorporados (puede llamar automáticamente a APIs de *embedding*), y capacidades de razonamiento sobre grafos de conocimiento combinados con vectores. Está optimizado para casos de uso que mezclan búsqueda semántica con relaciones estructuradas entre entidades.
- Qdrant: implementado en Rust, enfatiza eficiencia y seguridad de memoria, con soporte sofisticado para filtros condicionales (permitiendo pre-filtrar vectores por metadatos antes de la búsqueda de similitud, o post-filtrar después), indexación de datos asociados, y facilidad de despliegue. Popular en aplicaciones de producción por su rendimiento consistente.
- Vespa: desarrollado originalmente por Yahoo (ahora de código abierto), orientado a aplicaciones web de recomendación y búsqueda a gran escala, permite mezclar señales de ranking tradicionales (features manuales, BM25) con vectores, *machine learning ranking*, y lógica de negocio compleja en un sistema unificado. Usado en producción para servir recomendaciones a cientos de millones de usuarios.
- Chroma: enfocado en simplicidad y facilidad de uso para desarrolladores, especialmente en el ecosistema de aplicaciones LangChain y aplicaciones RAG. Perfecto para prototipado rápido y proyectos pequeños a medianos, aunque menos maduro para escala empresarial extrema.

El ecosistema se completa con herramientas y prácticas para gestión de ciclo de vida de *embeddings*: por ejemplo, mantener versiones de *embeddings* si se actualiza el modelo que los genera (para no mezclar vectores de BERT-base con los de un modelo más nuevo como *sentence-transformers* o *embeddings* multilingües, que viven en espacios vectoriales incompatibles), utilidades para detección y mitigación de sesgos en el espacio vectorial (importante en aplicaciones sensibles), y *frameworks* para evaluación continua de calidad de *retrieval* (midiendo *recall@k*, NDCG, etc. en conjuntos de validación).

3 ARQUITECTURA DE LA SOLUCIÓN

Los sistemas de gestión documental tradicionales suelen fallar cuando se enfrentan a operaciones multiformato, con restricciones de seguridad estrictas y con la necesidad de operar sobre datos que cambian en tiempo real. El reto aparece en la intersección de heterogeneidad documental, requisitos de seguridad y complejidad operativa.

Los fundamentos expuestos en el capítulo anterior —OCR, PLN con *transformers*, RAG, y almacenamiento híbrido con bases de datos relacionales y vectoriales— establecen la base teórica para el diseño de un sistema práctico de gestión documental.

3.1 Paradigma arquitectónico: monolito y microservicios

El apartado anterior ha presentado la visión general del sistema como un conjunto de componentes especializados que procesan, almacenan y consultan información de forma coordinada, garantizando modularidad, escalabilidad y resiliencia. No obstante, la materialización técnica de estos principios arquitectónicos requiere adoptar un paradigma de diseño que determine cómo se estructuran, despliegan e interconectan dichos componentes. Esta decisión requiere analizar las alternativas disponibles y evaluar de su idoneidad para el contexto de planteado.

La evolución de las arquitecturas de software ha experimentado múltiples etapas, ajustándose progresivamente a las crecientes demandas de escalabilidad, mantenibilidad y rapidez en los ciclos de desarrollo. En sus inicios, las aplicaciones se construían siguiendo el paradigma de arquitectura monolítica, en el cual todos los componentes —incluida la interfaz de usuario, la lógica del servidor y la gestión de bases de datos— se integraban en una única base de código cohesiva. Este enfoque, si bien facilita las fases iniciales del desarrollo debido a su simplicidad estructural, presenta limitaciones significativas a medida que el sistema incrementa su complejidad. En particular, incluso modificaciones menores pueden afectar amplias secciones del software, lo que dificulta las tareas de mantenimiento, ralentiza los procesos de actualización y convierte la corrección de errores en una actividad laboriosa y propensa a introducir nuevas incidencias [41].

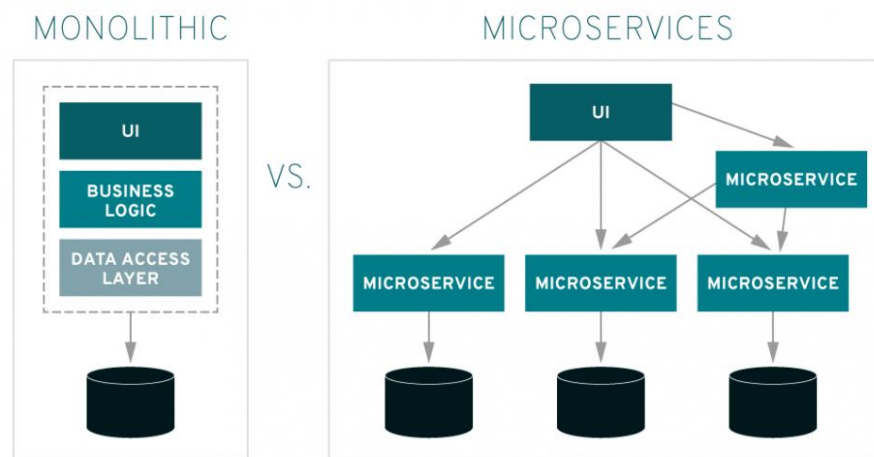


Figura 3-1. Representación comparativa de alto nivel entre arquitectura monolítica y microservicios [42].

Cuando la aplicación crece en funcionalidad y complejidad, las limitaciones del monolito se vuelven evidentes. Su alto acoplamiento dificulta evolucionar módulos de forma independiente, ya que cambios locales pueden provocar efectos colaterales y aumentar el riesgo de regresiones. Además, la escalabilidad obliga a replicar la aplicación completa, aunque solo un componente requiera más recursos (p. ej., OCR intensivo en CPU frente a consultas intensivas en BD), lo que genera ineficiencias. El ciclo de construcción y despliegue también se alarga con el tamaño del código, complicando la integración continua en equipos grandes. Frente a esto, la arquitectura de microservicios propone dividir el sistema en servicios pequeños y autónomos, permitiendo despliegue y escalado independientes y facilitando la evolución tecnológica por componente, como se aprecia en la Figura 3-1, donde se compara la estructura de un monolito con la descomposición en microservicios. [43].

Como evolución intermedia, los monolitos modulares ofrecen una opción que combina modularidad con menor complejidad operativa. En este enfoque la aplicación sigue siendo una única unidad desplegable, pero está organizada en módulos bien definidos que reducen las comunicaciones web entre componentes y, por tanto, el número de puntos de fallo distribuidos. Para un proyecto con requisitos de seguridad y latencia estrictos, un monolito modular puede simplificar la gestión operativa y disminuir la superficie de errores introducida por llamadas remotas.

Para superar las limitaciones propias de la arquitectura monolítica, en este trabajo fin de máster se adopta el modelo de arquitectura de microservicios. El sistema central de este trabajo sería un chatbot que ingiere documentación organizativa, responde tanto a preguntas logísticas como administrativas y técnicas y permite, bajo control, modificar una base de datos SQL a través de la interfaz conversacional. La arquitectura de microservicios facilita dividir la aplicación en servicios independientes —por ejemplo: ingestión y normalización de documentos, motor de procesamiento NLP/QA, servicio de gestión de la base de datos, y módulo de autenticación y auditoría—, de modo que cada componente pueda desarrollarse, desplegarse y escalarse de forma autónoma, mejorando la resiliencia y la mantenibilidad del sistema.

3.1.1 Elección arquitectónica

Para el diseño de un sistema de gestión documental organizativa basado en IA que debe operar en entornos de alta criticidad caracterizados por heterogeneidad documental, requisitos estrictos de seguridad y soberanía del dato, y complejidad operativa significativa, la arquitectura de microservicios ofrece ventajas estratégicas que justifican su adopción a pesar de la mayor complejidad operativa inicial. Esta decisión se fundamenta en cuatro pilares que alinean características arquitectónicas con requisitos del dominio.

3.1.1.1 Aislamiento funcional

La gestión de información sensible en entornos críticos requiere aplicar políticas de seguridad específicas a diferentes componentes del sistema según la naturaleza de las operaciones que realizan. La arquitectura de microservicios permite separar componentes con distintas características de seguridad, aplicando controles diferenciados que minimizan la superficie de ataque y facilitan la auditoría de módulos críticos [44].

Los servicios que gestionan ingesta de documentos pueden implementar controles estrictos de integridad verificando hashes criptográficos, escaneando contenido en busca de malware, validando formatos de archivo contra especificaciones conocidas, y rechazando documentos que no satisfagan criterios de admisibilidad. Estos servicios operan en la frontera del sistema, expuestos a contenido potencialmente malicioso o malformado, y su aislamiento en procesos dedicados previene que vulnerabilidades en el análisis de formatos complejos (PDF, Office, imágenes) comprometan otros componentes del sistema.

Los servicios de almacenamiento pueden aplicar cifrado transparente en reposo, políticas de retención diferenciadas según clasificación de seguridad de los documentos, y controles de acceso basados en atributos que verifican no solo la identidad del solicitante sino también el contexto de la solicitud (horario, ubicación de red, dispositivo utilizado). Al encapsular el acceso a datos sensibles detrás de APIs controladas, se garantiza que toda operación de lectura o escritura transite por puntos de validación centralizados que pueden auditar, registrar y potencialmente vetar accesos anómalos.

Los servicios de consulta pueden implementar controles de acceso granulares que filtran resultados según el perfil del usuario: diferentes roles organizativos acceden a subconjuntos específicos de información según su nivel de autorización y necesidad operativa, implementando el principio de mínimo privilegio de forma natural. Un analista técnico puede consultar especificaciones técnicas detalladas, pero no tiene visibilidad sobre información de adquisiciones; un oficial de operaciones accede a información de movimientos y distribución, pero no a detalles financieros. Esta segregación de información por servicio y por rol resulta más difícil de mantener en arquitecturas monolíticas donde todo el código comparte el mismo espacio de memoria y las fronteras de seguridad deben imponerse mediante disciplina de programación en lugar de aislamiento de procesos.

3.1.1.2 Escalabilidad dirigida

Los sistemas de gestión documental basados en IA presentan cargas de trabajo heterogéneas donde diferentes componentes experimentan patrones de demanda radicalmente distintos. La arquitectura de microservicios permite escalar horizontalmente solo aquellos componentes que experimentan picos de demanda, optimizando la utilización de recursos sin comprometer la estabilidad de servicios transaccionales [45].

Los módulos que ejecutan procesamiento de lenguaje natural, visión por computadora (OCR avanzado), o generación de *embeddings* mediante modelos de aprendizaje profundo son intensivos en recursos computacionales y pueden beneficiarse de aceleración por GPU. Durante operaciones de ingesta masiva de documentos históricos, estos servicios pueden escalar horizontalmente desplegando múltiples instancias en nodos con GPUs dedicadas, procesando lotes de documentos en paralelo sin bloquear otros componentes del sistema. Una vez completada la ingesta masiva, estas instancias pueden desescalarsse liberando recursos costosos.

En contraste, los servicios de consulta a bases de datos relacionales presentan patrones de carga más predecibles y constantes, correlacionados con el número de usuarios activos en horarios laborales, pero sin picos intensos de procesamiento. Estos servicios típicamente son intensivos en I/O (acceso a disco o red para recuperar datos) pero no requieren CPUs potentes ni GPUs. Separarlos de servicios intensivos en computación garantiza que tengan recursos garantizados incluso durante operaciones de procesamiento pesado en otros componentes.

Los servicios de inferencia con LLMs presentan requisitos singulares: alta latencia (varios segundos por solicitud), alto uso de memoria (modelos de decenas de GB en RAM/VRAM), y una tasa de procesamiento limitada por diseño. Desplegarlos en servicios dedicados permite configurar políticas de escalado apropiadas (por ejemplo, escalar basándose en longitud de cola de solicitudes pendientes en lugar de utilización instantánea de CPU), aplicar restricciones de tasa de solicitudes por usuario para prevenir monopolización de recursos, y gestionar versiones de modelo independientemente de otros componentes.

Esta escalabilidad selectiva resulta fundamental en contextos donde el presupuesto computacional es limitado (como en despliegues *on-premise* con hardware dedicado en lugar de elasticidad ilimitada de *cloud* público) y donde el costo de adquirir capacidad computacional adicional requiere justificación y planificación presupuestaria con meses de anticipación.

3.1.1.3 Despliegue y actualización ágiles

El campo de la IA evoluciona con velocidad vertiginosa: nuevos modelos de lenguaje, técnicas de *embeddings*, arquitecturas de recuperación de información, y *frameworks* de inferencia emergen con frecuencia mensual. Un sistema de gestión documental organizativa que desee mantenerse al día con el estado del arte requiere capacidad de actualización frecuente de componentes específicos sin interrumpir la operación del sistema completo ni forzar migraciones coordinadas masivas.

La arquitectura de microservicios permite redespigar únicamente el servicio afectado tras validar que la nueva versión es compatible con las interfaces existentes. Si se desea actualizar el modelo de *embeddings* para mejorar la calidad de búsqueda semántica (por ejemplo, migrando de BERT-base a un modelo multilingüe más reciente con mejor rendimiento en el idioma operativo del sistema), el proceso puede limitarse a: (1) reentrenar o ajustar finamente el nuevo modelo con datos del dominio; (2) regenerar *embeddings* del corpus documental existente usando el nuevo modelo en un proceso batch offline; (3) actualizar el servicio de búsqueda vectorial para usar el nuevo modelo en inferencia; (4) desplegar la nueva versión del servicio con pruebas A/B contra la versión anterior para validar mejoras. Durante todo este proceso, el resto del sistema continúa operando normalmente.

De forma similar, si se identifica que el módulo de OCR existente presenta limitaciones con ciertos tipos de formularios complejos, se puede evaluar, integrar y desplegar un motor de OCR alternativo actualizando únicamente el servicio de ingesta documental, sin requerir cambios en servicios de almacenamiento, consulta o generación que consumen el texto extraído.

Esta agilidad de despliegue resulta particularmente valiosa en contextos militares o gubernamentales donde los procesos de aprobación de cambios pueden ser prolongados, pero donde, una vez aprobada una modificación acotada, es crucial implementarla rápidamente sin exponer el sistema completo a riesgos de regresión asociados a despliegues monolíticos donde todo cambia simultáneamente.

3.1.1.4 Trazabilidad y auditoría y principio *human-in-the-loop*

Los requisitos de trazabilidad y auditoría en entornos regulados y críticos exigen que toda operación significativa quede documentada con información completa del contexto: quién solicitó la operación, cuándo, desde dónde, qué información se consultó o modificó, qué razonamiento guió las decisiones automatizadas, y qué validaciones se aplicaron antes de autorizar la operación. La arquitectura de microservicios facilita la implementación de estos controles al encapsular operaciones críticas en servicios dedicados que actúan como puntos de control centralizados.

Al centralizar las modificaciones en estos servicios, pueden aplicarse flujos de validación por etapas con registro de cada paso: recepción de la solicitud, verificación de permisos, normalización de parámetros, consulta del estado, comprobación de restricciones del dominio, propuesta de cambio, posible confirmación humana, ejecución tras aprobación y registro final inmutable. Esto permite reconstruir a posteriori la secuencia completa de decisiones que condujo a cualquier cambio.

Los servicios dedicados pueden implementar el patrón *human-in-the-loop* (HITL) de forma natural: operaciones críticas generan solicitudes de confirmación que se enrutan a colas de revisión humana donde supervisores autorizados pueden aprobar, rechazar o solicitar información adicional antes de que la operación se ejecute definitivamente. Ejemplos de operaciones que pueden requerir confirmación humana incluyen: eliminación de registros (requiere justificación y autorización de supervisor), modificaciones masivas de inventario (pueden indicar errores de análisis que deben validarse manualmente), operaciones que violan restricciones no estrictas (advertencias que no bloquean pero requieren justificación explícita), o acciones propuestas por modelos de IA cuya confianza está por debajo de un umbral (el sistema sugiere una acción pero solicita validación humana antes de ejecutarla).

Esta arquitectura facilita además la implementación de registros de auditoría inmutables mediante técnicas de *blockchain* interno o logs de escritura secuencial con firma criptográfica: cada operación registrada recibe un hash que depende del hash de la operación anterior, formando una cadena donde cualquier intento de modificación retroactiva sería detectable al romper la cadena de hashes.

3.1.2 Principios de diseño fundamentales

La propuesta se apoya en principios de diseño que aseguran coherencia y mantenibilidad del sistema distribuido; deben considerarse reglas de referencia y solo desviarse con justificación documentada.

3.1.2.1 Principio de responsabilidad única (Single Responsibility Principle)

Cada microservicio debe tener una responsabilidad bien definida y cohesiva, derivable de una única fuente de cambio en los requisitos del negocio. Un servicio de ingesta documental se responsabiliza únicamente de recibir documentos, validar su integridad y formato, extraer metadatos básicos, y preparar los documentos para su procesamiento posterior; no debe contener lógica de almacenamiento en bases de datos, generación de *embeddings*, o respuesta a consultas de usuarios. Esta separación facilita las pruebas (cada servicio puede probarse aisladamente con simulaciones de sus dependencias), el mantenimiento (los cambios en un servicio tienen impacto localizado), y la evolución independiente (diferentes equipos pueden iterar sobre diferentes servicios sin coordinación estrecha salvo en las interfaces compartidas).

La definición de "responsabilidad única" requiere juicio y debe evitar extremos: microservicios excesivamente granulares (nano-servicios) donde cada función trivial se convierte en un servicio independiente introducen sobrecarga de comunicación y complejidad operativa sin beneficio proporcional; servicios excesivamente amplios que agrupan múltiples responsabilidades solo porque comparten datos recaen en los problemas del monolito. El criterio heurístico recomendado es que un servicio debería ser mantenible por un equipo pequeño (2-8 personas) y su código base completa debería ser comprensible por un nuevo desarrollador en semanas, no meses.

3.1.2.2 Desacoplamiento mediante interfaces bien definidas

Los servicios deben comunicarse exclusivamente a través de interfaces públicas explícitamente diseñadas, preferiblemente mediante contratos formales que especifican esquemas de mensajes, semántica de operaciones, códigos de error, y garantías de compatibilidad. Esto permite que la implementación interna de un servicio evolucione (cambio de lenguaje de programación, reestructuración de estructuras de datos internas, optimización de algoritmos) sin afectar a los consumidores, siempre que se mantenga la compatibilidad de la interfaz pública.

El contrato de interfaz debe especificar no solo la sintaxis (tipos de datos, formato de serialización) sino también la semántica: qué garantías proporciona el servicio (idempotencia de operaciones, consistencia de lecturas tras escrituras, ordenamiento de resultados), qué pre-condiciones deben satisfacer las solicitudes (autenticación, autorización, validación de parámetros), y qué post-condiciones garantiza el servicio tras ejecución exitosa. Los contratos pueden expresarse mediante especificaciones formales (OpenAPI/Swagger para REST, Protocol Buffers para gRPC), documentación exhaustiva, o

idealmente suites de test de contrato que validan automáticamente que implementaciones de proveedores y consumidores permanecen compatibles a lo largo de evoluciones independientes.

El versionado de interfaces debe planificarse desde el diseño inicial: APIs públicas deben incluir información de versión (típicamente en URL para REST, en metadatos de mensaje para gRPC), y los servicios deben soportar múltiples versiones simultáneamente durante periodos de transición para permitir migraciones graduales de consumidores. La introducción de cambios incompatibles (modificaciones no retrocompatibles) debe ser extraordinaria y coordinarse mediante planes de deprecación con periodos de soporte prolongados de versiones antiguas.

3.1.2.3 Propiedad exclusiva de datos

Cada servicio debe ser el único responsable de su dominio de datos, con acceso exclusivo a su almacenamiento persistente. Si un servicio A necesita información gestionada por el servicio B, debe solicitarla mediante la API pública del servicio B en lugar de acceder directamente a la base de datos de B. Este principio garantiza la consistencia de los datos (todas las modificaciones transitan por la lógica de negocio del servicio propietario que puede aplicar validaciones y mantener invariantes), previene acoplamientos implícitos que dificultan la evolución del esquema de datos (cambios en la estructura interna de tablas de B no afectan a A si la API pública permanece estable), y facilita la migración tecnológica del almacenamiento (B puede migrar de SQL a NoSQL sin notificar a sus consumidores).

La aplicación estricta de este principio requiere replantear operaciones que tradicionalmente se resolverían mediante *joins* SQL entre tablas de diferentes dominios. En arquitectura de microservicios, estas operaciones se implementan mediante: (1) consultas secuenciales donde A solicita datos a B y luego combina localmente los resultados; (2) desnormalización controlada donde A mantiene una copia de lectura de subconjuntos de datos de B, actualizados mediante eventos publicados por B cuando los datos maestros cambian; (3) servicios de composición dedicados que orquestan llamadas a múltiples servicios y consolidan resultados.

Cada opción presenta *trade-offs*: consultas secuenciales aumentan latencia y acoplan la disponibilidad de A a la disponibilidad de B; desnormalización introduce consistencia eventual (la réplica puede quedar temporalmente desactualizada respecto al maestro) y duplicación de datos (mayor complejidad operativa). La elección depende de requisitos de latencia, consistencia y disponibilidad específicos de cada caso de uso, y debe documentarse explícitamente en la arquitectura.

3.1.2.4 Resiliencia y fallos

En sistemas distribuidos, los fallos son inevitables: servicios pueden estar temporalmente no disponibles debido a despliegues, actualizaciones, mantenimiento o errores; redes pueden experimentar latencias elevadas, pérdida de paquetes o particiones transitorias; recursos pueden agotarse temporalmente (memoria, conexiones a base de datos, descriptores de archivo). La arquitectura debe diseñarse asumiendo que los fallos ocurrirán regularmente, implementando patrones de resiliencia que permitan degradación controlada del servicio en lugar de fallos catastróficos en cascada donde la falla de un componente provoca efecto dominó colapsando el sistema completo.

Los patrones de resiliencia fundamentales incluyen: *timeouts* explícitos en todas las operaciones remotas (ninguna llamada debe bloquear indefinidamente esperando respuesta); reintentos con tiempos de espera exponencial y *jitter* (reintentar operaciones fallidas con intervalos crecientes y aleatorización para prevenir avalanchas simultáneas de solicitudes); *circuit breakers* que detectan cuando un servicio *downstream* está fallando sistemáticamente y abren el circuito temporalmente para prevenir desperdicio de recursos intentando operaciones condenadas al fracaso; compartimentos que aíslan pools de recursos (hilos, conexiones) de forma que fallos en un flujo no saturan recursos compartidos afectando otros flujos; *fallbacks* que proporcionan respuestas degradadas pero útiles cuando servicios primarios no están disponibles (por ejemplo, responder con información cacheada en lugar de información actualizada, o con respuesta genérica en lugar de personalizada).

La implementación de estos patrones requiere instrumentación sistemática del código para exponer métricas de salud de servicios, latencias de operaciones, tasas de error, y utilización de recursos, permitiendo a los *circuit breakers* y políticas de reintentos tomar decisiones informadas basadas en el estado observado del sistema en lugar de heurísticas estáticas que pueden ser demasiado agresivas (provocando falsos positivos que degradan innecesariamente el servicio) o demasiado conservadoras (permitiendo que fallos se propaguen antes de activar mecanismos de protección).

3.1.2.5 Observabilidad desde el diseño

Cada servicio debe instrumentarse desde su concepción para exponer métricas de rendimiento, trazas de ejecución y logs estructurados que permitan diagnosticar problemas en producción. En sistemas distribuidos donde una solicitud de usuario puede atravesar múltiples servicios (por ejemplo, una consulta compleja que requiere recuperación de información documental, consulta a base de datos relacional, ejecución de cálculos de validación, y síntesis de respuesta mediante LLM), la capacidad de correlacionar eventos y reconstruir el flujo completo de ejecución resulta fundamental para *debugging* y análisis de rendimiento.

La observabilidad requiere tres pilares complementarios: (1) Métricas agregadas que proporcionan visibilidad cuantitativa del comportamiento del sistema (latencias de operaciones, rendimiento efectivo medido en solicitudes por segundo, tasas de error por tipo, utilización de recursos computacionales); (2) Logs estructurados con contexto completo (*timestamp* preciso, identificador de servicio, identificador de solicitud, severidad, mensaje descriptivo, contexto estructurado en formato procesable); (3) Trazas distribuidas que correlacionan eventos a través de múltiples servicios mediante identificadores únicos propagados en encabezados de mensajes que permiten seguir una solicitud de usuario desde el punto de entrada hasta la respuesta final, visualizando qué servicios se invocaron, en qué orden, cuánto tiempo consumió cada servicio, y dónde se originaron errores si los hubo.

Los *frameworks* modernos de observabilidad como OpenTelemetry proporcionan estándares e instrumentación automática que facilita la implementación de estos pilares sin requerir excesivo código manual repetitivo. La inversión en observabilidad desde las fases tempranas del desarrollo paga dividendos exponenciales durante la operación del sistema, reduciendo el tiempo medio de detección de incidentes (MTTD) y el tiempo medio de resolución (MTTR) de forma que el sistema puede alcanzar niveles de disponibilidad elevados incluso cuando componentes individuales experimentan fallos transitorios frecuentes.

3.1.3 Patrones de comunicación entre microservicios

Un aspecto fundamental que define el comportamiento y el ciclo de vida de estos servicios es la modalidad de comunicación elegida, la cual impacta directamente en el acoplamiento y la tolerancia a fallos del sistema:

1. **Comunicación Síncrona (API REST):** se puede utilizar para interacciones en tiempo real, como la respuesta del asistente al usuario. Aunque facilita una integración directa, este enfoque aumenta la dependencia entre servicios: si el microservicio de inventario se satura, el servicio de chat que depende de él también sufrirá latencia o fallará (efecto de fallo en cascada), creando cuellos de botella operativos.
2. **Arquitectura Orientada a Eventos (*Event-Driven*):** Para mitigar los riesgos de la comunicación síncrona en procesos pesados (como la ingesta masiva de manuales), es posible implementar un paradigma asíncrono basado en eventos. En este modelo, un servicio no espera una respuesta inmediata, sino que envía un mensaje a un gestor de mensajes. Este componente enruta el evento hacia los servicios interesados (por ejemplo, el motor de OCR), que lo procesan de manera independiente.

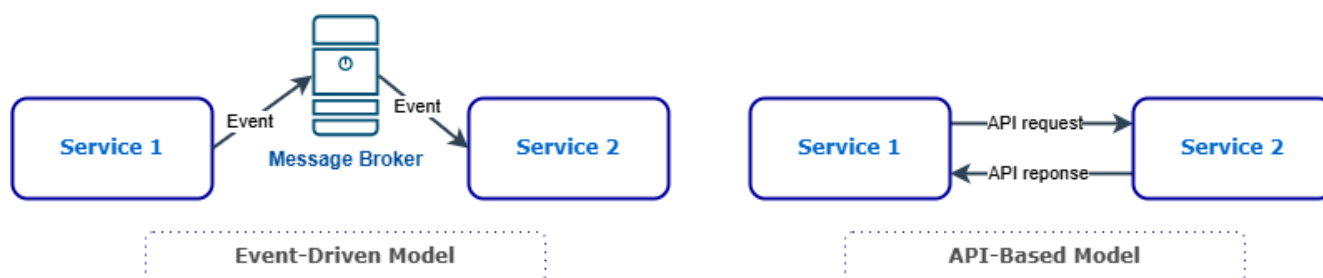


Figura 3-2. Representación de la comunicación síncrona y asíncrona (elaboración propia).

La adopción de este enfoque híbrido (REST + Eventos) permite que los microservicios sean verdaderamente autónomos, aumentando el paralelismo en el procesamiento y evitando bloqueos. Esto habilita un ciclo de vida de desarrollo desacoplado: cada servicio puede ser desarrollado en el lenguaje más adecuado para su función (Python para IA, Java para lógica transaccional), probado mediante tests unitarios aislados y desplegado independientemente sin detener el sistema completo. No obstante, esta modularidad exige una gestión rigurosa de la compatibilidad de versiones durante el despliegue para asegurar la interoperabilidad.

3.1.4 Escalabilidad

La gestión documental de municiones presenta cargas de trabajo asimétricas. La arquitectura de microservicios permite aplicar estrategias de **escalado horizontal** selectivo:

- **Escalado de Ingesta (OCR):** Durante la digitalización masiva de manuales históricos, el servicio de OCR y procesamiento de documentos sufre una carga intensiva de CPU/GPU. La arquitectura permite instanciar automáticamente múltiples réplicas de este servicio para paralelizar el trabajo, volviendo a reducirse a una única instancia cuando finaliza la carga, optimizando los recursos hardware *on-premise*.
- **Estabilidad Transaccional:** Por el contrario, los servicios de consulta de inventario (SQL) suelen tener una carga más estable y predecible. Mantenerlos separados evita que un pico de procesamiento de IA sature el servidor y ralentice las operaciones logísticas críticas.

3.2 Preprocesamiento

Etapas vistas en estado del arte

Etapas	Técnicas principales	Ventajas	Limitaciones
OCR	CNN, CRNN, Transformers	Alta precisión	Requiere GPU
NER	BiLSTM-CRF, BERT	Adaptabilidad	Coste de anotación
Tablas	CNN, GNN, atención	Comprensión estructural	Variabilidad de formatos
Normalización	Diccionarios, DL seq2seq	Corrección contextual	Datos incompletos

Tabla 3-1. Etapas para el preprocesamiento de documentación.

Aunque las etapas descritas se presentan de manera secuencial, en la práctica su aplicación depende del tipo de fuente documental y del grado de estructuración de los datos. El sistema debe ser capaz de adaptar dinámicamente el pipeline de extracción, omitiendo o iterando etapas según la naturaleza del documento y los requisitos operativos.

En el contexto de la gestión documental en entornos críticos, este enfoque modular resulta esencial para integrar fuentes heterogéneas —desde documentos escaneados hasta registros digitales— garantizando a la vez precisión, trazabilidad y eficiencia.

Tipo de documento	OCR	Extracción estructurada	Limpieza	Preprocesamiento	Validación
PDF nativo (texto digital)	No	Sí	Sí	Sí	Sí
Escaneo físico o imagen	Sí	Sí	Sí	Sí	Sí
Base de datos o CSV estructurado	No	Parcial (solo mapeo de campos)	Sí	Sí	Sí
Texto libre (informes, correos, mensajes, manuales)	No	Sí (NER, NLP)	Sí	Sí	Sí

Tabla 3-2. Fases de preprocesamiento para cada tipo de documento.

El OCR es la base para digitalizar documentos físicos o escaneados, convirtiéndolos en datos procesables por sistemas de IA. En 2025, los avances en OCR, impulsados por redes neuronales profundas, han alcanzado precisiones de hasta el 97% en documentos complejos, como facturas, contratos o registros técnicos [46].

Avances tecnológicos: Modelos como Easy OCR y enfoques híbridos que combinan OCR con visión por computadora (CV) y PLN han mejorado la capacidad para extraer texto de documentos no estructurados, incluyendo tablas, firmas y campos clave. Por ejemplo, Emburse utiliza OCR avanzado para procesar decenas de miles de documentos financieros con una precisión superior al 95% (Emburse Blog). En logística de defensa, el OCR es crucial para digitalizar órdenes de suministro, registros de inventario y documentos normativos, reduciendo errores humanos y acelerando procesos.

Herramientas: Tesseract [47], PaddleOCR [48], EasyOCR y motores comerciales desplegables *on-premise* (p. ej., ABBYY FineReader Engine), seleccionados según idioma, calidad del escaneo y requisitos de certificación.

3.3 Almacenamiento y bases de datos

El almacenamiento constituye uno de los pilares fundamentales de cualquier sistema de gestión documental basado en IA, especialmente en contextos de alta criticidad, donde convergen requisitos de seguridad, trazabilidad y rendimiento.

A diferencia de los sistemas convencionales, las arquitecturas modernas orientadas a IA requieren una aproximación híbrida. Esta dualidad responde a la naturaleza heterogénea de la información procesada:

- **Datos transaccionales estructurados:** Movimientos logísticos, inventarios de polvorines, registros de trazabilidad de lotes.
- **Contenido documental no estructurado:** Manuales técnicos, normativas OTAN y expedientes que han sido transformados en *embeddings* semánticos.

Esta arquitectura híbrida permite optimizar cada consulta según su naturaleza, integrando dos tipos de bases de datos que la IA unifica para una experiencia de usuario satisfactoria:

1. **Bases de Datos Relacionales (SQL):** Ofrecen garantías ACID, consistencia transaccional y atomicidad. Son imperativas para responder consultas exactas (p. ej: "*¿Cuál es el inventario actual del lote X?*").
2. **Bases de Datos Vectoriales:** Especializadas en espacios de alta dimensionalidad. Permiten recuperar fragmentos mediante métricas de distancia semántica (p. ej: "*¿Qué normativa aplica a municiones clase 1.1?*").

3.3.1 Estrategias de indexado en bases de datos vectoriales

Las bases de datos vectoriales gestionan colecciones masivas de vectores generados por modelos de *embeddings*. A diferencia de las búsquedas tradicionales, estas bases implementan algoritmos para calcular similitudes en espacios multidimensionales, algo fundamental en arquitecturas RAG.

El rendimiento depende críticamente de las estrategias de indexado:

- **Búsqueda exhaustiva (Fuerza Bruta/FLAT):** Compara el vector de consulta contra *todos* los almacenados. Garantiza exactitud máxima, pero tiene un coste computacional lineal, haciéndolo impracticable en grandes colecciones.
- **Índices basados en particionamiento (IVF - Inverted File Index):** Agrupan vectores mediante *clustering* (típicamente k-means). Restringen la búsqueda a los clústeres más cercanos, reduciendo notablemente las comparaciones necesarias.
- **Grafos Navegables Jerárquicos (HNSW - Hierarchical Navigable Small World):** Organizan vectores en capas. La búsqueda comienza en capas superiores (saltos largos) y desciende a capas inferiores para el refinamiento local. Ofrece un equilibrio excepcional entre velocidad y precisión.

Para aumentar la precisión es crucial el enriquecimiento mediante metadatos (clasificación de seguridad, vigencia, tipo de munición). Motores modernos como Milvus, Qdrant o Pinecone permiten aplicar filtros booleanos sobre estos metadatos antes o después de la búsqueda vectorial.

Finalmente, la tendencia consolidada es la fusión de recuperación semántica + búsqueda léxica (BM25) mediante técnicas de *Reranking*. Esto asegura capturar tanto el contexto conceptual como las coincidencias exactas de códigos alfanuméricos (NSN, números de lote).

3.3.2 Base de datos relacional

Paralelamente a la gestión vectorial, es imperativo mantener registros estructurados de alta integridad. Las bases de datos relacionales siguen siendo la norma para garantizar las propiedades **ACID** (Atomicidad, Consistencia, Aislamiento, Durabilidad) en operaciones sensibles como:

- Actualizaciones de inventario de munición.
- Autorizaciones de movimiento de material clasificado.
- Registros de auditoría forense.

El modelo relacional y el lenguaje SQL permiten implementar lógica de negocio compleja, como el cálculo de consumos proyectados o la detección de discrepancias de inventario.

La orquestación entre la capa relacional y vectorial debe ser cuidadosa:

1. **Ingesta:** Se inserta el documento en la BD relacional (obteniendo ID único, *timestamp* y *checksum*).
2. **Vectorización:** Se generan los *embeddings* y se asocian al ID relacional como metadato.
3. **Consulta:** El sistema puede filtrar primero por criterios estructurados (SQL) y luego realizar la búsqueda vectorial solo en el subconjunto de documentos aplicables.

Selección del Motor

- PostgreSQL: Destaca por su robustez, soporte de JSON y extensiones como pgvector.
- Oracle / SQL Server: Opciones comerciales con fuertes características de seguridad empresarial.

3.4 LLMs y RAG

Los LLMs representan un cambio de paradigma en la IA: hemos pasado del modelado estadístico simple a sistemas capaces de razonar, traducir y sintetizar conocimiento. Entrenados sobre billones de tokens, estos modelos exhiben capacidades emergentes que se aproximan a la competencia humana.

Sin embargo, en entornos de defensa y logística crítica, los LLMs "puros" presentan limitaciones inaceptables: falta de trazabilidad, conocimiento estático (congelado en su fecha de entrenamiento) y riesgo de alucinaciones.

La Solución propuesta es una arquitectura híbrida RAG. El paradigma de Generación Aumentada por Recuperación (RAG) emerge como la respuesta arquitectónica necesaria.

Esta aproximación garantiza tres pilares vitales para la gestión documental en los entornos de alta criticidad:

1. **Información dinámica:** El sistema responde con el último cambio de normativa sin reentrenar el modelo.
2. **Trazabilidad:** Cada respuesta cita el documento fuente (ej. *Reglamento de Pólvoras*, pág. 45).
3. **Veracidad:** Reduce drásticamente la generación de contenido inventado.

3.4.1 Elección de LLM

La base tecnológica de los LLMs es la arquitectura Transformer vista en la sección 2.3. Su innovación principal es el procesamiento de secuencias mediante mecanismos de atención que permiten modelar dependencias de largo alcance sin las limitaciones de las arquitecturas recurrentes previas.

La elección del modelo de lenguaje que actuará como núcleo generativo del sistema determina no solo sus capacidades funcionales, sino también los requisitos de infraestructura y las implicaciones de seguridad. En contextos civiles, esta decisión suele inclinarse hacia APIs comerciales por su conveniencia y rendimiento, pero en entornos militares las restricciones operativas reordenan completamente las prioridades.

El primer criterio es la soberanía del dato. Consultar a un LLM comercial mediante API implica que cada pregunta sobre inventarios, capacidades o procedimientos sensibles atraviesa infraestructura ajena. Aunque los proveedores aseguren que no almacenan las consultas, el mero tránsito de información clasificada por redes externas vulnera protocolos básicos de seguridad.

Otro aspecto relevante es el determinismo de las respuestas. Los LLMs modernos incorporan un parámetro de temperatura que controla su creatividad: valores altos generan respuestas variadas y originales, valores bajos producen salidas más predecibles. En aplicaciones creativas esto es deseable, pero en consultas sobre procedimientos reglamentarios la variabilidad es inaceptable. Si en un entorno crítico se consultan dos veces sobre el mismo artículo normativo y se reciben interpretaciones ligeramente distintas, el sistema no es fiable. La configuración del modelo debe garantizar que una misma pregunta produzca respuestas idénticas, eliminando cualquier componente aleatorio del proceso generativo.

Para el prototipo desarrollado en este trabajo se ha seleccionado Mistral 7B, ejecutado mediante Ollama. Esta elección responde directamente a los criterios previos: se ejecuta completamente en local sin conexión externa, muestra buen rendimiento en español, y permite configuración de temperatura cero para asegurar determinismo. Esta no es necesariamente la mejor opción técnica en términos absolutos, sino la más adecuada para un prototipo académico con restricciones de hardware. Mistral 7B puede ejecutarse en equipos convencionales con GPU de gama media, lo que permite desarrollar y validar el sistema completo sin infraestructura especializada.

La arquitectura propuesta asume explícitamente que ningún modelo de lenguaje, por potente que sea, posee conocimientos técnicos específico sobre municiones. Sus datos de entrenamiento son

genéricos y están congelados en el tiempo, lo que significa que desconocen actualizaciones normativas recientes, códigos de munición específicos o procedimientos particulares de la Armada. Intentar "enseñar" este conocimiento al modelo mediante reentrenamiento completo sería inviable: requeriría miles de ejemplos etiquetados por expertos, semanas de entrenamiento en clústeres de GPUs y un coste económico extremadamente alto.

Por ello, el sistema delega la precisión técnica en la arquitectura RAG que se detalla en el siguiente apartado. En lugar de confiar en que el modelo "recuerde" la normativa, el sistema la inyecta explícitamente en cada consulta. El modelo actúa como motor de razonamiento sobre información verificable recuperada en tiempo real, no como oráculo que responde desde su memoria paramétrica. Esta separación entre capacidad de razonamiento (LLM) y fuente de verdad (documentos recuperados) es la clave para mitigar alucinaciones en datos críticos como cantidades de explosivo o códigos de munición, donde un error no es una molestia sino un riesgo operativo real.

De cara a un despliegue operativo real, el sistema debería evolucionar hacia modelos más potentes. Si las restricciones de seguridad lo permitieran, opciones comerciales como GPT-4 ofrecerían capacidades superiores de síntesis y razonamiento complejo. Dado el requisito de seguridad, la ruta más probable sería migrar hacia modelos *open-source* de mayor tamaño como Llama 3 70B o Mixtral 8x7B, que requieren servidores con múltiples GPUs pero mantienen el control total sobre los datos.

Sin embargo, la mejora más eficiente en términos de coste-beneficio no sería escalar el LLM generativo, sino realizar un ajuste fino del modelo de *embeddings*. Entrenar el sistema de recuperación con un corpus de pares pregunta-documento reales de la Armada le enseñaría a entender jerga específica, acrónimos y la estructura de los documentos logísticos, asegurando que la información que llega al modelo generativo sea siempre la más pertinente.

3.4.2 Arquitectura de Generación Aumentada por Recuperación (RAG)

La Generación Aumentada por Recuperación (RAG) explicada en el apartado 2.4.3 puede dividir el procesamiento de la información en 3 fases:

1. **Ingesta y Estructuración** Transformación de documentos heterogéneos (PDFs, documentos Excel) en datos procesables.
 - **Chunking:** Los documentos se fragmentan en unidades semánticamente coherentes mediante estrategias de *chunking* que respetan límites naturales del texto como párrafos, secciones, o artículos normativos, evitando particiones arbitrarias que destruirían la coherencia contextual.
 - **Vectorización:** Cada fragmento se procesa mediante un modelo de *embeddings* que lo transforma en un vector denso de alta dimensionalidad que captura su significado semántico. Para el prototipo se emplea "mxbai-embed-large", cuya función es radicalmente distinta a la del LLM generativo: mientras Mistral sintetiza respuestas, el modelo de *embeddings* traduce texto a vectores numéricos que capturan su significado semántico.
 - **Almacenamiento:** Indexación de *embeddings* en bases de datos vectoriales.
2. **Recuperación** algoritmos que identifican fragmentos relevantes para una consulta específica con medidas de similaridad semántica en el espacio vectorial. Cuando el sistema recibe una consulta, esta se transforma en un vector con el mismo modelo de *embeddings* utilizado durante la ingesta, garantizando compatibilidad entre los espacios vectoriales de consultas y documentos.
 - **Búsqueda Semántica:** una búsqueda de k vecinos más cercanos en la base vectorial, identificando los fragmentos cuyas representaciones exhiben mayor similaridad con el vector de consulta según métricas como distancia coseno o producto interno.

- **Búsqueda Léxica (BM25):** Encuentra coincidencias exactas (superior para coincidir términos técnicos específicos, códigos alfanuméricos, o identificadores que no admiten variación semántica).
 - **Reranking:** Un modelo discriminativo reordena los resultados para asegurar la máxima relevancia.
 - **Filtros:** Antes de la generación, el sistema aplica filtros basados en metadatos que restringen el espacio de búsqueda según criterios contextuales, garantizando que solo fragmentos relevantes, vigentes y accesibles según el nivel de autorización del usuario se incluyan en el contexto proporcionado al modelo generativo. Este filtrado puede implementar lógica compleja que considera múltiples atributos simultáneamente, por ejemplo, restringiendo la búsqueda a documentos de un tipo específico, publicados dentro de un rango temporal determinado, aplicables a una categoría concreta de entidades operativas, y accesibles según el perfil de seguridad del usuario consultante.
3. **Síntesis Generativa** integra la consulta original con los fragmentos recuperados en un *prompt* estructurado que instruye al LLM a generar una respuesta fundamentada exclusivamente en la información proporcionada

La construcción de este *prompt* constituye un aspecto crítico del diseño, requiriendo balance cuidadoso entre proporcionar suficiente contexto para respuestas completas, mantener claridad en las instrucciones para evitar generaciones desalineadas, incluir mecanismos de validación que prevengan la utilización de conocimiento paramétrico del modelo cuando no existe evidencia en el contexto recuperado, y estructurar la información de forma que el modelo pueda referenciar fuentes específicas en sus generaciones

El resultado de esta fase es una respuesta que sintetiza información de múltiples fragmentos cuando es necesario, mantiene coherencia narrativa, y proporciona referencias explícitas a los documentos consultados, permitiendo que usuarios o auditores verifiquen independientemente la corrección de las afirmaciones realizadas.

3.4.3 Representaciones semánticas mediante *Embeddings*

Los *embeddings* son el motor de búsqueda del sistema. Transforman texto en vectores de alta dimensionalidad donde la proximidad geométrica equivale a similitud conceptual.

Actualmente los modelos de *embeddings* están contextualizados, por lo que el vector de una palabra cambia según su contexto (la palabra "batería" tiene vectores distintos en un contexto de artillería vs. un contexto de energía eléctrica).

Los modelos preentrenados de propósito general proporcionan representaciones semánticas competentes para lenguaje general, pero el rendimiento en dominios técnicos o especializados es inferior. El ajuste fino contrastivo: datos del dominio objetivo, donde se optimiza la función de pérdida para acercar representaciones de pares de texto semánticamente similares mientras se alejan representaciones de pares disímiles, permite adaptar el espacio de *embeddings* hacia las características específicas del corpus de aplicación que los modelos generalistas desconocen.

Por otra parte, los *embeddings* multilingües: Permiten buscar en documentación en diversos idiomas realizando consultas en español, unificando el conocimiento sin necesidad de traducción previa.

3.4.4 Ingeniería de *prompts* y control de generación

La ingeniería de *prompts* constituye la disciplina de diseñar entradas textuales que consigan los comportamientos deseados de los modelos de lenguaje, funcionando como una forma de programación usando lenguaje natural donde las instrucciones especifican qué tarea realizar, qué información considerar, qué restricciones respetar, y qué formato producir.

En arquitecturas RAG, el diseño del *prompt* resulta particularmente crítico porque debe integrar armónicamente múltiples componentes heterogéneos, incluyendo la consulta original del usuario, fragmentos documentales recuperados que pueden proceder de múltiples fuentes con diferentes estructuras y niveles de granularidad, datos estructurados extraídos de bases relacionales cuando corresponde, y resultados de cálculos deterministas cuando la consulta requiere validación de restricciones u optimización.

La estructuración efectiva de estos elementos en un *prompt* coherente que guíe al modelo hacia generaciones fundamentadas, precisas y verificables, requiere comprensión profunda tanto de las capacidades y limitaciones del modelo como de los requisitos específicos del dominio de aplicación

Los *prompts* efectivos típicamente incluyen varios componentes estructurados que cumplen funciones específicas en el control de la generación, por ejemplo:

1. Rol: *"Eres un oficial de logística experto en seguridad de municiones."*
2. Tarea: *"Responde a la consulta basándote exclusivamente en el contexto proporcionado."*
3. Grounding (Anclaje): *"Cita el documento y página de cada afirmación. Si la información no está en el contexto, indica que no se puede responder."*
4. Cadena de Pensamiento (CoT): *"Antes de responder, explica paso a paso tu razonamiento para verificar la compatibilidad de los materiales."*

Las estrategias de *prompting* de pocos ejemplos, donde se incluyen demostraciones de entradas y salidas esperadas antes de la tarea real, han demostrado efectividad notable para guiar el comportamiento del modelo hacia patrones deseados.

El *prompting* en cadena de pensamiento representa una técnica avanzada particularmente valiosa para consultas que requieren razonamiento multi-paso o integración de información procedente de múltiples fragmentos con diferentes niveles de abstracción. Esta aproximación instruye al modelo a generar explícitamente pasos intermedios de razonamiento antes de producir la respuesta final, exponiendo el proceso de inferencia que condujo a la conclusión.

En sistemas RAG, esto permite verificar que el modelo interpretó correctamente la información recuperada, identificó relaciones relevantes entre fragmentos distintos, aplicó apropiadamente reglas o restricciones documentadas en el contexto, y sintetizó coherentemente información potencialmente contradictoria o parcial.

La transparencia añadida por esta exposición del razonamiento facilita tanto la auditoría humana de decisiones del sistema como la identificación de errores en la lógica de inferencia que podrían indicar problemas en la recuperación de contexto o formulación del *prompt*.

3.4.5 Limitaciones y estrategias de mitigación

A pesar de las capacidades transformadoras de los modelos de lenguaje y las mejoras proporcionadas por arquitecturas RAG, estas tecnologías exhiben limitaciones fundamentales que deben reconocerse y mitigarse usando estrategias arquitectónicas apropiadas cuando se despliegan en entornos de alta criticidad. En la Tabla 3-3, se detallan los desafíos y sus salvaguardas técnicas:

Desafío / Riesgo	Descripción	Estrategia de Mitigación (Defensa)
Alucinaciones	Generación de datos falsos o inventados con alta confianza.	Grounding estricto: Instrucciones negativas ("No inventes") y verificación de citas cruzadas contra el documento fuente.
Opacidad (Caja Negra)	Dificultad para explicar por qué el modelo dio una respuesta.	Logging exhaustivo de <i>prompts</i> y contextos para auditoría.
Sesgos del Entrenamiento	Datos desactualizados o sesgos culturales del corpus original.	RAG: El conocimiento proviene de los documentos recuperados, no de la memoria del modelo. Ajuste fino ético.
Coste Computacional	Alto consumo de GPU y energía.	Cuantización: Reducción de precisión numérica (de 32 a 4 u 8 bits) manteniendo rendimiento. Uso de modelos híbridos (pequeños para rutina, grandes para complejo).
Ataques Adversariales	<i>Prompt Injection</i> para manipular el sistema o extraer secretos.	Sanitización de entradas: Validación rigurosa de <i>prompts</i> . Arquitectura de defensa en profundidad y monitoreo de patrones anómalos.

Tabla 3-3. Potenciales riesgos de los LLM y estrategias de mitigación

3.5 Proceso de Ingesta

El proceso de ingesta constituye el paso a través del cual información heterogénea procedente de múltiples fuentes se transforma en representaciones estructuradas, optimizadas tanto para recuperación eficiente como para procesamiento inteligente. Este proceso implementa un pipeline de transformaciones secuenciales que convierte documentos en su forma original —archivos digitales nativos, documentos escaneados requiriendo OCR o flujos de datos transaccionales— en fragmentos semánticamente coherentes, enriquecidos con metadatos descriptivos y representaciones vectoriales que capturan su significado conceptual.

El proceso de ingesta se estructura en seis fases conceptuales que transforman progresivamente datos sin procesar en conocimiento estructurado consultable:

3.5.1 Control de acceso y autenticación

Este componente garantiza que únicamente entidades autorizadas incorporen información al sistema, previniendo la introducción de datos falsos o la violación de compartimentación de seguridad.

- **Mecanismos de Autenticación:** se implementa autenticación contra directorios centralizados (LDAP/AD) reforzada con autenticación multifactorial (MFA) o biometría para operaciones sensibles.
- **Control de Acceso Basado en Roles (RBAC):** Se asignan permisos a roles abstractos. Por ejemplo, un rol de "*Oficial de Documentación*" puede ingestar manuales técnicos, mientras que un rol de "*Integrador Logístico*" solo puede volcar datos de stock, respetando el principio de privilegio mínimo.
- **Auditoría Forense:** Cada operación genera un rastro inmutable que incluye la identidad, *timestamp* y hashes criptográficos de los contenidos. Esto es vital para cumplir con normativas de defensa, donde la retención de logs puede ser obligatoria durante años para investigaciones posteriores.

3.5.2 Fragmentación semántica de documentos (*Chunking*)

La fragmentación o *chunking* divide documentos extensos en unidades optimizadas para la vectorización. Esta operación enfrenta una tensión fundamental: fragmentos muy pequeños pierden contexto; fragmentos muy grandes diluyen la precisión semántica al mezclar tópicos.

Existen tres estrategias principales aplicables al corpus documental:

1. **Longitud Fija:** Divide por número de tokens con solapamiento (*overlap*). Es simple y predecible, pero corre el riesgo de cortar oraciones o ideas a la mitad, ignorando la estructura semántica.
2. **Consciente de la Estructura (Structure-Aware):** Aprovecha los límites naturales del documento (títulos, secciones, artículos). En normativas militares, donde la jerarquía es estricta (Libro > Capítulo > Sección), esta estrategia es superior, ya que preserva el contexto jerárquico de cada fragmento.
3. **Fragmentación Semántica:** Utiliza NLP para detectar cambios de tópico en el texto y crear cortes dinámicos. Aunque computacionalmente más costosa, genera los fragmentos más coherentes conceptualmente.

Tratamiento de Tablas y Figuras: Elementos no textuales requieren atención especial. Las tablas de especificaciones de munición, por ejemplo, deben serializarse (JSON/YAML) o resumirse con LLMs antes de vectorizarse, para no perder la relación fila-columna que el texto plano destruiría.

3.5.3 Clasificación automática de contenido

La clasificación permite dirigir documentos hacia flujos de procesamiento especializados. Un sistema avanzado debe distinguir entre:

- **Documentos Narrativos:** Manuales de procedimiento (ruta de vectorización).
- **Datos Estructurados:** Hojas de carga o inventarios (ruta de extracción SQL).
- **Formularios:** Partes de incidencia (ruta de extracción de entidades).

Se emplean dos enfoques técnicos:

- **Clasificadores basados en LLM:** Utilizan *prompt engineering* sobre una muestra del documento para determinar su categoría. Son flexibles y se adaptan a nuevas taxonomías sin reentrenamiento.
- **Aprendizaje Supervisado:** Modelos entrenados (ej. BERT) sobre corpus etiquetados. Son más eficientes en inferencia, pero rígidos ante cambios en las categorías.

Además, la clasificación detecta y filtra contenido irrelevante o duplicado (mediante *hashing* perceptual), evitando que el "ruido" degrade la calidad del sistema RAG.

3.5.4 Generación de representaciones vectoriales

La transformación de texto en vectores densos (384 a 1536 dimensiones) es el núcleo del RAG. La calidad de los *embeddings* determina si el sistema encontrará la información correcta. Depende de los siguientes procesos.

- **Normalización:** Los vectores se escalan a longitud unitaria para permitir el uso de la *similitud coseno* (producto interno), optimizando el cálculo en procesadores modernos.
- **Procesamiento por Lotes (Batching):** Para ingestas masivas, se explota el paralelismo de las GPUs. Estrategias de *batching* inteligente agrupan textos de longitud similar para minimizar el desperdicio computacional.

- **Gestión de Longitud:** Para textos que exceden la ventana del modelo, se emplean estrategias de segmentación o *embeddings* agregados (promediado ponderado) para no perder información ubicada al final del fragmento.

Los modelos modernos basados en Transformers se pueden beneficiar del ajuste fino contrastivo (*Contrastive Fine-tuning*). Esto entrena al modelo para acercar vectores de textos semánticamente similares (ej. "proyectil 120mm" y "munición de carro de combate") y alejar los distintos, algo crucial en dominios técnicos con jerga específica.

3.5.5 Enriquecimiento con metadatos

Los metadatos son el puente entre la búsqueda semántica y las restricciones operativas del mundo real. Permiten aplicar filtros que la similitud vectorial por sí sola no puede resolver.

El sistema extrae y anexa a cada vector:

- **Metadatos de Origen:** Autor, fecha, clasificación de seguridad.
- **Entidades Nombradas (NER):** Identificación automática de calibres, unidades militares o zonas geográficas mencionadas en el texto.
- **Contexto Jerárquico:** Títulos de las secciones "padre", permitiendo al usuario saber si un fragmento pertenece a un anexo técnico o al cuerpo principal de una norma.

Versionado Temporal: En logística, saber qué norma estaba vigente en una fecha pasada es vital para auditorías. Los metadatos de "fecha de vigencia" y "timestamp de ingesta" permiten realizar consultas históricas ("¿Qué procedimiento aplicaba el 14 de febrero de 2023?").

3.5.6 Extracción estructurada de información

Ciertos documentos (albaranes, reportes estandarizados) contienen información que debe vivir en una base de datos relacional (SQL), no solo en un índice vectorial.

Este sub-proceso utiliza una arquitectura híbrida: combina análisis sintáctico y modelos de lenguaje para extraer campos específicos (Lote, Cantidad, NSN) y poblarlos en esquemas predefinidos (JSON/XML).

La orquestación de estos flujos asegura que un mismo documento pueda procesarse en paralelo: vectorizado para búsquedas semánticas y analizado para extracción de datos, sincronizando los resultados para disponibilidad unificada. La arquitectura debe además considerar aspectos operativos como idempotencia (evitar duplicados al reprocesar), recuperación ante fallos, y versionado para mantener historial de transformaciones auditable.

3.6 Proceso de Recuperación

La arquitectura de recuperación y generación constituye el componente cognitivo del sistema, responsable de transformar consultas de usuarios en respuestas fundamentadas que integran conocimiento de múltiples fuentes heterogéneas. A diferencia de arquitecturas monolíticas donde un único modelo de lenguaje intenta resolver todas las tareas, el sistema adopta un paradigma de agentes especializados coordinados por un componente orquestador central. Este enfoque multi-agente permite que cada agente desarrolle capacidades profundas en su dominio particular, mientras que la coordinación centralizada garantiza coherencia semántica y trazabilidad operativa en el flujo completo de procesamiento.

El sistema se estructura en cinco componentes arquitectónicos diferenciados que colaboran para transformar consultas en respuestas verificables:

3.6.1 Orquestador central (LLM Router)

El sistema utiliza un LLM como orquestador central que actúa como punto de entrada para todas las consultas. Este componente analiza cada petición del usuario y decide qué herramientas o agentes debe activar para resolverla.

Usando técnicas de *prompt engineering* estructurado, el orquestador transforma consultas en planes de acción descompuestos en subtareas, especificando qué agentes deben intervenir, en qué orden, considerando dependencias entre subtareas, y definiendo cómo integrar resultados parciales en la respuesta final.

Según la naturaleza de la consulta, el orquestador dirige las subtareas a los agentes apropiados siguiendo un esquema de enrutamiento basado en clasificación semántica. Las consultas que requieren información contextual documental se derivan al agente de recuperación semántica, que opera sobre la base vectorial de conocimiento. Las consultas que necesitan acceso a datos estructurados operativos se enrutan al agente de consultas sobre la base relacional. Las operaciones que demandan cálculos numéricos precisos, validaciones de restricciones complejas, u optimización combinatoria se delegan a módulos deterministas especializados. Finalmente, las consultas abiertas que requieren síntesis de información heterogénea se procesan por el agente generativo con anclaje contextual.

El orquestador además garantiza coherencia entre fuentes consultadas, control de flujo ante situaciones excepcionales, y trazabilidad operativa completa mediante registro detallado de cada decisión tomada.

3.6.2 Agente de recuperación semántica

Este agente es responsable de recuperación contextual de conocimiento documental previamente vectorizado e indexado. Opera sobre una base de datos vectorial que almacena representaciones semánticas densas de documentación normativa, procedimientos operativos, especificaciones técnicas, y cualquier otro corpus de conocimiento relevante. El agente ejecuta un proceso que incluye tres fases principales:

1. Formulación de la consulta semántica transformando la petición del usuario en vector con el mismo modelo de *embeddings* utilizado durante indexado.
2. Recuperación de fragmentos candidatos mediante búsqueda de similaridad en el espacio vectorial usando métricas como distancia coseno, y aplicación de filtrado y consolidación del contexto donde metadatos asociados a cada fragmento implementan restricciones contextuales que garanticen que solo información válida, vigente y accesible según el nivel de autorización del usuario sea incluida en el contexto retornado.
3. Los fragmentos recuperados se acompañan de metadatos completos que identifican documento origen, sección específica, fecha de vigencia, y cualquier otra información necesaria para establecer trazabilidad completa desde la respuesta final hasta las fuentes primarias consultadas.

3.6.3 Agente de consultas estructuradas

Este agente gestiona la recuperación de información desde bases de datos relacionales que almacenan información estructurada actual del sistema. Traduce necesidades de información expresadas en lenguaje natural a consultas SQL sintácticamente correctas y semánticamente apropiadas.

El proceso implementa múltiples capas de validación: análisis de intención de la consulta identificando entidades y relaciones requeridas, generación de SQL, validación de autorización verificando que el usuario tiene permisos para acceder a las tablas y vistas involucradas, sanitización de entrada previniendo inyección SQL mediante parametrización de consultas, y ejecución con *timeout* y límites de resultados para prevenir consultas que saturan recursos.

Los resultados se retornan en formato estructurado que el orquestador puede combinar con contexto documental recuperado por otros agentes.

3.6.4 Módulos Deterministas de Validación y Optimización

Los modelos de lenguaje, aunque exhiben capacidades notables de comprensión contextual y generación, presentan limitaciones fundamentales en razonamiento matemático exacto, validación de restricciones complejas, y optimización combinatoria.

En entornos críticos como la Defensa, siempre que se vayan a realizar consultas de este tipo, es necesario que la arquitectura incorpore módulos deterministas especializados que implementan algoritmos precisos para estas tareas críticas.

Estos módulos operan a través de interfaces bien definidas que aceptan parámetros estructurados y retornan resultados verificables, integrándose armónicamente con componentes basados en modelos de lenguaje.

El flujo típico involucra colaboración: el modelo de lenguaje interpreta la consulta, extrae parámetros relevantes, identifica restricciones aplicables mediante recuperación de documentación normativa, y formula el problema en términos de esquema de optimización o validación estructurado; el módulo determinista recibe esta formulación, ejecuta algoritmos apropiados garantizando corrección matemática usando métodos formalmente verificables, y retorna solución acompañada de certificados de optimalidad o factibilidad cuando corresponda; finalmente, el modelo de lenguaje traduce el resultado estructurado nuevamente a lenguaje natural, produciendo explicación comprensible que documenta qué restricciones se consideraron, qué cálculos se realizaron, y por qué la solución retornada satisface los requisitos especificados.

3.6.5 Generador con Anclaje Contextual

El agente generador representa el componente cognitivo final, responsable de sintetizar información heterogénea procedente de múltiples fuentes en respuestas coherentes, fundamentadas y explicables. Opera bajo un principio fundamental de *grounding* estricto: el anclaje de toda generación de texto en evidencia verificable procedente de fuentes autorizadas.

El proceso de generación opera mediante plantillas de *prompt* cuidadosamente diseñadas que integran múltiples tipos de contexto de forma estructurada: contexto documental recuperado mediante búsqueda semántica, datos estructurados obtenidos de la base relacional, resultados de módulos deterministas cuando la consulta requirió validación o cálculos, y la consulta original del usuario formulada explícitamente.

Las instrucciones de *grounding* constituyen el elemento más crítico del diseño del *prompt*. Estas reglas obligan a que toda afirmación esté respaldada por el contexto, que las cifras provengan únicamente de datos/cálculos, que las citas normativas incluyan referencia exacta a documento y sección, y que ante falta de información el sistema lo declare sin especular. También contempla casos excepcionales: contradicciones entre fuentes, datos incompletos o peticiones de información clasificada no accesible según autorización.

3.7 Flujo de procesamiento

3.7.1 Consulta

El procesamiento de consultas sigue un patrón general que puede adaptarse según complejidad específica de cada pregunta:

1. **Recepción y Análisis:** El orquestador recibe la consulta y analiza su estructura semántica, identificando qué tipos de información se requieren (normativa, datos operativos, cálculos) y qué agentes deben intervenir.

2. **Planificación:** Se genera un plan de ejecución especificando orden de invocaciones, manejo de dependencias entre subtarefas, y estrategia de consolidación de resultados.
3. **Recuperación Paralela:** Los agentes de recuperación semántica y consultas estructuradas operan potencialmente en paralelo, cada uno consultando sus fuentes especializadas bajo supervisión del orquestador.
4. **Validación Determinista:** Si la consulta involucra validación de restricciones o cálculos de optimización, los resultados de recuperación se pasan a módulos deterministas que producen respuestas verificables.
5. **Síntesis Fundamentada:** El agente generador recibe todo el contexto recopilado y produce una respuesta que sintetiza información manteniendo trazabilidad explícita hacia fuentes.
6. **Registro de Trazabilidad:** El resultado se almacena conjuntamente con el contexto completo que lo generó, incluyendo consulta original, todos los fragmentos documentales recuperados, consultas y resultados de base de datos, cálculos deterministas ejecutados, y *prompt* completo utilizado. Este registro exhaustivo permite auditorías posteriores que verifiquen corrección del proceso de razonamiento.

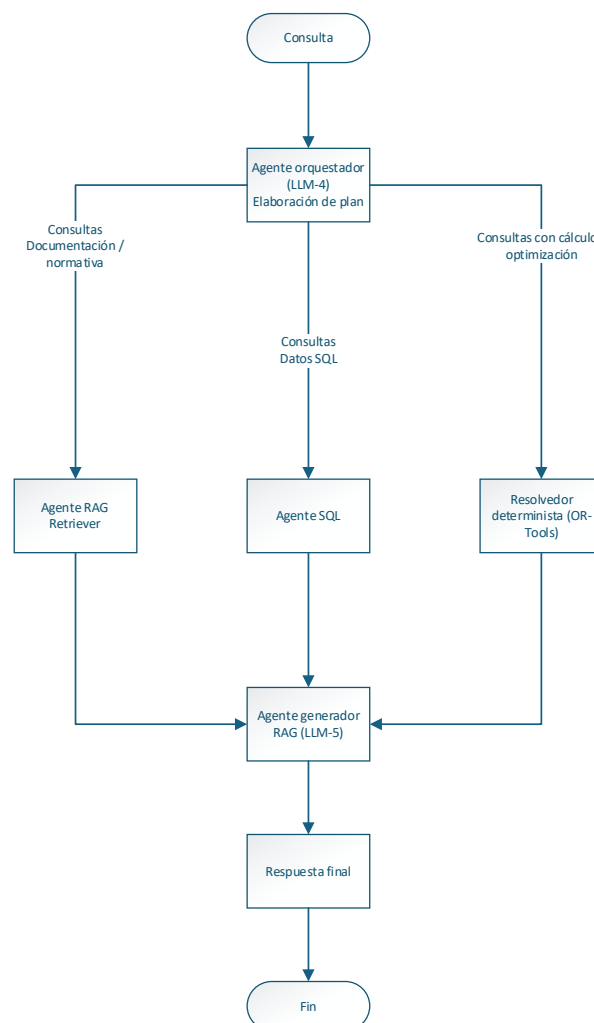


Figura 3-3. Diagrama de flujo de consultas (elaboración propia).

Todas las acciones, especialmente las de escritura, son validadas, registradas en un log inmutable y firmadas digitalmente.

3.7.2 Ingesta de documentos

Un usuario autorizado (rol técnico *writer* o *admin*, adaptable a roles operativos) sube un documento (manual, informe, mensaje, albarán, orden de entrega) en formato PDF o imagen. La ingesta registra metadatos, calcula un hash de integridad y aplica controles de autorización antes de cualquier procesamiento [49], [50].

El procesamiento de documentos sigue un pipeline secuencial con bifurcaciones condicionales según el tipo de contenido detectado. El flujo completo puede representarse en las siguientes etapas operativas:

1. **Recepción y Autenticación:** Un documento entra al sistema a través de interfaces de carga (web UI, API, integración con sistemas externos). El sistema autentica al usuario remitente, verifica sus permisos de ingesta, y registra la operación con *timestamp*. Si la autenticación falla, el documento se rechaza antes de cualquier procesamiento. Si el usuario carece de permisos para el nivel de clasificación declarado del documento, se genera alerta al administrador de seguridad.
2. **Detección de Formato y Preprocesamiento:** El sistema identifica el formato del documento (PDF nativo, imagen escaneada, documento Office, archivo de texto plano) y aplica el preprocesamiento apropiado. Documentos escaneados pasan por OCR con corrección y mejora de contraste. Documentos nativos digitales se extraen directamente. Formatos estructurados como hojas de cálculo requieren serialización que preserve relaciones tabulares. El resultado de esta fase es texto plano enriquecido con información de estructura (títulos, secciones, tablas) extraída del formato original.
3. **Clasificación y Enrutamiento:** El documento preprocesado se clasifica automáticamente en una de las categorías predefinidas del sistema: documentación normativa (requiere fragmentación semántica y vectorización), datos transaccionales (requiere extracción estructurada a base de datos relacional), documentos mixtos (requieren ambos flujos de procesamiento), o documentos fuera de alcance (se archivan sin procesamiento adicional, pero se registra su existencia). Esta clasificación determina qué ramas del pipeline se activarán y en qué orden.
4. **Enriquecimiento Contextual:** Los fragmentos y datos extraídos se enriquecen con información contextual que facilita filtrado y recuperación posterior. Los metadatos incluyen información de proveniencia (documento origen, sección específica, página), temporal (fecha de creación, fecha de vigencia, historial de versiones), seguridad (clasificación, permisos de acceso, compartimentación), y semántica (entidades nombradas identificadas, términos técnicos detectados, referencias cruzadas a otros documentos). Este enriquecimiento transforma fragmentos aislados en nodos de un grafo de conocimiento conectado.
5. **Validación *Human-in-the-Loop*:** Para datos críticos que alimentarán decisiones operativas, el sistema presenta resultados de extracción automática a operadores humanos usando interfaces de validación. Los operadores confirman, corrigen o rechazan datos extraídos, proporcionando *feedback* explícito. Las correcciones se almacenan como pares (extracción automática, corrección humana) que posteriormente alimentan procesos de reentrenamiento y mejora continua de los modelos. Documentos validados reciben marca de aprobación que autoriza su uso en producción.
6. **Indexación y Persistencia:** Los fragmentos validados se indexan en la base de datos vectorial utilizando estructuras optimizadas (HNSW, IVF) que aceleran búsquedas posteriores. Simultáneamente, datos estructurados se insertan en la base relacional respetando restricciones de integridad referencial. Si alguna inserción falla, se ejecuta *rollback* completo para mantener consistencia.

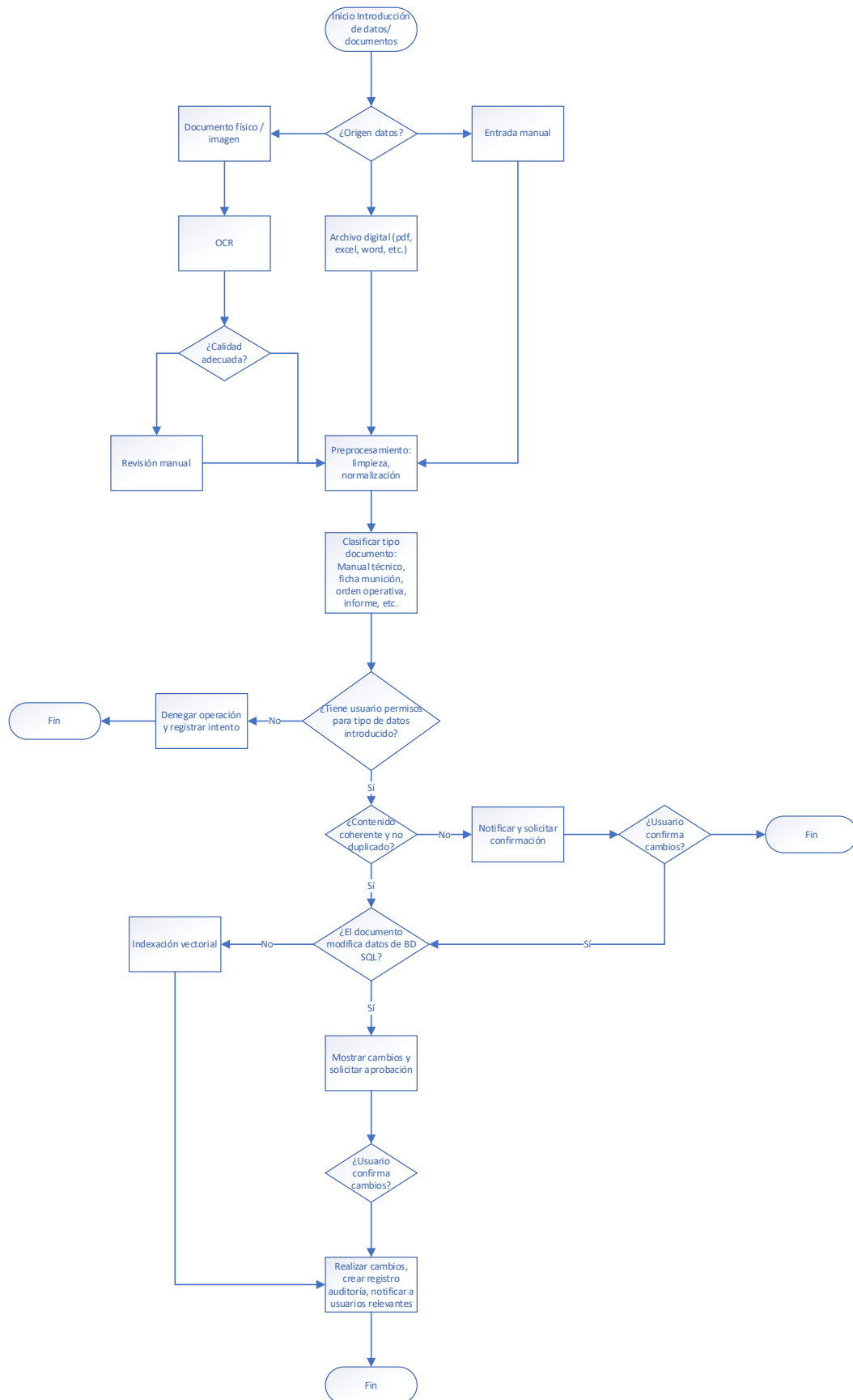


Figura 3-4. Diagrama de flujo de inserción de documentos (elaboración propia).

3.8 Seguridad de LLMs, y sistemas RAG

La incorporación de LLMs y arquitecturas RAG en entornos empresariales —y más aún en contextos críticos como el militar— presenta uno de los desafíos más complejos en materia de seguridad de IA. A diferencia de los sistemas tradicionales de software, donde los vectores de ataque suelen estar bien delimitados (*SQL injection*, desbordamiento de búfer, etc.), en sistemas con LLM surgen vectores de ataque completamente nuevos: desde la manipulación maliciosa de *prompts* hasta el envenenamiento de los datos de entrenamiento o de las bases de conocimiento vectoriales. Por tanto, el panorama de amenazas requiere marcos de seguridad innovadores y una estricta adherencia a regulaciones emergentes.

El panorama regulatorio europeo, liderado por el Reglamento de IA de la UE [51] y las directrices de la Agencia de Ciberseguridad de la UE (ENISA), converge con marcos estadounidenses como el NIST AI Risk Management Framework y directrices del Departamento de Defensa de EE. UU., estableciendo requisitos sin precedentes sobre transparencia, trazabilidad y ciberseguridad en sistemas de IA. Esta convergencia normativa, sumada a incidentes de alto perfil (por ejemplo, la filtración de información sensible a través de ChatGPT en Samsung en 2023, o el ataque "ConfusedPilot" contra Microsoft 365 Copilot en 2024 [52], subraya la urgencia de implantar controles de seguridad robustos desde el diseño inicial de estos sistemas. Las organizaciones deben tomar decisiones arquitectónicas fundamentales, como optar por despliegues *on-premise* versus en la nube, establecer estrategias de *backup* para componentes distribuidos, y desarrollar mecanismos de control que equilibren funcionalidad con seguridad en un ecosistema tecnológico que evoluciona diariamente.

A continuación, se abordan los principales vectores de amenaza y las contramedidas recomendadas para sistemas basados en LLM y RAG, así como consideraciones de despliegue y cumplimiento normativo.

3.8.1 Superficie de amenazas y vulnerabilidades específicas de LLM

Los LLM introducen superficies de ataque que no existían en aplicaciones tradicionales [53]. La investigación académica reciente documenta una rápida proliferación de estudios sobre vulnerabilidades en estos modelos: conferencias de primer nivel (NeurIPS, Black Hat, DEF CON) han presentado numerosas técnicas de explotación, mientras que conjuntos de datos públicos de *prompts* adversarios (como 15.000 *jailbreaks* recopilados en 2024) permiten probar y refinar ataques de forma sistemática [54]. Esta explosión investigadora refleja tanto la novedad del campo como la velocidad a la que los atacantes descubren brechas.

Entre las vulnerabilidades más críticas destaca la inyección de *prompts*. Esta técnica permite a un atacante manipular el comportamiento del LLM mediante entradas cuidadosamente diseñadas, ya sea de forma directa (instrucciones maliciosas por parte del usuario) o indirecta (ocultando texto malicioso en documentos que el sistema RAG recupere). No existe aún una solución perfecta para este problema. El ataque HouYi [55] demostró tasas de éxito en la evasión de restricciones en 31 de 36 intentos contra distintos modelos comerciales, mientras que la técnica Deceptive Delight de Palo Alto Networks logró un 65% de éxito atravesando filtros de ocho modelos en octubre de 2024. Más preocupante aún, la investigación de Anthropic sobre "*many-shot jailbreaking*" [56] reveló que precisamente una de las ventajas alardeadas de algunos LLM —las ventanas de contexto largas— puede volverse en contra, ya que permiten a un atacante insertar cientos de ejemplos sutiles que gradualmente erosionan las salvaguardas del modelo.

Otra amenaza inquietante es el envenenamiento de datos, tanto del corpus de entrenamiento como de la base de conocimiento en un sistema RAG. Contrariamente a lo que podría pensarse, modelos más grandes no son necesariamente más resilientes: un estudio de Anthropic (2024) mostró que apenas 250 documentos maliciosos insertados en el corpus de entrenamiento pueden inducir comportamientos erróneos en modelos de cualquier tamaño. En aplicaciones concretas, el ataque "ConfusedPilot" [52] ejemplificó cómo documentos maliciosamente alterados almacenados en la plataforma SharePoint de

Microsoft 365 Copilot comprometieron la integridad y confidencialidad de las respuestas generadas por el sistema. Incidentes reales han validado estos riesgos teóricos: la exposición accidental de tokens de usuarios en integraciones con Hugging Face, fugas de datos confidenciales en funciones de IA de Slack, y accesos indebidos a historiales de ChatGPT, ilustran que los entornos de producción ya están lidiando con estos problemas.

La extracción de modelos (*model stealing*) y los ataques de inferencia inversionista son otro frente preocupante. Mediante técnicas de aprendizaje activo y consultas adaptativas, se ha demostrado que es posible reconstruir, al menos parcialmente, modelos propietarios costosos interrogándolos como si fueran cajas negras. Investigaciones recientes sobre LLaMA 3 [57] demostraron la extracción exitosa de información de identificación personal que supuestamente el modelo había olvidado, mientras que ataques por aprendizaje dividido (*splitting*) lograron tasas de éxito del 38–75% en extracción de parámetros en pruebas de enero de 2025. Como defensa emergente destaca la privacidad diferencial durante el entrenamiento (DP-SGD), que añade ruido controlado para proteger datos individuales en el corpus, aunque conlleva sacrificios en rendimiento que deben sopesarse.

Los sistemas RAG, al combinar múltiples componentes (bases de datos vectoriales, motores de búsqueda, LLM), introducen vectores de ataque adicionales. La seguridad de un pipeline RAG requiere proteger cada fase: en la recuperación, asegurar que las consultas no revelen información sensible y que los índices no sean explotables; en la capa vectorial, evitar accesos no autorizados o manipulaciones; y en la generación, monitorear que el LLM no exponga contenido que debería permanecer confidencial.

El marco OWASP actualizado en 2025 [58] incluye una categoría específica (LLM08) para "Debilidades en Vectores y *Embeddings*", reconociendo ataques emergentes como la inferencia inversa de *embeddings* (donde se logra reconstruir parcialmente el texto original o información sensible a partir del vector almacenado). La investigación reciente propone mitigaciones avanzadas: por ejemplo, el marco FRAG (*Federated Retrieval-Augmented Generation*) [59] sugiere implementar sistemas RAG federados con garantías de privacidad, de modo que los datos sensibles nunca residan completamente en un solo nodo. También se exploran enfoques criptográficos, como el cifrado homomórfico (ej., herramientas de IronCore Labs), que permitirían realizar comparaciones de similitud vectorial sobre datos cifrados, evitando exponer contenidos en texto claro ni siquiera en memoria.

En síntesis, asegurar sistemas basados en LLM y RAG requiere una visión holística: adoptar medidas de ciberseguridad clásicas (seguridad perimetral, autenticación fuerte, auditoría de accesos) y complementarlas con nuevas técnicas específicas (filtrado de *prompts*, monitoreo de salidas anómalas, robustez en *embeddings*). Asimismo, involucra un proceso continuo de actualización, ya que tanto las amenazas como las defensas están evolucionando rápidamente en este campo novel.

3.8.2 Mecanismos de autenticación y cifrado en sistemas de IA generativa

La autenticación y el cifrado en sistemas de IA generativa van más allá de lo básico, sobre todo en entornos de alta criticidad donde hay que proteger datos sensibles, ya sea en reposo o en uso. Es fundamental que solo la gente autorizada pueda acceder a cierta información o ver respuestas clasificadas, y que todo esté bien cifrado y separado.

Para autenticar usuarios, ya no basta con contraseñas o tokens; ahora se usan biometría avanzada y análisis de comportamiento (como UEBA) para añadir capas extra. No obstante, incluso estos métodos enfrentan nuevos riesgos: la misma tecnología generativa ha facilitado la creación de *deepfakes* de voz o rostro que pueden intentar engañar sistemas biométricos.

Por ello, se están desarrollando métodos de autenticación anti-spoofing y mecanismos de verificación continua durante la sesión, que monitorizan indicadores de identidad para confirmar que el usuario inicial sigue siendo el mismo.

En cuanto a la autenticación y autorización a nivel de modelo, en despliegues que involucran múltiples LLMs con distintos grados de sensibilidad, es esencial implementar controles granulares de

acceso. Por ejemplo, un modelo entrenado con documentos clasificados podría restringirse a roles específicos.

El cifrado se aplica en múltiples capas. Los datos en reposo —incluyendo documentos, registros de interacciones y conjuntos de datos de entrenamiento— deben protegerse mediante algoritmos robustos como AES-256, con gestión de claves en módulos de seguridad hardware (HSM) o servicios equivalentes. En tránsito, tanto entre componentes internos como entre el usuario y el sistema, se emplean protocolos seguros como HTTPS/TLS o canales VPN en entornos controlados.

Finalmente, las estrategias de redundancia y recuperación ante incidentes son indispensables. Esto incluye *backups* cifrados y actualizados de modelos y datos, junto con planes de restauración para contrarrestar amenazas como *ransomware* o sabotaje. Asimismo, la capacidad de revocar accesos comprometidos —por ejemplo, invalidando tokens de API de manera inmediata y auditando su uso— forma parte integral de los protocolos de respuesta.

3.8.3 *Despliegue on-premise versus cloud: soberanía y cumplimiento*

Una decisión estratégica al implementar soluciones de IA generativa es elegir entre un despliegue local (*on-premise*) en infraestructuras propias o un despliegue en la nube (ya sea pública o híbrida). En el contexto de soberanía y seguridad nacional, esta elección tiene implicaciones significativas en términos de control sobre los datos, cumplimiento normativo y riesgo de exposición.

El despliegue *on-premise* ofrece el máximo control: todos los modelos, datos y componentes residirán en servidores bajo dominio de la organización (por ejemplo, en centros de datos del Ministerio de Defensa). Esto facilita el cumplimiento de políticas de soberanía de datos, asegurando que información sensible nunca abandone redes seguras y certificadas. También permite ajustar configuraciones de seguridad a medida, aislar físicamente sistemas críticos y evitar dependencias de proveedores externos en aspectos clave. Por otro lado, mantener sistemas *on-premise* conlleva una carga: se requiere contar con hardware adecuado (GPUs o TPUs para entrenamiento/ejecución de LLM), personal especializado para operarlo y escalarlo, y asumir la responsabilidad total de la continuidad del servicio y actualizaciones de seguridad.

El despliegue en la nube (pública) ofrece, en contrapartida, agilidad y potencialmente costos menores en el corto plazo (pago por uso, elasticidad). Proveedores como Microsoft, Google, Amazon y otros ofrecen plataformas robustas de IA como servicio, donde se puede desplegar un modelo o usar modelos pre-entrenados con apenas unas configuraciones. Sin embargo, surgen preocupaciones de confidencialidad: incluso si los datos están cifrados, el hecho de procesarlos en infraestructuras compartidas podría ir contra políticas gubernamentales que exigen aislamiento. Para mitigar esto, han aparecido ofertas de nube soberana, donde se garantiza que los datos permanezcan en ciertos territorios y bajo jurisdicciones nacionales, con controles adicionales (por ejemplo, Azure Government o regiones dedicadas de AWS/Google Cloud) [60].

Un término medio sería adoptar enfoques híbridos: mantener *on-premise* la información más crítica y quizás los componentes de inferencia sensibles, mientras se aprovecha la nube para entrenamiento o para escalar cargas no confidenciales. Por ejemplo, se podría entrenar un modelo base en la nube con datos públicos y luego hacer el ajuste fino final *on-premise* con los datos clasificados [60].

En cuanto a cumplimiento, el Reglamento de IA de la UE mencionado define diferentes niveles de riesgo para los sistemas de IA. Un sistema que asista en entornos críticos podría ser considerado de alto riesgo (dado que errores graves podrían afectar seguridad o cumpliría funciones sensibles). Para estos, la normativa impone obligaciones como gestión continua de riesgos, gobernanza de datos (asegurando calidad y representatividad), documentación técnica extensa, mantenimiento de registros, supervisión humana y robustez/ciberseguridad demostrable. Un despliegue *on-premise* facilita satisfacer ciertos controles (por ejemplo, registro en base nacional, seguimiento por auditorías internas), mientras que en la nube habría que cerciorarse de que el proveedor cumpla estándares equivalentes (certificaciones como ISO/IEC 27001, mecanismos contractuales de responsabilidad compartida, etc.).

Una consideración de seguridad adicional es la actualización y parcheo. En sistemas *on-premise*, la organización debe estar al tanto y aplicar parches de seguridad a los componentes de IA (por ejemplo, si se descubre una vulnerabilidad en la biblioteca de DL usada, o en el servidor de la base vectorial). En la nube, en teoría, muchos de esos parches los aplica el proveedor sin intervención del usuario, aunque a cambio se pierde algo de visibilidad en cómo y cuándo se realizan los cambios.

3.8.4 Marcos de gobernanza y estándares de IA

Más allá de las consideraciones técnicas, la implementación de IA en entornos críticos se enmarca en un creciente ecosistema de marcos de gobernanza y estándares internacionales, destinados a asegurar un desarrollo y uso confiable, ético y seguro de la IA. Adoptar estos marcos es parte del estado del arte para organizaciones, ya que proporciona guías estructuradas para gestionar riesgos y responsabilidades.

A nivel internacional, destaca la serie de estándares ISO/IEC 42001:2023, que establece el primer Sistema de Gestión de IA (AIMS, *AI Management System*) similar a las ISO de calidad (9001) o seguridad de la información (27001). Este estándar, publicado en diciembre de 2023, define requisitos para establecer, implementar, mantener y mejorar un marco de gestión dentro de una organización que esté desarrollando o usando sistemas de IA. Incluye cláusulas sobre contexto organizacional, liderazgo, planificación, soporte, operación, evaluación de desempeño y mejora continua, con anexos que detallan controles específicos para IA (38 controles en el Anexo A) y guías para gestión de riesgos de IA (Anexo B y C, alineados con ISO 31000). La adopción de ISO 42001 puede ayudar a la Armada a sistematizar cómo gobierna sus proyectos de IA, asegurando que se identifiquen riesgos (como sesgos, fallos potenciales, impactos en derechos fundamentales) y se implementen medidas mitigadoras desde la fase de diseño hasta el despliegue y retiro de los sistemas [61].

Junto a ISO 42001, el estándar ISO/IEC 23894:2023 provee una guía específica para la gestión de riesgos de IA, complementando con detalle las buenas prácticas para identificar, evaluar y tratar riesgos inherentes a los sistemas de IA. Este tipo de guía es particularmente relevante en entornos de alto riesgo, pues va más allá de riesgos cibernéticos, cubriendo también riesgos éticos, sociales y de fiabilidad técnica [62].

Las agencias nacionales de ciberseguridad europeas también han desarrollado guías específicas. La francesa ANSSI publicó en abril de 2024 el documento "Recomendaciones de Seguridad para un Sistema de IA Generativa" (ANSSI-PA-102), proponiendo un enfoque por fases (Entrenamiento, Despliegue, Producción) con 35 recomendaciones de seguridad concretas. Entre ellas, destacan: el aislamiento de entornos (mantener separados los entornos de entrenamiento, prueba y producción, con segmentación de red para que un compromiso en uno no se extienda a otros); la seguridad de los datos (control estricto sobre fuentes de entrenamiento, incluyendo validación de datos y protección de las bases vectoriales contra exfiltración); la prevención de ataques de manipulación (asegurando el sistema contra *prompts* adversarios, envenenamiento de datos o inserción de *backdoors* en modelos); controles de acceso basados en roles bien definidos para usuarios, desarrolladores y administradores; filtrado de entradas y salidas (implementando capas de defensa antes y después del modelo de IA, por ejemplo, un filtrado de *prompts* de usuario y un post-procesador de la respuesta); registro y monitorización exhaustivos (trazabilidad completa de consultas y respuestas para facilitar auditorías y detección de usos indebidos); y seguridad en la integración de *plugins* o herramientas externas al modelo.

En España, el Centro Criptológico Nacional (CCN) enmarca la seguridad de IA dentro del Esquema Nacional de Seguridad (ENS), adaptando sus principios a esta nueva tecnología. El ENS, establecido por Real Decreto 951/2015, obliga a clasificar los sistemas en niveles de seguridad y aplicar medidas proporcionales. Sistemas de IA que manejen información clasificada o que tengan impacto alto en operaciones probablemente deban alinearse con el nivel alto del ENS, implicando medidas rigurosas de control de acceso, cifrado, monitorización y respuesta a incidentes. El CCN-CERT emite guías y alertas (los Informes de Buenas Prácticas del CCN, por ejemplo, ya han abordado aspectos de seguridad en aprendizaje automático adversario).

4 CASO DE ESTUDIO: LOGÍSTICA EN DEFENSA

La implementación de sistemas de IA en el ámbito de la defensa no constituye meramente una actualización tecnológica, sino un cambio de paradigma en la concepción del mando, el control y la gestión logística. El enfoque arquitectónico desarrollado en los capítulos anteriores encuentra aplicación natural en el contexto de la logística de municiones, un dominio que ejemplifica de forma paradigmática los tres pilares fundamentales identificados en el apartado 1.2: heterogeneidad documental, requisitos de seguridad y soberanía del dato, y complejidad operativa de alta dimensionalidad.

4.1 Panorámica general

La gestión logística de municiones representa uno de los casos más complejos y regulados dentro de la cadena logística militar, constituyendo un caso que materializa los desafíos teóricos identificados en el apartado 1.2. A diferencia de la logística comercial, donde la optimización suele centrarse en coste y velocidad, la gestión de material explosivo prioriza la seguridad integral, la trazabilidad absoluta y el cumplimiento normativo estricto bajo marcos como las publicaciones aliadas OTAN y regulaciones nacionales de mercancías peligrosas.

4.1.1 Situación actual

Como se ha mencionado anteriormente, la información crítica coexiste en formatos diversos. Esta complejidad material se traduce en un ecosistema documental vasto y fragmentado. Para la gestión logística de municiones en la Armada, hemos clasificado la información en cuatro tipologías documentales diferenciadas, cada una con su propio formato, estructura y nivel de estructuración que el sistema debe ser capaz de ingerir y procesar:

1. **Documentación normativa y de mando:** Establece el marco legal y operativo. Incluye:
 - Reglamentos y normas militares (nacionales y OTAN).
 - Autorizaciones de cargos y asignaciones de cupos operativos a las unidades.
2. **Documentación técnica:** Contiene las características del material, así como manuales, procedimientos, y es crítica para gestionar de manera segura las municiones. Incluye:
 - **Prontuario logístico:** suele acompañar tanto municiones convencionales como artificios, en esta categoría se encuentran las tarjetas de datos de munición y especificaciones técnicas (características físicas, operativas, componentes, materiales y vida útil).

Aquí también se encuentran las instrucciones de actuación ante fallos de fuego para la neutralización de la munición.

- Manuales técnicos: todos los manuales para municiones de altas prestaciones (misiles, torpedos, municiones merodeadoras). Comprenden varios tomos con instrucciones, planos, y todo tipo de documentación relativa este tipo de municiones complejas.
 - Documentación de ciclo de vida: Informes de pruebas ordinarias y extraordinarias, estudios de ampliación de vida útil (informes técnicos) y planes de desmilitarización o destrucción.
3. **Documentación administrativa:** Necesaria para la trazabilidad legal y contractual. Abarca:
- Pliegos de contratación y requisitos técnicos.
 - Certificados de garantía, calidad, conformidad y catalogación.
4. **Documentos logísticos operativos:** Generados en el día a día de las operaciones:
- Guías de entrega de munición.
 - Incidencias de munición.

En la forma tradicional de trabajar, los documentos y la información que contienen se almacenan mayoritariamente en papel o PDF, y la información relevante para la gestión logística se introduce manualmente en bases de datos relacionales digitalizadas. Esta aproximación tradicional introduce múltiples problemas: falta de información relevante en la base de datos, que posteriormente debe buscarse en archivos físicos (detalles de cada tipo de munición, tipo de empaque y dimensiones, informes de pruebas de vigilancia), además el proceso de transcripción es propenso a errores humanos.

Por otra parte, la validación de las restricciones de seguridad cruzando estas fuentes no es trivial. El almacenamiento conjunto de un artículo del grupo B (iniciadores primarios) con uno del grupo J (explosivos secundarios) puede estar prohibido o requerir medidas de segregación excepcionales dependiendo de la normativa aplicada. En la forma de trabajo tradicional, la lógica de validación reside en la experiencia del personal experto y en la consulta manual de estas fuentes dispares.

El disponer de una base de datos que depende de los datos manuales introducidos, presenta varias deficiencias críticas:

- **Falta de previsión:** No genera automáticamente previsiones de consumo a partir de los cupos asignados. Las previsiones dependen de una ardua recopilación de datos hecha por los trabajadores.
- **Silos de información:** La gestión documental se realiza en cada unidad (arsenales, cuarteles generales, unidades), pudiendo ser difícil de acceder para otras unidades.
- **Gestión por mensajería no integrada:** El sistema de envío y recepción de mensajes para movimientos o autorizaciones, hace que los datos deban ser reintroducidos por un operador. Esto duplica el trabajo y aumenta la probabilidad de introducir datos incorrectos.

A continuación, se describen los flujos de trabajo que podrían ser más problemáticos en un flujo de trabajo tradicional:

- **Optimización logística en almacenamiento**

La asignación de munición entrante a ubicaciones de almacenamiento se realiza actualmente de forma manual basándose en la experiencia del personal. Durante la redistribución de inventarios entre arsenales o la consolidación de existencias, el proceso requiere: identificar qué municiones pueden coexistir verificando grupo de riesgo y compatibilidad usando tablas de compatibilidad, calcular manualmente el NEQ acumulado para verificar que no se exceden los límites de seguridad del polvorín, e iterar múltiples configuraciones hasta encontrar una distribución válida.

Este proceso puede requerir horas o días para redistribuciones complejas, conduciendo a soluciones subóptimas donde se infrautiliza la capacidad disponible o se distribuyen las municiones en más polvorines de los necesarios, complicando la gestión y vigilancia posterior.

- **Optimización logística para transportes**

La planificación de transportes de munición enfrenta restricciones múltiples que actualmente se gestionan de forma manual y secuencial: verificación de compatibilidad consultando tablas de segregación para cada par de artículos, cálculo de capacidad volumétrica sin herramientas de optimización del espacio, y desconocimiento del volumen y peso final hasta tener la carga conformada. Esta última limitación impide la planificación global de costes, tiempos y disponibilidad de medios de transporte, siendo crítica en redistribuciones entre arsenales o grandes despliegues operativos.

La ausencia de optimización conduce a soluciones que requieren más viajes o medios de los estrictamente necesarios, incrementando costes y tiempos de entrega.

4.1.2 Solución propuesta

El sistema propuesto, denominado Sistema Inteligente de Gestión Logística de Municiones (SIGLM, nombre ficticio), tiene como objetivo fundamental digitalizar la lógica experta que actualmente reside en el personal especializado, optimizando la gestión logística de municiones dentro de la Armada mediante la aplicación de IA.

Esta solución se plantea bajo una arquitectura capaz de procesar documentación heterogénea —tanto estructurada como no estructurada— para minimizar los errores humanos derivados de la transcripción manual, garantizar la integridad de la información y facilitar la toma de decisiones estratégicas y operativas.

Para lograrlo, el sistema estructura sus capacidades en torno a la ingesta inteligente, la gestión del ciclo de vida, la validación de seguridad y la interacción en lenguaje natural. A continuación, se detalla cada una de estas capacidades:

1. **Ingesta Inteligente y Procesamiento de Lenguaje Natural:** Esta capacidad permite la introducción automática de documentación, realizando tareas de extracción, clasificación y estructuración de datos para minimizar errores de transcripción u omisión. El sistema está diseñado para gestionar fuentes de entrada críticas:
 - **Documentación normativa y técnica:** Procesamiento de normas (OTAN, nacionales y propias de la Armada), así como manuales técnicos de cada tipo de munición.
 - **Documentación operativa:** Lectura automatizada de actas de recepción, guías de entrega, informes de reconocimientos físicos o químicos, y tablas de temperatura y humedad en almacenamiento.
 - **Mensajes operativos (NLP):** Capacidad para interpretar instrucciones en lenguaje natural (ej. "Gasté 100 balas de 12,7 mm del lote XXXX") o mensajes no estructurados, extrayendo entidades clave (NSN, designación, cantidad, origen, destino) para proponer transacciones validadas.
2. **Gestión Inteligente de Inventario y Trazabilidad (Data Lineage):** Una vez procesada la información, el sistema permite un control exhaustivo sobre el inventario y la "historia" del dato, asegurando la trazabilidad desde el documento origen hasta el estado actual:
 - **Catalogación y Ubicación:** Gestión centralizada de municiones, misiles y torpedos, con control preciso de su distribución en los distintos polvorines y ubicaciones.

- **Control de Estado y Vida Útil:** Monitorización del estado de conservación, validando si los lotes poseen ampliaciones de vida útil vigentes (basado en la extracción de informes técnicos) y detectando necesidades de desmilitarización.
 - **Gestión de Cupos:** Control y seguimiento de los cupos de munición asignados frente a los consumidos.
3. **Motor de Validación de Seguridad (Híbrido) y Optimización:** El SIGLM integraría un motor que combina reglas deterministas con extracción por IA para garantizar la seguridad y eficiencia:
- **Validación de Restricciones:** Verificación automática de compatibilidades de transporte y almacenamiento (ej. compatibilidad entre riesgos 1.2E y 1.4S según normativa AASTP-5) y distancias de seguridad. El sistema rechaza operaciones inseguras generando explicaciones fundamentadas.
 - **Optimización Logística:** Uso de algoritmos (*solvers*) para la asignación óptima de munición a polvorines, planificación de transportes y paletización de carga, respetando las restricciones de seguridad y maximizando la operatividad.
4. **Asistencia Técnica (RAG) y Generación Automatizada de Documentación:** El sistema incorpora una gestión de usuarios basada en roles (desde personal técnico hasta Estado Mayor), permitiendo la interacción con consultas en lenguaje natural y la generación de salidas complejas:
- **Consultas Operativas:** Respuestas inmediatas sobre stock y situación (ej. “¿Cuántos proyectiles de 155mm quedan?”, “¿En dónde están?”, “¿Qué capacidad tengo en el polvorín XXX?”).
 - **Consultas Semánticas (RAG):** Análisis fundamentado en el corpus documental (ej. “¿Qué necesidades de munición tendré el próximo año?”, consultas técnicas o de normativa).
 - **Acciones Estructuradas:** Ejecución de inserciones, modificaciones o actualizaciones en base de datos previa confirmación humana.
 - **Reportes Automatizados:** Soporte a la redacción de informes y análisis, consolidando información de múltiples fuentes.

El sistema busca digitalizar la lógica experta que actualmente reside en el personal especializado, ingiriendo la documentación heterogénea descrita en el apartado 4.2 —que varía desde manuales técnicos de torpedos con diagramas complejos hasta directivas de seguridad en lenguaje natural— para permitir consultas semánticas y validaciones automáticas. La arquitectura propuesta utiliza modelos de lenguaje (LLM) para unificar estas fuentes, permitiendo al usuario formular preguntas complejas como: “¿Es compatible el transporte de munición 1.2E con 1.4S según la normativa AASTP-5?” o “Verificar si el lote X tiene una ampliación de vida útil vigente según el último informe INTA”.

Como resultado, el sistema no solo debe devolver información, sino que ejecutar acciones estructuradas (inserciones, modificaciones o actualizaciones en la base de datos previa confirmación humana) y generar reportes automatizados, consolidando información de múltiples fuentes para responder a los requerimientos del Mando.

A continuación, se detallan las capacidades técnicas del sistema para resolver la complejidad operativa: integración de fuentes heterogéneas, validación de restricciones regulatorias, optimización combinatoria y generación de reportes complejos.

4.1.2.1 Integración de fuentes heterogéneas en tiempo real

La gestión de municiones genera flujos continuos de información desde múltiples fuentes. Cada una de estas fuentes utiliza formatos y estructuras diferentes y todas deben integrarse de forma coherente actualizando el estado global del inventario sin introducir inconsistencias.

La problemática de la operativa manual, que se vio en el apartado 4.1.1, introduce múltiples puntos de fallo: errores de transcripción en NSNs o códigos de lote, interpretación incorrecta de cantidades o unidades de medida, omisión de información crítica presente en el mensaje original y retrasos temporales entre la recepción del mensaje y su reflejo en el sistema pueden generar ventanas de inconsistencia donde el estado registrado no coincide con la realidad operativa.

El SIGLM debe automatizar este proceso de extracción de información estructurada desde mensajes no estructurados, empleando técnicas de procesamiento de lenguaje natural para identificar entidades clave (NSN, designación de munición, cantidad, origen, destino, fecha de movimiento), validar la coherencia semántica de la información extraída (verificando, por ejemplo, que las cantidades sean positivas, que las fechas sean coherentes con la fecha del mensaje, que los códigos NSN existan en el catálogo), y proponer transacciones estructuradas para su confirmación por el operador antes de ejecutar las actualizaciones en la base de datos relacional. Esta arquitectura semiautomática reduce los errores de transcripción manteniendo supervisión humana en el flujo crítico de actualización de inventarios.

4.1.2.2 Validación automática de restricciones de seguridad

Las restricciones de almacenamiento, transporte y manipulación de municiones constituyen reglas deterministas derivadas de normativa técnica que deben verificarse de forma automática y exhaustiva antes de autorizar operaciones. Estas restricciones son complejas e interdependientes:

- **Compatibilidad de grupos:** La normativa AASTP-1 y AASTP-5 de la OTAN establece tablas de compatibilidad entre grupos de riesgo (1.1, 1.2, 1.3, 1.4, 1.5, 1.6) y grupos de compatibilidad (designados por letras de la A a la S). Determinadas combinaciones están prohibidas bajo cualquier circunstancia (por ejemplo, iniciadores del grupo B con determinados explosivos secundarios), otras requieren medidas de segregación específicas (distancias mínimas, barreras físicas, compartimentación), y otras son compatibles sin restricciones adicionales. El sistema debe consultar estas tablas complejas evaluando no solo pares de artículos sino configuraciones completas de polvorines o compartimentos de transporte donde coexisten múltiples tipos de munición simultáneamente.
- **Distancias de seguridad:** El almacenamiento de munición con determinadas NEQ (Cantidad Neta de Explosivo) requiere distancias mínimas respecto a instalaciones habitadas, vías de comunicación, y otros almacenes de explosivos. Estas distancias se calculan mediante fórmulas que dependen del tipo de explosivo, el NEQ total almacenada, y el tipo de construcción del polvorín (enterrado, superficie, reforzado). Las distancias de seguridad además son diferentes en función de los grupos de riesgo almacenados, o combinaciones de los mismos que haya. El sistema debe poder calcular estas distancias automáticamente y verificar que la ubicación propuesta para un nuevo lote cumple las separaciones requeridas respecto al resto de instalaciones del arsenal, alertando cuando se exceden umbrales críticos de NEQ acumulada en una zona.
- **Condiciones ambientales y vida útil:** Cada tipo de munición tiene especificaciones de temperatura y humedad de almacenamiento, intervalos de inspección de vigilancia, y vida útil condicionada a las condiciones de almacenamiento reales. El sistema debe monitorizar el cumplimiento de estas condiciones, generando alertas cuando se exceden umbrales ambientales que puedan degradar el material, o cuando la vida útil está próxima a expirar y se requieren estudios de ampliación o procesos de baja y desmilitarización.

La validación manual de estas restricciones cruzando múltiples manuales técnicos y tablas normativas es el proceso lento y propenso a errores que el SIGLM busca automatizar. El sistema debe

incorporar un motor de reglas que codifique estas restricciones de forma declarativa, ejecute las validaciones sobre el estado actual del inventario y las operaciones propuestas, y genere explicaciones comprensibles cuando una operación es rechazada (p. ej. *"El almacenamiento conjunto de NSN X (1.1D, 450 kg NEQ) con NSN Y (1.2E, 300 kg NEQ) en el polvorín Z excede la NEQ máxima permitida según AASTP-1 tabla 5.3 para polvorines de superficie tipo A, límite 500 kg"*).

4.1.2.3 Problemas de optimización combinatoria

La asignación óptima de munición a ubicaciones de almacenamiento, la planificación de transportes, y la preparación de paletas de carga para embarque en buques constituyen problemas de optimización combinatoria NP-duros donde el espacio de soluciones crece exponencialmente con el número de elementos a asignar. Actualmente se requieren horas de trabajo para recopilar datos sobre existencias en distintas ubicaciones de cada tipo de munición para adquisiciones o redistribuciones. Esta recopilación manual, artículo por artículo, de las existencias dispersas en múltiples arsenales y unidades operativas para determinar necesidades agregadas y planificar redistribuciones óptimas es precisamente el tipo de tarea que puede automatizarse mediante técnicas de optimización.

Formalmente, el problema de asignación de munición a polvorines respetando restricciones de capacidad volumétrica, NEQ máxima por polvorín, compatibilidad de grupos, y minimizando el número de polvorines utilizados (para facilitar gestión y vigilancia) puede formularse como un problema de *bin packing* multidimensional con restricciones de incompatibilidad. La resolución requiere el uso de *solvers* de optimización lineal entera mixta (MILP) o algoritmos heurísticos que exploren eficientemente el espacio de soluciones generando asignaciones factibles cuasi-óptimas en tiempos computacionales razonables.

De forma similar, la planificación de transportes de munición entre ubicaciones, minimizando costes y tiempos mientras se respetan restricciones de compatibilidad en los vehículos de transporte, capacidades de carga, y ventanas temporales de disponibilidad de medios de transporte, constituye una variante del problema de ruteo de vehículos con restricciones que requiere técnicas especializadas de optimización combinatoria.

4.1.2.4 Generación de reportes complejos y apoyo a la decisión

El Estado Mayor requiere capacidades de análisis y generación de reportes que integren información de múltiples fuentes para apoyar decisiones estratégicas y operativas. Estas consultas van más allá de simples agregaciones de inventario para incorporar razonamiento sobre normativa técnica, proyecciones de consumo basadas en cupos operativos asignados, análisis de riesgos por concentración excesiva de material explosivo en ubicaciones específicas, o evaluación de disponibilidad operativa considerando vidas útiles próximas a expirar.

Por ejemplo, una consulta típica del tipo "generar informe de capacidad operativa de artillería de 76mm considerando existencias actuales, consumos proyectados según cupos de las unidades para los próximos seis meses, vidas útiles de los lotes almacenados, y restricciones de disponibilidad por incidencias técnicas documentadas" requiere:

1. Consultar la base de datos relacional de inventarios filtrando por familia de proyectiles de artillería de 76mm y agregando existencias por arsenal y unidad operativa.
2. Recuperar de la documentación administrativa los cupos asignados a cada unidad y calcular tasas de consumo proyectadas.
3. Consultar los manuales técnicos y certificaciones de vida útil para identificar lotes con fecha de expiración en el horizonte temporal del análisis.
4. Buscar en la documentación operativa (informes de incidencias, mensajes de fallo) cualquier notificación de problemas técnicos que afecte a lotes específicos.

5. Consolidar toda esta información en un reporte estructurado que presente la disponibilidad neta considerando todos los factores, identifique cuellos de botella potenciales, y fundamente cada afirmación con referencias a los documentos y registros que sustentan el análisis.

La generación manual de este tipo de reportes por parte del personal especialista es el proceso intensivo en tiempo. La arquitectura del SIGLM, combinando consultas a bases de datos relacionales, búsqueda semántica sobre el corpus documental mediante RAG, y orquestación de múltiples agentes especializados, permite automatizar la generación de estos reportes complejos respondiendo a consultas formuladas en lenguaje natural por los tomadores de decisiones.

4.1.2.5 Gestión del conocimiento institucional y continuidad operativa

El SIGLM debe actuar como repositorio centralizado de conocimiento institucional, capturando no solo los documentos formales sino también las decisiones operativas históricas, las justificaciones técnicas de asignaciones o redistribuciones realizadas, y los criterios aplicados en situaciones excepcionales que constituyen precedentes valiosos. Esta base de conocimiento consultable a través de interfaces conversacionales permite que personal con menor experiencia acceda rápidamente a información crítica y aprenda de las decisiones pasadas, reduciendo la curva de aprendizaje y mejorando la resiliencia organizativa ante contingencias.

4.1.2.6 Pilares funcionales del sistema

El SIGLM se ha estructurado en torno a cuatro pilares funcionales que cubren el ciclo de vida de la información logística:

1. **Asistencia Técnica y Normativa (Motor RAG):** Esta funcionalidad constituye el núcleo de inteligencia del sistema. Permite a los operadores realizar búsquedas semánticas sobre el corpus documental. A diferencia de un buscador por palabras clave, el motor RAG comprende la intención de la pregunta y recupera los fragmentos de texto más relevantes para sintetizar una respuesta precisa.
 - *Trazabilidad de la Información:* Cada respuesta generada incluye citas directas a los documentos fuente (título, página, párrafo), permitiendo al usuario verificar la información original. Esto es crucial para mantener la cadena de confianza en entornos militares.
 - *Gestión de la Ambigüedad:* Si la documentación contiene normas contradictorias (por ejemplo, una diferencia entre una norma nacional y una OTAN), el sistema debe estar diseñado para resaltar esta discrepancia en lugar de alucinar una resolución, presentando ambas fuentes al operador para su juicio experto.
2. **Cálculo y Validación de Seguridad (Módulos Híbridos):** El sistema integra la capacidad de extracción de información de los LLM con motores de cálculo deterministas.
 - *Cálculo de NEQ:* El sistema extrae automáticamente los valores de NEQ de las fichas técnicas de los lotes entrantes y actualiza en tiempo real la carga de fuego de los polvorines. Si una operación de movimiento propuesta excede los límites de seguridad calculados según los algoritmos de distancia-cantidad (*Quantity-Distance*, QD) definidos en los manuales de seguridad, el sistema emite una alerta bloqueante.
 - *Matriz de Compatibilidad Dinámica:* Al planificar un transporte, el sistema cruza los códigos de clasificación de todos los ítems involucrados contra las matrices de compatibilidad digitalizadas, alertando sobre mezclas prohibidas (ej. Grupo A con cualquier otro grupo).

3. **Gestión Inteligente de Inventario y Trazabilidad:** El sistema actúa como una capa de inteligencia sobre el sistema de gestión de almacén tradicional. Utiliza agentes de IA para procesar documentación no estructurada que acompaña a la munición, como albaranes de proveedores, actas de tiro y reportes de inspección de calidad.

- Trazabilidad de datos: Se mantiene un registro inmutable de cada cambio de estado de la munición, desde "En servicio" a "Restringido" o "Caducado", basándose en la interpretación automática de los informes técnicos.

4. **Generación Automatizada de Documentación:** El sistema asiste en la redacción de informes reglamentarios, solicitudes de transporte y documentación de auditoría. Utilizando plantillas predefinidas y la capacidad generativa del LLM, el sistema puede redactar borradores de informes de incidencias o memorandos de seguridad, reduciendo significativamente la carga burocrática del personal técnico.

4.1.2.7 Beneficios esperados

La implementación del SIGLM persigue objetivos tangibles en términos de eficiencia, seguridad y gestión del conocimiento. La Tabla 4.1 resume los beneficios esperados y los indicadores asociados.

Dimensión	Beneficio Esperado	Justificación
Eficiencia Operativa	Reducción del tiempo dedicado a la búsqueda de información técnica en manuales dispersos.	El acceso instantáneo a datos técnicos precisos acelera la toma de decisiones en mantenimiento y logística.
Seguridad Operativa	Minimización del error humano en cálculos de NEQ, de QD y validación de compatibilidad.	La validación algorítmica actúa como una segunda capa de seguridad redundante sobre el juicio humano.
Gestión del Conocimiento	Preservación del conocimiento experto y reducción de la curva de aprendizaje.	El sistema democratiza el acceso al conocimiento experto contenido en la documentación técnica.
Cumplimiento Normativo	Aseguramiento de la aplicación de la normativa vigente y auditoría continua.	La actualización centralizada de la normativa asegura que todos los cálculos se basen en la última versión vigente.
Planificación Estratégica	Anticipación en la gestión de inventarios y apoyo a la toma de decisiones de aprovisionamiento.	La capacidad analítica del sistema permite identificar patrones de consumo, prever necesidades futuras y alertar sobre vencimientos próximos, transformando datos históricos en inteligencia estratégica para el mando.

Tabla 4-1. Beneficios Esperados e Indicadores de Impacto del SIGLM

4.1.2.8 Seguridad, soberanía del dato y controles de integridad

Los requisitos de seguridad en la gestión logística de municiones trascienden consideraciones técnicas para convertirse en imperativos de seguridad nacional. La naturaleza clasificada de la información —capacidades operativas, ubicaciones de arsenales, existencias específicas— exige que la seguridad se conciba no como un módulo añadido, sino como una propiedad demostrable del sistema integrada desde el diseño (*security by design*). El control de acceso, la soberanía tecnológica, el cumplimiento normativo y la auditoría exhaustiva constituyen pilares fundamentales de la arquitectura propuesta.

La información gestionada presenta múltiples niveles de clasificación que deben segregarse rigurosamente. Las especificaciones técnicas detalladas de sistemas de armas, existencias concretas por ubicación y planes de asignación operativa constituyen información clasificada cuya exposición no autorizada comprometería capacidades estratégicas de la Fuerza. El sistema debe implementar controles de acceso granulares basados en el principio de mínimo privilegio: cada usuario accede únicamente a la información estrictamente necesaria para su función operativa. Esta segmentación requiere marcado de documentos con etiquetas de clasificación jerárquicas, políticas de acceso dinámicas que evalúen autorización y rol funcional, enmascaramiento o redacción de contenido cuando proceda para mostrar versiones sanitizadas, e integración con sistemas corporativos de autenticación. La implementación de RBAC (*Role-Based Access Control*) permite refinar roles técnicos básicos (*reader/writer/admin*) a roles operativos específicos (artificiero, oficial de logística, auditor), reforzados con control de acceso a nivel de registro (Row-Level Security) cuando sea necesario.

El SIGLM estaría concebido para desplegarse en la infraestructura clasificada del Ministerio de Defensa, específicamente en el Sistema de Mando y Control Nacional (SC2N), red acreditada para manejar información clasificada hasta nivel RESERVADO NACIONAL que forma parte de la Infraestructura Integral de Información de la Defensa (I3D). Esta red, que alcanzó su capacidad operativa inicial en 2022, proporcionaría el entorno seguro necesario para el procesamiento de información logística clasificada. El despliegue en SC2N garantiza que todos los componentes del sistema —bases de datos, modelos de lenguaje, servicios de procesamiento documental— residen en el "Núcleo Protegido" de la I3D, ubicado en instalaciones militares, con acceso físico restringido. Esta arquitectura permite la segregación física entre dominios de seguridad, donde información de máxima clasificación se procesa en servidores físicamente aislados sin conectividad externa. La prohibición absoluta de utilizar servicios en la nube de terceros o APIs externas queda satisfecha con este despliegue completo *on-premise*, eliminando cualquier exposición a jurisdicciones extranjeras y manteniendo soberanía tecnológica completa sobre el procesamiento de información sensible.

La capacidad de auditoría constituye un requisito no negociable del sistema. Cada consulta, modificación de inventario, acceso a manuales y generación de reportes debe registrarse en logs de auditoría inmutables que capturen: identidad del usuario, *timestamp*, recurso accedido, operación realizada, contexto operativo y resultado. Estos registros permiten la reconstrucción forense de eventos y la activación de alertas ante patrones anómalos. Todas las operaciones que alteran el estado del sistema pasan por un flujo controlado que incluye validación automática mediante reglas SQL y reglas de negocio, confirmación humana con presentación clara de los cambios propuestos, y registro inmutable que documenta quién solicitó la operación, qué documentación se utilizó como base, y la trazabilidad completa de la decisión automatizada.

Para respuestas generadas por el asistente conversacional, el sistema mantiene trazabilidad completa orientada a reproducibilidad y explicabilidad: versión del modelo empleado, consulta original, *prompt* final construido, respuesta producida y referencias a los fragmentos utilizados como soporte. Esta trazabilidad permite a auditores reconstruir cómo el sistema alcanzó una respuesta específica y diferenciar errores atribuibles a la documentación de origen frente a limitaciones del modelo.

La arquitectura técnica implementa múltiples controles de integridad y fiabilidad demostrable. El cálculo del hash SHA256 sobre cada documento original y su uso como clave única garantiza idempotencia documental, evitando que un mismo documento se procese múltiples veces con efectos operativos indeseados. El uso de transacciones a nivel SERIALIZABLE en PostgreSQL para todo el flujo de ingesta implementa el principio de atomicidad (la 'A' de ACID), asegurando que la cadena de operaciones —OCR, normalización, inserción en múltiples tablas— se completa en su totalidad o no se ejecuta en absoluto, previniendo estados de datos corruptos o inconsistentes.

4.2 Prototipo de validación (PoC)

Con el objetivo de validar las hipótesis fundamentales de este trabajo —específicamente la viabilidad de una arquitectura híbrida que combine RAG (no determinista) con validaciones SQL (deterministas) para la seguridad de municiones— se ha desarrollado una Prueba de Concepto (PoC). Este prototipo implementa el núcleo lógico del sistema SIGLM, abstrayendo la complejidad de infraestructura distribuida para centrarse en la corrección de los flujos de datos y seguridad.

La decisión de construir este prototipo responde a la necesidad de demostrar empíricamente tres aspectos críticos del diseño propuesto. En primer lugar, la posibilidad real de que un LLM actúe como orquestador central capaz de enrutar consultas según su naturaleza (semántica vs. transaccional) sin incurrir en errores de clasificación que comprometan la seguridad. En segundo lugar, la efectividad del patrón híbrido de almacenamiento: validar que consultas sobre normativa técnica se resuelvan mediante búsqueda semántica en base vectorial mientras que operaciones de inventario y restricciones de seguridad se ejecuten determinísticamente contra bases de datos relacionales. En tercer lugar, demostrar que la integración de OCR selectivo permite procesar documentación heterogénea (PDFs escaneados, albaranes físicos digitalizados) dentro de un pipeline de ingesta automatizado, extrayendo información estructurada que alimente ambos sistemas de almacenamiento.

4.2.1 Alcance del prototipo

El prototipo adopta intencionadamente un enfoque monolítico implementado en Python con Streamlit como capa de presentación, priorizando agilidad de desarrollo sobre distribución arquitectónica. Esta decisión se justifica en el contexto de un trabajo académico donde el objetivo primordial es validar conceptos antes que optimizar para escala. No obstante, el diseño interno del prototipo mantiene una separación lógica clara entre capas que se corresponde directamente con la arquitectura de microservicios propuesta para producción.

La interfaz de usuario construida con Streamlit simula el *frontend* Angular que se proyecta para el sistema final. Proporciona componentes interactivos para la carga de documentos PDF, un chatbot conversacional para consultas en lenguaje natural, y paneles de visualización del inventario de municiones. Aunque carece de la complejidad de gestión de estado (NgRx), ruteo avanzado y componentes modulares de Angular, implementa los mismos flujos de usuario fundamentales: autenticación basada en roles (operador, supervisor), carga de normativa técnica y albaranes de movimiento, consultas sobre capacidad de polvorines, validación de restricciones de seguridad, y ejecución de operaciones de inventario.

La lógica de negocio se implementa en Python puro, articulando funciones especializadas que emulan el comportamiento de microservicios independientes. Cada función encapsula una responsabilidad acotada y se comunica con las demás pasando parámetros estructurados, que simulan las interfaces de API (REST/gRPC) de la arquitectura distribuida. Por ejemplo, la función `validar_polvorin()` replica el comportamiento del "*Safety Rules Engine Service*" al consultar documentación normativa para verificar límites de NEQ, mientras que `ejecutar_movimiento_municion()` actúa como el "*Inventory Transaction Service*" orquestando validaciones múltiples antes de modificar la base de datos.

La capa de datos utiliza SQLite como base relacional para almacenar inventario de municiones (tabla inventario con campos: `polvorin_id`, `nombre_municion`, `cantidad`, `neq_total_lote`, `grupo_riesgo`), reproduciendo la estructura que tendría PostgreSQL en producción pero con un motor embebido que elimina dependencias de infraestructura. Para la base de datos vectorial se emplea ChromaDB, una alternativa ligera a Milvus que soporta el mismo flujo: generación de *embeddings* con modelos de código abierto, indexación HNSW para búsqueda aproximada de vecinos, y recuperación de fragmentos relevantes mediante similitud coseno. Aunque ChromaDB carece de las optimizaciones de escalabilidad

de Milvus (sharding, replicación), implementa correctamente la semántica de búsqueda vectorial necesaria para validar el patrón RAG.

4.2.2 Mapeo de componentes

La arquitectura lógica del prototipo replica los patrones fundamentales de la solución completa mediante correspondencias directas entre funciones Python y servicios de producción. Este mapeo no es meramente analógico sino que constituye una implementación simplificada pero funcionalmente equivalente de los mismos algoritmos y flujos de decisión.

1. Orquestador Central (LLM Router)

En la arquitectura de microservicios, el *Saga Orchestrator Service* coordina transacciones distribuidas manteniendo máquinas de estado complejas y gestionando compensaciones ante fallos. El prototipo implementa este concepto con la función `ejecutar_con_herramientas()`, que actúa como punto de entrada único para todas las consultas de usuario. Esta función incorpora un sistema de prioridades estructurado en ocho niveles que clasifican intenciones mediante patrones de expresión regular combinados con análisis del LLM:

- **Prioridad 0** - Comandos de movimientos desde albaranes escaneados: Detección de instrucciones para ejecutar operaciones previamente extraídas de PDFs, implementada mediante tres sub-comandos específicos: (0A) ejecutar movimiento #N, (0B) detalles movimiento #N, y (0C) listar movimientos pendientes.
- **Prioridad 1** - Confirmaciones de acciones: Respuestas del usuario confirmando operaciones propuestas por el sistema (sí, confirmar, adelante), gestionando flujos multi-turno con estado persistente en `st.session_state`.
- **Prioridad 2** - Consultas sobre polvorines: Preguntas sobre capacidades, características técnicas o listados de almacenes, enrutadas a búsqueda RAG sobre documentación.
- **Prioridad 3** - Consultas de inventario SQL: Solicitudes de estado cuantitativo actual (stock, existencias) ejecutadas determinísticamente contra SQLite.
- **Prioridad 4** - Extracciones de munición: Operaciones de salida de inventario con validación de stock disponible.
- **Prioridad 5** - Adiciones de munición: Operaciones de entrada con validación completa en cascada (existencia polvorín, capacidad NEQ, compatibilidad de grupos).
- **Prioridad 6** - Consultas técnicas generales: Preguntas sobre normativa, especificaciones, compatibilidades, resueltas usando RAG.
- **Prioridad 7** - Conversación general: Respuestas contextuales del LLM para interacciones no clasificadas en categorías anteriores.

La decisión de implementar clasificación inicial mediante *regexes* (en lugar de depender exclusivamente del LLM) es intencional: proporciona un cortafuegos determinista que previene errores de enrutamiento del modelo en consultas críticas. Por ejemplo, una solicitud que contenga verbos de movimiento específicos (mete, añadir, cargar, transportar, almacenar) activa invariablemente el flujo de adición de munición, independientemente de ambigüedades en el resto del contexto. Esta estrategia híbrida (reglas deterministas para intenciones críticas, LLM para casos ambiguos) refleja directamente el principio arquitectónico de separación entre componentes generativos y deterministas establecido en el capítulo 3.

El sistema gestiona estado conversacional mediante `st.session_state`, manteniendo persistencia de información entre turnos: movimientos pendientes de albaranes escaneados (`movimientos_pendientes`), índice de operación a confirmar (`movimiento_a_ejecutar`), e historial completo de conversación (`messages`). Esta gestión de estado simula las capacidades de coordinación transaccional del Saga

Orchestrator, permitiendo flujos complejos donde el sistema propone acciones basadas en extracción de datos de PDFs y solicita confirmación explícita antes de modificar inventario.

Una vez clasificada la intención con el sistema de prioridades, el orquestador puede invocar directamente funciones especializadas o delegar en el LLM mediante el mecanismo de *binding* de herramientas de LangChain. En este segundo caso, el modelo recibe las descripciones de las cinco herramientas disponibles decoradas con `@tool` y genera llamadas estructuradas en formato JSON que el *framework* interpreta para ejecutar las funciones apropiadas. Este patrón simula la comunicación entre microservicios mediante APIs contractuales: cada herramienta expone una interfaz bien definida (parámetros tipados, semántica documentada) y el orquestador las compone dinámicamente según el contexto de la conversación.

2. Agentes Especializados (Tools)

El prototipo define cinco herramientas principales decoradas con `@tool` de LangChain, cada una encapsulando una capacidad específica del sistema. Estas herramientas están definidas en el código fuente del prototipo (ver Anexo II. Código prototipo SIGLM) y constituyen las abstracciones de alto nivel que el LLM puede invocar dinámicamente:

consultar_inventario(filtro: str = ""): Replica el *"Inventory Query Service"*. Ejecuta consultas SQL parametrizadas sobre SQLite con agregación por polvorín y tipo de munición. Implementa el principio fundamental de que consultas sobre estado cuantitativo actual (cuántas unidades hay, en qué polvorines) deben resolverse determinísticamente contra la fuente de verdad transaccional, nunca con respuestas generadas por un modelo que podría alucinar cantidades incorrectas.

consultar_capacidad_polvorin(polvorin_id: str): Integra datos de ambas fuentes de almacenamiento para proporcionar vista consolidada de ocupación. Combina información normativa estática (límites de diseño de polvorines) desde búsqueda semántica sobre PDFs con información transaccional dinámica (contenido actual) desde base relacional. Esta función materializa el patrón híbrido fundamental del sistema.

consultar_documentacion_tecnica(pregunta: str): Implementa búsqueda híbrida que combina BM25 (Okapi Best Match 25) para coincidencia exacta de palabras clave, técnicas con *embeddings* semánticos para captura de significado contextual. La fusión de ambos resultados con pesos configurables resuelve la limitación de cada método aislado: BM25 es superior para códigos técnicos específicos mientras que *embeddings* capturan mejor consultas conceptuales. Este flujo replica exactamente el comportamiento del *"AI Adapter & Orchestration Service"* en producción.

ejecutar_movimiento_municion(polvorin_destino, nombre_municion, cantidad, neq_unitario, grupo_riesgo): Simula el *"Inventory Transaction Service"* con validaciones en cascada que verifican secuencialmente: validación de datos técnicos con búsqueda automática en PDFs si faltan, consulta de estado actual del polvorín, validación de capacidad NEQ consultando a documentación normativa, validación de compatibilidad de grupos de riesgo consultando normativa AASTP, y solo tras superar todas las validaciones, ejecución transaccional con confirmación inmediata. Esta estructura de validaciones en cascada, donde cada paso puede abortar la operación antes de tocar la base de datos, materializa el patrón de programación defensiva esencial en sistemas críticos.

extraer_municion(polvorin_origen, nombre_municion, cantidad): Gestiona operaciones de salida de inventario con validaciones simétricas a las de entrada. Implementa búsqueda difusa con SQL LIKE para coincidencia parcial, valida stock suficiente, y mantiene trazabilidad retornando mensajes detallados de la operación ejecutada.

El funcionamiento detallado de cada una de estas herramientas, con casos de uso específicos y capturas de pantalla del prototipo, se documenta exhaustivamente en el Anexo I. Prototipo SIGLM: casos de uso, donde se presentan cinco escenarios representativos que demuestran la integración práctica de estos componentes.

El sistema de herramientas valida empíricamente tres hipótesis arquitectónicas clave: primero, la capacidad de composición donde un LLM actúa como *router* inteligente seleccionando dinámicamente las funciones apropiadas según contexto conversacional; segundo, la separación de responsabilidades donde cada *tool* encapsula una responsabilidad única y delega en las fuentes de datos apropiadas (RAG para normativa, SQL para transacciones); y tercero, la robustez ante errores a través de búsqueda automática de datos faltantes en PDFs y validación estricta de parámetros antes de ejecución.

4.2.3 Integración de componentes y flujos end-to-end

El prototipo implementa tres flujos integrados que demuestran la composición completa del sistema: procesamiento documental con OCR selectivo y detección automática de albaranes, consultas RAG con *grounding* estricto en documentación recuperada, y operaciones transaccionales con validaciones de seguridad determinísticas. Cada flujo involucra múltiples componentes coordinados por el orquestador central.

El procesamiento documental implementa estrategia adaptativa que prioriza extracción de texto nativo de PDFs pero recurre a OCR cuando detecta páginas escaneadas mediante heurísticas de calidad textual. Los documentos se fragmentan con *overlap* configurable, se generan *embeddings* locales, y se indexan en ChromaDB. Paralelamente, el sistema detecta albaranes de movimiento mediante clasificación híbrida (patrones determinísticos + LLM) y extrae campos estructurados que se almacenan en cola de operaciones pendientes. Este pipeline valida la capacidad de procesamiento heterogéneo sin configuración manual por documento.

Las consultas RAG ejecutan búsqueda vectorial con recuperación de fragmentos relevantes, construcción de *prompt* aumentado con instrucciones de *grounding* estricto, y generación de respuesta condicionada exclusivamente al contexto recuperado. El *prompt engineering* defensivo incluye instrucciones explícitas de no inventar información ausente, contrarrestando la tendencia del modelo a completar patrones especulativamente. Este flujo demuestra que LLMs pueden proporcionar capacidades de comprensión semántica sin comprometer la confiabilidad en dominios críticos.

Las operaciones transaccionales implementan arquitectura defensiva de cinco capas: detección de intención mediante *regex*, extracción de parámetros por LLM con *prompt* estructurado, enriquecimiento automático vía RAG de datos técnicos faltantes, validaciones de negocio determinísticas consultando normativa para verificar límites de seguridad, y confirmación bidireccional humana como *guardrail* final. La función `validar_polvorin()` materializa la separación estricta entre razonamiento semántico (comprensión de documentación compleja por el LLM) y autorización de seguridad (lógica *if/then* clásica sobre valores extraídos). Esta separación no solo mitiga riesgos de alucinación, sino que facilita auditoría: logs pueden registrar "operación rechazada por NEQ=157.3kg > límite=150kg" (trazabilidad total) versus "operación rechazada porque el LLM dijo que no" (caja negra no auditable).

El Anexo I. Prototipo SIGLM: casos de uso documenta estos flujos integrados mediante cinco casos de uso con instrucciones detalladas de reproducción, análisis técnico de la ingeniería de *prompt* empleada, y orientación para capturar evidencia visual del comportamiento del sistema. El Anexo II. Código prototipo SIGLM complementa con el código fuente completo comentado.

4.2.4 Tecnologías utilizadas en la PoC

La elección de tecnologías para el prototipo prioriza soberanía de datos, reproducibilidad y facilidad de adopción en entornos de defensa con restricciones de conectividad externa.

LangChain constituye el *framework* de orquestación que articula la interacción entre componentes de IA. Proporciona abstracciones de alto nivel para cadenas RAG, gestión de *prompts* mediante plantillas parametrizables, y *binding* automático de herramientas al LLM mediante *function calling*. Su arquitectura modular permite sustituir componentes subyacentes sin modificar la lógica de orquestación.

Streamlit proporciona la capa de presentación con capacidad de renderizado reactivo y componentes interactivos (file *uploaders*, chat interfaces, *dataframes* editables) con código declarativo

mínimo. La interfaz estructura la pantalla en *sidebar* de gestión documental (*uploads*, configuración, tabla de inventario) y área principal de chat conversacional. El sistema gestiona flujos multi-turno mediante estado persistente, permitiendo que albaranes detectados en PDFs se almacenen temporalmente y el usuario los confirme explícitamente antes de ejecutar transacciones. Esta arquitectura valida tres requisitos de UX para sistemas operativos: visibilidad del estado del sistema, prevención de errores mediante confirmaciones explícitas, y consistencia en patrones de interacción.

Ollama actúa como *runtime* local para modelos de lenguaje, resolviendo el requisito de soberanía de datos. Ejecuta modelos en formato GGUF cuantizados para reducir requisitos de memoria, permitiendo inferencia en hardware doméstico. El prototipo utiliza Mistral (7B parámetros) para generación y mxbai-embed-large (335M parámetros) para *embeddings*, ambos optimizados para español e inglés. La ejecución local de estos modelos demuestra empíricamente la viabilidad de despliegue *on-premise* sin dependencias de APIs externas, cumpliendo requisitos de seguridad en instalaciones militares que operan frecuentemente en áreas con restricciones de comunicaciones.

ChromaDB proporciona base de datos vectorial con persistencia local e indexación HNSW para búsqueda aproximada de vecinos en tiempo logarítmico. Soporta filtrado mediante metadatos y almacena junto a vectores el texto original de cada fragmento, eliminando necesidad de *joins* con almacén documental separado.

PyMuPDF + Tesseract conforman el pipeline de procesamiento documental: PyMuPDF extrae texto nativo de PDFs cuando está disponible con rendimiento superior a bibliotecas tradicionales, mientras que Tesseract OCR se activa selectivamente mediante detección automática de páginas escaneadas. La estrategia de OCR selectivo optimiza tiempo de procesamiento: documentación moderna indexa en segundos mientras que albaranes escaneados pueden requerir minutos, pero se procesan correctamente.

SQLite proporciona motor SQL completo con semántica ACID en archivo embebido, eliminando dependencias de servidor de base de datos, pero manteniendo integridad transaccional. Soporta transacciones **SERIALIZABLE** que previenen *race conditions* en operaciones concurrentes de inventario.

La arquitectura del prototipo valida los flujos de datos fundamentales sin incurrir en complejidad operativa de sistemas distribuidos. El prototipo demuestra que: (1) un LLM puede enrutar consultas complejas a las fuentes de datos apropiadas con precisión adecuada tras *prompt engineering*, (2) búsqueda semántica sobre documentación técnica recupera fragmentos relevantes con *recall* suficiente para anclar respuestas generadas, (3) validaciones de seguridad deterministas se integran transparentemente en flujos orquestados por IA sin comprometer auditabilidad, (4) procesamiento documental heterogéneo se automatiza con precisión aceptable, y (5) el sistema completo puede operar offline en hardware doméstico, validando viabilidad de despliegue *on-premise* en entornos clasificados.

Estas validaciones empíricas proporcionan fundamento técnico para escalar el diseño hacia la arquitectura de microservicios completa descrita en secciones posteriores.

4.3 Estrategia de escalado: hacia una arquitectura operativa

El prototipo desarrollado en el apartado 4.4 demostró empíricamente la viabilidad de las hipótesis fundamentales del trabajo: un LLM puede enrutar consultas a fuentes de datos apropiadas manteniendo separación entre decisiones generativas y deterministas, el procesamiento híbrido de documentos (OCR selectivo + extracción estructurada) automatiza ingesta heterogénea, y la integración RAG+SQL proporciona respuestas ancladas en documentación verificada sin alucinaciones sobre datos transaccionales críticos. Sin embargo, la materialización del prototipo como aplicación monolítica en Python/Streamlit, justificada para validación ágil de conceptos, presenta limitaciones estructurales que impiden su despliegue en un entorno operativo real de defensa.

Las tecnologías utilizadas en el prototipo se resumen en la Tabla 3-1, que justifica el uso de herramientas que a la hora de escalar no serán las adecuadas:

Componente	Tecnología PoC	Producción Equivalente	Justificación Escalado
Arquitectura	Monolítica Python	Microservicios distribuidos	Escalado independiente por componente según demanda real, despliegue continuo sin interrupciones, aislamiento de fallos entre servicios [63].
Orquestación IA	LangChain 0.1.x	LangChain + LangGraph	Workflows complejos con ramificación condicional, gestión de estado distribuido, trazabilidad de decisiones [64].
Modelo Generativo	Ollama/Mistral-7B (local)	Mistral/Llama 70B+ (on-premise)	Mayor capacidad de razonamiento en consultas complejas, contextos extensos (32k+ tokens), soberanía datos clasificados [65].
Embeddings	mxbai-embed-large (local)	multilingual-e5-large	Robustez multilingüe español técnico-militar, rendimiento superior recuperación semántica a escala [40].
Base Vectorial	ChromaDB (persistencia local)	Milvus + HNSW clustering	Índices distribuidos millones de vectores, búsqueda ANN <100ms, replicación y alta disponibilidad [40].
Base Relacional	SQLite (archivo .db)	PostgreSQL HA cluster	Concurrencia ACID 50+ usuarios simultáneos, replicación síncrona, point-in-time recovery [66].
Procesamiento PDF	PyMuPDF + Tesseract	Apache Tika + Tesseract	Detección automática formatos heterogéneos, integración pipelines distribuidos, procesamiento por lotes [67].
Frontend	Streamlit (Python)	Angular + TypeScript	Separación frontend/backend,, actualizaciones independientes del núcleo [68].
Infraestructura	Proceso Python único	Docker + Kubernetes	Orquestación automática réplicas, auto-escalado horizontal, tolerancia a fallos y recuperación automática. [69]

Tabla 4-2. Tecnologías utilizadas en la PoC, frente a las propuestas para la arquitectura en producción.

El paso del prototipo a producción exige incorporar requisitos no funcionales que la PoC no cubre: alta disponibilidad, escalabilidad, observabilidad y seguridad en entornos clasificados (auditoría completa, control de acceso granular y trazabilidad inmutable).

Este apartado presenta la estrategia de escalado que convierte el prototipo validado en una arquitectura distribuida de microservicios, detallando la descomposición funcional, la coordinación transaccional, la resiliencia, la observabilidad y los principios de interfaz para uso operativo militar. Se prioriza la coherencia trazable: cada servicio se deriva de funciones del prototipo o de requisitos operativos, evitando soluciones ad-hoc.

4.3.1 De monolito a microservicios: descomposición funcional.

La descomposición del prototipo monolítico en microservicios sigue el principio de responsabilidad única (*Single Responsibility Principle*, establecido en Cap.3.1.2) aplicado al dominio específico de gestión logística de municiones. Cada microservicio encapsula una capacidad funcional cohesiva derivable de una única fuente de cambio en requisitos de negocio, facilitando evolución independiente y aislamiento de fallos. La Tabla 4-3 mapea funciones del prototipo Python a servicios distribuidos propuestos, evidenciando continuidad lógica entre PoC y arquitectura de producción:

Función Prototipo (Python)	Microservicio Propuesto	Justificación de Descomposición	Tecnología Recomendada
<code>cargar_pdf_hibrido()</code> + <code>analizar_pdf_movimiento_path()</code>	Document Manager Service	Procesamiento OCR y extracción son CPU-intensivos, requieren escalado independiente de consultas	Python + Tesseract + FastAPI
<code>procesar_nuevos_pdfs()</code> (vectorización)	AI Adapter & Orchestration Service	Generación de embeddings demanda GPU, separar de lógica transaccional	Python + LangChain + vLLM
<code>obtener_vectorstore()</code> + <code>retrieval</code>	Vector Database Adapter	Abstrae Milvus, permite migración a otros backends sin afectar consumidores	Python wrapper sobre Milvus API
Tabla inventario (SQLite)	Inventory Transaction Service	Datos transaccionales críticos requieren ACID completo y auditoría	Java/Kotlin + Spring Boot + PostgreSQL
<code>validar_polvorin()</code> + validación de grupos	Safety Rules Engine	Validaciones de seguridad NUNCA deben compartir runtime con componentes generativos	Rust (garantías de memoria) + lógica determinista
<code>ejecutar_con_herramientas()</code> (router)	Saga Orchestrator Service	Coordinación de transacciones distribuidas requiere gestión de estado y compensaciones	Go + Temporal.io
Interfaz Streamlit	API Gateway + Angular SPA	Separación frontend/backend, terminación SSL, rate limiting	Kong Gateway + Angular 17+
Logs en sidebar	Audit & Logging Service	Registros inmutables requieren aislamiento y protección contra modificación	ELK Stack (Elasticsearch + Logstash + Kibana)

Tabla 4-3. Descomposición funcional desde la PoC monolítica a la arquitectura propuesta de microservicios.

La arquitectura de microservicios propuesta distribuye las responsabilidades del prototipo monolítico en servicios especializados que pueden evolucionar y escalar de forma independiente, tal como se propuso en el capítulo 3. Esta descomposición no es arbitraria: responde a los patrones de uso identificados y a los requisitos no funcionales de cada componente.

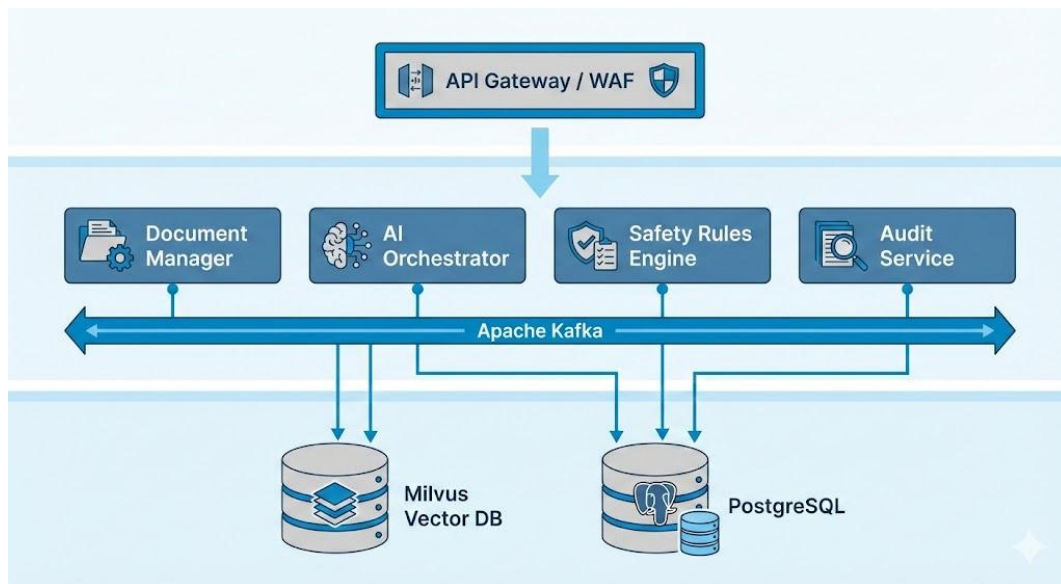


Figura 4-1. Arquitectura simplificada del *backend* del SIGLM mostrando los microservicios principales (elaboración propia)

Los servicios especializados implementan capacidades específicas del sistema SIGLM:

El ***Document Manager Service*** se encarga de todo el ciclo de ingesta documental, pero añadiendo las validaciones de seguridad que el prototipo omite por simplicidad. Cuando un usuario sube un PDF, el servicio no se fía de la extensión del archivo: verifica la firma MIME real para evitar que alguien renombre un ejecutable malicioso como "manual.pdf". Calcula además un hash SHA-256 de cada documento, lo que permite detectar duplicados y mantener trazabilidad forense completa. En entornos de producción, todos los archivos pasan por un escaneo antimalware con ClamAV antes de ser procesados.

Este servicio funcionaría con un modelo asíncrono. En el prototipo, cuando cargas un PDF debes esperar mientras se hace el OCR, se extraen metadatos y se generan embeddings. Esto funciona con un documento, pero imagina procesar cien albaranes simultáneamente: el sistema se saturaría. El Document Manager retorna inmediatamente un identificador único de la tarea y publica un evento en Kafka indicando que hay un documento pendiente. El procesamiento real ocurre en segundo plano, y otros servicios especializados van consumiendo estos eventos según su capacidad.

El ***AI Adapter & Orchestration Service*** encapsula toda la complejidad de trabajar con modelos de IA. En lugar de que cada servicio tenga que saber cómo conectarse a Ollama, generar *embeddings* o construir *prompts*, todos delegan en este adaptador. La ventaja es clara: si mañana queremos cambiar de Mistral a Llama 3, o migrar de Ollama local a un clúster distribuido con vLLM, solo modificamos este servicio. El resto del sistema ni se entera.

Este servicio gestiona varios modelos especializados simultáneamente. Uno para generar *embeddings* multilingües (1024 dimensiones), otro grande para conversación (70B parámetros si el hardware lo permite), y modelos específicos *fine-tuned* sobre formularios militares para extraer datos estructurados. La orquestación RAG completa vive aquí: cuando llega una consulta, el servicio decide primero si necesita búsqueda semántica o basta con SQL determinista. Si es RAG, genera *embeddings*, recupera contexto, lo reordena con un modelo *cross-encoder*, construye el *prompt* final y valida la respuesta contra el contexto recuperado. Si la similitud coseno entre respuesta y contexto es menor de 0.7, rechaza la generación porque probablemente sea una alucinación.

El ***Safety Rules Engine*** materializa la separación crítica entre lógica generativa y validaciones deterministas que discutimos en el capítulo 3. Este servicio nunca consulta a un LLM para decidir si una operación es segura. Ejecuta validaciones puramente algorítmicas: consulta los límites de NEQ configurados para cada polvorín (que mantiene en caché actualizado cada minuto), hace las comparaciones aritméticas necesarias, y retorna una decisión binaria. Si rechaza, incluye un código de error específico que explica qué restricción se violó.

Finalmente, el ***Saga Orchestrator Service*** coordina operaciones complejas que involucran múltiples servicios. En el prototipo, cuando ingestas un documento, una función Python ejecuta toda la secuencia: validación, OCR, extracción, *embeddings*, indexación. Si algo falla a mitad de camino, el estado queda inconsistente. En producción, esto es inaceptable.

El orquestador implementa el patrón Saga usando Temporal.io, un *framework* diseñado específicamente para *workflows* distribuidos. Cuando inicia una ingesta documental, persiste cada paso en PostgreSQL. Si Milvus rechaza la indexación porque está saturado, el Saga ejecuta compensaciones: elimina los *embeddings* ya generados, marca el documento como "procesamiento fallido", y programa un reintento con *backoff* exponencial. El operador puede ver el estado de todos los *workflows* en ejecución a través de una interfaz web y, si es necesario, intervenir manualmente.

4.3.2 Orquestación distribuida (Sagas) y gestión transaccional.

La necesidad de Sagas surge de una limitación fundamental de los sistemas distribuidos. En un monolito, si algo falla durante una transacción compleja, es posible hacer *rollback*. En microservicios, donde cada servicio tiene su propia base de datos, no existe un coordinador transaccional global que pueda revertir todo atómicamente.

El patrón Saga resuelve esto mediante transacciones locales con compensaciones explícitas. La idea es simple: si una operación compleja tiene cinco pasos (T1, T2, T3, T4, T5) y T4 falla, ejecutas las compensaciones en orden inverso (C3, C2, C1) para deshacer los efectos de los pasos que sí se completaron.

la Figura 4-2)

Veamos cómo funciona la Saga de Ingesta Documental (Figura 4-2):

1. El orquestador recibe la solicitud y persiste el estado inicial
2. Llama al *Document Manager* para validar y almacenar el PDF
3. Si se completa, solicita al AI Adapter generar *embeddings*
4. Con los *embeddings* listos, ordena al Vector DB indexarlos
5. Actualiza los metadatos en PostgreSQL
6. Marca la saga como completada y notifica al usuario

Si en el paso 4 Milvus rechaza la indexación (quizá está reiniciándose), el orquestador:

- Le pide al *AI Adapter* que libere los recursos del modelo
- Instruye al *Document Manager* marcar el documento como "en cuarentena"
- Registra el evento de fallo en la base de datos de auditoría
- Programa un reintento automático en 5 minutos

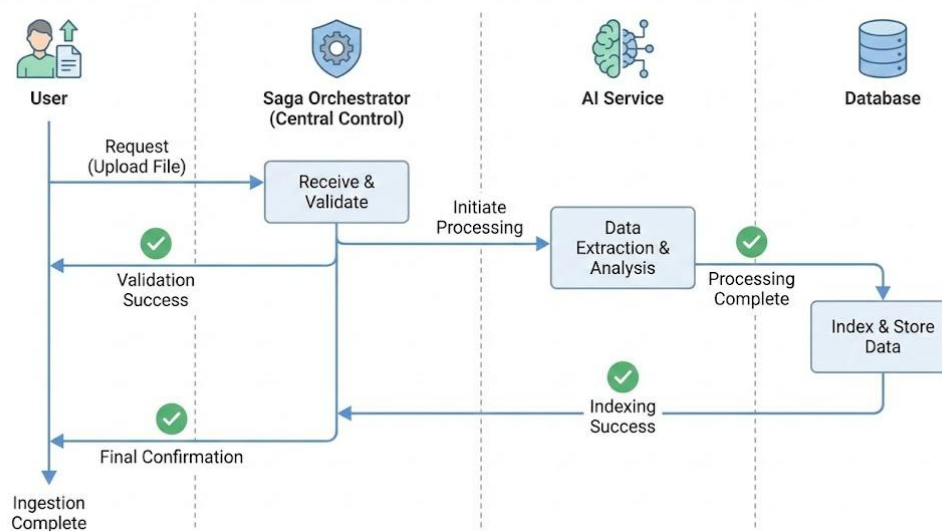


Figura 4-2. Patrón Saga para orquestación de ingesta documental (elaboración propia).

Lo crucial es que las compensaciones son operaciones de negocio, no *rollbacks* técnicos. El log de auditoría no borra los eventos exitosos previos, sino que añade eventos de compensación. Esto mantiene trazabilidad completa: si hay una investigación, podemos reconstruir exactamente qué pasó y cuándo.

La implementación con Temporal.io añade varias ventajas sobre una implementación manual: el estado del *workflow* sobrevive a reinicios del servicio orquestador (está persistido en PostgreSQL), los operadores tienen visibilidad completa de *workflows* en progreso mediante una UI web, y cada paso puede tener políticas de reintento configurables (por ejemplo, "T3 puede reintentar 3 veces con *backoff* exponencial antes de fallar el *workflow* completo").

La **Saga de Movimiento de Munición** coordina una operación aún más delicada. Cuando un usuario quiere mover munición de un polvorín a otro, el flujo involucra validaciones distribuidas en varios servicios. Cada paso es una llamada remota por gRPC o REST. Si alguna validación de seguridad rechaza el movimiento, el *Audit Service* registra el intento para análisis de patrones de uso (quizá alguien está intentando operaciones peligrosas repetidamente).

Esta coordinación mediante Sagas introduce latencia adicional comparada con el monolito. Cada llamada remota suma milisegundos, cada persistencia de estado suma escrituras a disco. Pero a cambio obtenemos tres beneficios críticos:

- Resiliencia ante fallos parciales: si un servicio se cae a mitad de un flujo de trabajo, cuando se recupere puede retomar desde el último *checkpoint* persistido.
- Trazabilidad granular: cada paso se audita independientemente con metadatos completa (*timestamp*, servicio ejecutor, usuario, resultado).
- Contención de fallos: un bug en el servicio de *embeddings* no puede corromper el inventario SQL porque son transacciones locales en bases de datos separadas.

4.3.3 Estrategias de observabilidad y resiliencia.

Cuando pasamos de un prototipo monolítico que corre en un PC a un sistema distribuido con diez servicios corriendo en contenedores, aparece un problema fundamental: ¿cómo saber qué está pasando cuando algo va mal? En el prototipo, si hay un error, es posible consultar la consola de Streamlit y ver el *traceback*. En producción, ese error puede estar enterrado en los logs de un contenedor que ya ni siquiera existe porque Kubernetes lo reinició.

La observabilidad aborda precisamente esto. No se trata solo de guardar logs, sino de instrumentar el sistema para que podamos reconstruir su estado interno en cualquier momento. Implementamos tres pilares fundamentales:

Logging centralizado con el *stack* ELK (Elasticsearch, Logstash, Kibana). Cada servicio emite logs estructurados en JSON con campos estandarizados: *timestamp*, nombre del servicio, un *trace_id* único que identifica toda la operación *end-to-end*, nivel de severidad y el mensaje. Logstash los recoge desde los contenedores Docker, los enriquece con metadatos de infraestructura (en qué pod de Kubernetes estaba corriendo, en qué nodo físico) y los almacena en Elasticsearch. El resultado es que cuando un usuario reporta "mi consulta falló hace dos horas", podemos buscar por su usuario, recuperar el *trace_id*, y ver exactamente qué pasó en cada servicio que participó en esa consulta.

Métricas de aplicación con Prometheus y Grafana. Cada servicio expone un *endpoint /metrics* donde Prometheus puede leer contadores, histogramas y gauges en tiempo real. Las métricas críticas que monitorizamos incluyen la latencia *end-to-end* de consultas RAG (p50, p95, p99), la tasa de validaciones de seguridad que se rechazan, la tasa de procesamiento documental en documentos por hora, y la disponibilidad de cada servicio crítico. Grafana convierte estos números en *dashboards* visuales donde podemos detectar patrones: si vemos que la latencia p95 de RAG se duplica cada noche a las 3 AM, investigamos qué proceso está saturando el sistema.

Trazas distribuidas con Jaeger. Cuando una consulta del usuario atraviesa cinco servicios diferentes, Jaeger nos muestra una línea temporal visual de todo el recorrido: cuánto tardó cada llamada remota, dónde se gastó el tiempo, qué servicio fue el cuello de botella. Esto es invaluable para diagnosticar problemas de rendimiento en sistemas complejos.

La resiliencia, por su parte, implementa patrones que evitan que fallos locales se amplifiquen en cascadas catastróficas. El concepto clave es que los fallos son inevitables, así que diseñamos el sistema para que se degrade de forma controlada en lugar de colapsar completamente.

Los **circuit breakers** protegen servicios de sobrecarga. Si el *AI Adapter* detecta que Milvus está respondiendo muy lento o rechazando conexiones, abre el circuito: durante 30 segundos, no le envía más peticiones. Esto le da tiempo a Milvus para recuperarse sin ser bombardeado con reintentos. Después de esos 30 segundos, envía una petición de prueba; si funciona, cierra el circuito y vuelve a la normalidad. Si falla, permanece abierto otros 30 segundos.

La **degradación controlada** permite que el sistema siga funcionando con capacidades reducidas. Si Milvus falla completamente, el *AI Adapter* degrada automáticamente la búsqueda semántica a búsqueda por keywords usando Elasticsearch. La calidad de las respuestas baja (no captura sinónimos ni relaciones semánticas) pero el chatbot sigue operativo. Si incluso el *AI Adapter* completo falla, el sistema entra en "modo solo-inventario": los usuarios pueden consultar stock SQL, ejecutar movimientos de munición con validaciones deterministas, pero no pueden hacer consultas sobre normativa. Cada nivel de degradación se notifica claramente en la interfaz.

Los **reintentos con backoff exponencial** manejan fallos transitorios. Si una llamada gRPC al Safety Rules Engine falla con *timeout*, no reintentamos inmediatamente. Esperamos 1 segundo, luego 2, luego 4. Esto resuelve problemas por fluctuaciones momentáneas de red sin saturar un servicio que ya está lento con reintentos agresivos.

Finalmente, **Kubernetes gestiona la auto-recuperación**. Cada pod expone dos *endpoints de health*: */health/liveness* indica que el servicio está vivo y respondiendo, */health/readiness* indica que está listo para recibir tráfico de producción. Si *liveness* falla tres veces seguidas, Kubernetes mata el pod y levanta uno nuevo. Si *readiness* falla, Kubernetes lo saca del balanceador de carga hasta que se recupere, pero no lo mata (es posible que solo esté temporalmente sobrecargado).

La inversión en observabilidad y resiliencia permite alcanzar alta disponibilidad incluso cuando componentes individuales fallan frecuentemente. Esto es especialmente crítico en entornos de defensa

donde el sistema puede estar operando en condiciones adversas: redes satelitales con latencia alta, hardware envejecido, operadores sin conexión estable a internet.

4.3.4 Interfaces de operación y experiencia de usuario

La interfaz de usuario es donde todas las capacidades técnicas del sistema se convierten en valor real para los operadores. No importa qué tan sofisticada sea tu arquitectura de microservicios si el cabo que tiene que registrar un movimiento de munición a las 2 AM durante un ejercicio se encuentra con una interfaz confusa.

Decidimos implementar el *frontend* como una Single Page Application (SPA) con Angular 17+, alejándonos del Streamlit del prototipo. Esta decisión responde a requisitos concretos: necesitamos tipado estricto, arquitectura basada en componentes que facilite testing unitario, y gestión de estado complejo mediante NgRx para mantener consistencia en una aplicación con historial de chat, documentos procesándose en segundo plano y notificaciones de sistema.

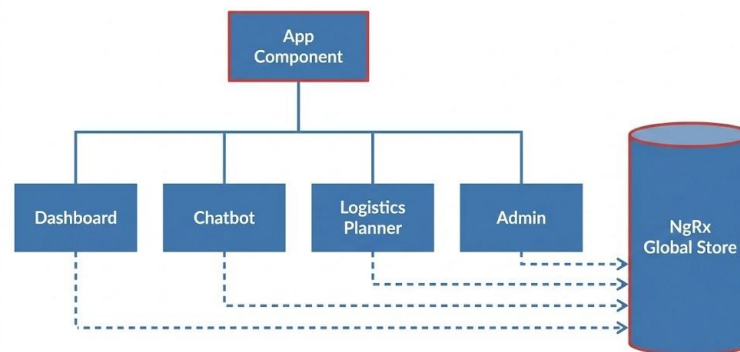


Figura 4-3. Arquitectura modular del *frontend* SIGLM basada en Angular (elaboración propia).

La aplicación se estructura en **módulos funcionales** (Figura 4-3). El módulo de gestión documental proporciona *drag-and-drop* para subir PDFs con vista previa y barra de progreso durante el procesamiento asíncrono. El módulo de consultas RAG implementa el chatbot con tres características clave: el historial de conversación se persiste localmente en IndexedDB (sobrevive a cierres del navegador), el usuario puede seleccionar cualquier fragmento de una respuesta y hacer click derecho / "Ver fuente" para que el sistema navegue al documento original y resalte ese párrafo exacto, y hay un panel lateral que muestra los documentos que el RAG recuperó durante la búsqueda (transparencia algorítmica).

El módulo de administración, reservado para supervisores y comandantes, gestiona usuarios y permisos RBAC, configuración de sistema (umbrales de alertas, parámetros de los modelos), y auditoría de operaciones con filtros avanzados.

La gestión de estado con NgRx mantiene consistencia en esta complejidad. El store global incluye: autenticación (token JWT, perfil, permisos), historial de chat (array de mensajes con metadatos), estado de documentos en procesamiento, y notificaciones del sistema. Cuando un documento termina de procesarse, el *backend* envía un evento WebSocket, el *reducer* de NgRx actualiza el estado, y todos los componentes que dependan de esa información se re-renderizan automáticamente.

La comunicación con el *backend* se abstrae usando servicios Angular especializados: *DocumentService*, *ChatService*, *InventoryService*, *AuthService*. Esto permite simular fácilmente durante las pruebas y cambiar implementaciones *backend* sin tocar componentes UI.

En cuanto a seguridad en cliente, implementamos múltiples capas de defensa, sabiendo que nunca sustituyen las validaciones del servidor. Los *guardrails* de Angular protegen rutas según roles, pero el *backend* siempre valida autorización y rechaza con HTTP 403 cualquier petición no autorizada.

Finalmente, la interfaz se adapta al rol del usuario. Un operador ve un panel simplificado: consultar inventario, añadir/extraer munición con validaciones automáticas, cargar documentación. Un supervisor ve capacidades adicionales: aprobar operaciones que requieren confirmación manual, revisar alertas de sistema, consultar auditoría de su unidad. El usuario administrador accede al panel completo: configurar sistema, gestionar usuarios, auditoría global, reportes ejecutivos.

Esta separación cumple dos objetivos: previene sobrecarga cognitiva (operadores no ven opciones irrelevantes) y reduce superficie de error (un operador no puede accidentalmente cambiar configuración crítica del sistema porque esas opciones ni siquiera están en su interfaz).

4.4 Marco de evaluación: metodología y pruebas

El prototipo SIGLM desarrollado en el apartado 4.2 y la arquitectura distribuida descrita en el 4.3 requieren validación sistemática que trascienda verificación de funcionalidad básica para evaluar corrección técnica, utilidad operativa y conformidad con requisitos de seguridad críticos. La validación de sistemas de IA para defensa enfrenta desafíos únicos: ausencia de corpus de evaluación públicos (documentación operativa clasificada), necesidad de cuantificar riesgos de alucinaciones que pueden conducir a decisiones incorrectas, y requisito de demostrar cumplimiento normativo ante auditorías de organismos de defensa y ciberseguridad.

Este apartado establece marco metodológico de evaluación estructurado en tres dimensiones complementarias: métricas técnicas de calidad RAG que validan corrección de recuperación y generación, plan de pruebas funcionales y de seguridad que verifica comportamiento del sistema en escenarios operativo, y análisis de riesgos con KPIs de impacto operativo que miden valor del sistema en términos del negocio. La metodología adopta enfoque conservador propio de sistemas críticos: umbrales de aceptación exigentes, validación multinivel (automatizada + humana experta), y caracterización explícita de limitaciones y modos de fallo conocidos.

Es importante aclarar que métricas, umbrales y procedimientos descritos constituyen propuesta metodológica para eventual implementación operativa, no resultados de evaluación real. Los valores numéricos presentados en tablas son objetivos propuestos establecidos mediante análisis de requisitos del dominio de defensa y referencias a sistemas similares en literatura académica, sirviendo como marco de referencia cuantitativo que guiaría evaluación durante fases de desarrollo futuras.

4.4.1 Métricas de recuperación y generación

La evaluación de sistemas RAG requiere descomposición en métricas específicas para componente de recuperación (*capability* del sistema para encontrar documentos relevantes) y componente de generación (fidelidad de respuesta al contexto recuperado). Esta separación es crítica para diagnóstico de fallos: respuestas incorrectas pueden originarse en recuperación de contexto irrelevante o en alucinaciones del modelo de lenguaje, requiriendo intervenciones correctivas radicalmente diferentes según componente defectuoso.

El *framework* de evaluación adoptado sigue metodología RAGas (Retrieval Augmented Generation Assessment), especializada en sistemas RAG con métricas validadas académicamente que han demostrado correlación con juicio humano experto [70]. La Tabla 4-4 consolida métricas clave, su interpretación y umbrales propuestos para SIGLM:

Métrica	Fórmula / Definición	Umbral propuesto	Justificación Dominio
RECUPERACIÓN			
Context Recall	Proporción de afirmaciones en <i>ground truth</i> atribuibles a contexto recuperado	≥ 0.85	Omitir normativa crítica es inaceptable
Context Precision	Proporción de fragmentos recuperados que son relevantes	≥ 0.70	Evitar ruido que confunda LLM
MRR (Mean Reciprocal Rank)	Promedio de 1/posición del primer hit relevante	≥ 0.75	Usuario típicamente revisa top-3
NDCG@k (k=5)	Normalized Discounted Cumulative Gain	≥ 0.80	Ranking de calidad importa para UX
GENERACIÓN			
Faithfulness	Proporción de afirmaciones en respuesta verificables en contexto	≥ 0.90	Cero tolerancia a alucinaciones normativas
Answer Relevancy	Similitud semántica entre pregunta y respuesta	≥ 0.80	Evitar evasivas genéricas
Context Relevancy	Proporción de contexto usado en respuesta final	≥ 0.60	Eficiencia en uso de contexto recuperado

Tabla 4-4. Métricas clave de recuperación y generación

4.4.1.1 Métricas de Recuperación

Context Recall (exhaustividad) mide si contexto recuperado contiene **toda** información necesaria para responder completamente. Calculada como proporción de afirmaciones en respuesta *ground truth* atribuibles a contexto, valor 1.0 indica completitud perfecta. Un umbral ≥ 0.85 refleja que, en dominio de municiones, omitir información crítica (restricción de seguridad, incompatibilidad química) puede tener consecuencias catastróficas. Valores inferiores señalan que estrategia de recuperación no captura todos fragmentos relevantes, requiriendo ajustes en *k* (número documentos recuperados), umbrales de similitud, o *query expansion* mediante reformulación de consulta en variantes semánticas.

Context Precision evalúa proporción de fragmentos recuperados realmente relevantes versus ruido. Exceso de información irrelevante diluye señal en contexto extenso que LLM debe procesar, aumentando latencia y riesgo de confusión del modelo. Un umbral ≥ 0.70 balancea exhaustividad (*recall* alto requiere recuperar más documentos, aumentando ruido potencial) con calidad (fragmentos irrelevantes degradan generación). Valores bajos sugieren que modelo de *embeddings* no discrimina relevancia adecuadamente, requiriendo *fine-tuning* con ejemplos específicos del dominio o implementación de reranking con *cross-encoders* que reordenan resultados iniciales mediante modelos más sofisticados.

Mean Reciprocal Rank (MRR) mide posición del primer documento relevante en ranking. Usuarios típicamente revisan solo top-3 resultados; documento crítico en posición 8 es funcionalmente invisible. Contribución a MRR: posición 1 $\rightarrow$ 1.0, posición 2 $\rightarrow$ 0.5, posición 3 $\rightarrow$ 0.33. Un umbral ≥ 0.75 implica que, en promedio, primer hit relevante aparece en top-2. MRR bajo indica que *embeddings* capturan relevancia, pero ranking es subóptimo, señalando necesidad de reranking o ajuste de parámetros de búsqueda.

NDCG@k (Normalized Discounted Cumulative Gain) generaliza MRR considerando todos documentos relevantes en top-k con ponderación logarítmica que penaliza posiciones inferiores. $NDCG@5 \geq 0.80$ garantiza que documentos altamente relevantes aparecen consistentemente en primeras posiciones, crítico para UX donde usuarios deciden basados en primeros resultados mostrados. Normalización permite comparación entre *queries* con diferente número de documentos relevantes disponibles.

4.4.1.2 Métricas de Generación

Faithfulness (fidelidad) es la métrica más crítica en sistemas que operan en dominios regulados: cuantifica la proporción de afirmaciones presentes en la respuesta generada que pueden verificarse directamente en el contexto recuperado. Su cálculo se basa en tres pasos: (1) extracción de afirmaciones atómicas de la respuesta, (2) verificación de cada una frente al contexto disponible y (3) cómputo de la proporción de afirmaciones verificables. Un umbral ≥ 0.90 refleja una tolerancia nula a las alucinaciones en materia normativa: cualquier afirmación sobre compatibilidad química, límites de NEQ o procedimientos de almacenamiento debe estar explícitamente respaldada por la documentación recuperada. Un valor de fidelidad < 0.90 indica que el diseño del *prompt* no basta para anclar el modelo al contexto, lo que exige plantillas más restrictivas, ejemplos de referencia que muestren respuestas correctamente fundamentadas o mecanismos de entrenamiento que penalicen la generación de información no presente en el contexto.

Answer Relevancy mide si respuesta generada realmente responde pregunta formulada versus proporcionar información genérica o evasiva. Calculada por similitud semántica (típicamente coseno de *embeddings*) entre pregunta original y respuesta. Valores bajos (< 0.70) indican que modelo evade preguntas específicas con generalidades ("depende de varios factores...", "consulta normativa aplicable...") sin proporcionar respuesta concreta cuando contexto la contiene. Un umbral ≥ 0.80 asegura respuestas directas y accionables.

Context Relevancy evalúa eficiencia en uso de contexto: proporción de contexto recuperado que realmente se utiliza en respuesta final. Valores muy bajos (< 0.40) sugieren que sistema recupera excesivo ruido que modelo debe filtrar. Valores extremadamente altos (> 0.90) pueden indicar que sistema no sintetiza ni abstrae, sino que copia textualmente fragmentos, limitando valor añadido. Umbral ≥ 0.60 balancea síntesis inteligente con uso efectivo de información recuperada.

4.4.1.3 Evaluación Humana Complementaria

Las métricas automáticas proporcionan escalabilidad para evaluar miles de consultas, pero la evaluación humana por expertos del dominio resulta insustituible para validar aspectos sutiles de calidad. Se define un protocolo de evaluación humana donde artificieros y oficiales de logística revisan muestras aleatorias de respuestas del sistema (típicamente 10-15% del conjunto test) clasificándolas según múltiples dimensiones:

- **Correcto y útil:** Respuesta correcta, completa y directamente aplicable.
- **Correcto pero mejorable:** Técnicamente correcto, pero podría ser más claro.
- **Parcialmente correcto:** Contiene información correcta pero también omisiones o imprecisiones.
- **Incorrecto pero seguro:** Respuesta incorrecta pero el error no conduciría a decisiones peligrosas.
- **Incorrecto y peligroso:** Respuesta que podría conducir a configuraciones inseguras.

Esta última categoría, "incorrecto y peligroso", tiene peso desproporcionado en la evaluación global: incluso una incidencia baja de respuestas peligrosas ($> 1\%$) descalifica el sistema para despliegue operativo hasta que se identifiquen y remedien las causas raíz.

4.4.2 Plan de pruebas funcionales y de seguridad

La validación de SIGLM requiere estrategia de pruebas multinivel que progrese desde validación técnica en laboratorio hasta piloto operativo con usuarios reales, con criterios explícitos en cada fase que autorizan progresión a siguiente nivel de exposición. El enfoque incremental mitiga riesgos: problemas detectados en fases tempranas se remedian antes de escalar complejidad y visibilidad del sistema.

Fase 1: Validación Técnica (Alpha Testing)

La primera fase se centraría en validación técnica exhaustiva en entorno de laboratorio completamente aislado de redes operativas. Esta fase ejecutaría bancos de pruebas.

Actividades principales:

- Ejecución de baterías de pruebas automatizadas sobre miles de consultas test, calculando métricas RAGas (*Context Recall, Precision, Faithfulness, Answer Relevance*)
- Optimización de hiperparámetros
- Pruebas de carga concurrente y validar que el sistema mantiene latencias aceptables bajo estrés
- Pruebas de penetración conducidas por equipo de *red team* para identificar vulnerabilidades.

Fase 2: Validación Funcional con Usuarios Piloto (Beta Testing)

La segunda fase incorporaría evaluación humana intensiva por expertos del dominio que representan los usuarios finales objetivo del sistema. Esta fase resultaría crítica para validar que el sistema no solo es técnicamente correcto según métricas automáticas, sino que es operacionalmente útil según el juicio de profesionales que trabajarán con él.

Los expertos formularían consultas al sistema como lo harían en operaciones reales, y evaluarían las respuestas según múltiples dimensiones:

- Correcto y directamente aplicable
- Correcto pero mejorable
- Parcialmente correcto
- Incorrecto pero seguro
- Incorrecto y potencialmente peligroso

Este *feedback* cualitativo se utilizaría para múltiples propósitos: identificar patrones de consultas donde el sistema exhibe comportamiento subóptimo, detectar fallos de razonamiento, y potencialmente reentrenar o ajustar modelos con ejemplos del dominio específico donde el sistema falló.

Ejemplo de criterios de salida: Al menos el 85% de respuestas deberían clasificarse como "Correcto y directamente aplicable". La proporción de respuestas "Incorrectas y potencialmente peligrosas" debería ser inferior al 1%.

Fase 3: Piloto Operativo Limitado

La tercera fase desplegaría el sistema en un entorno operativo real, pero con alcance limitado a una única unidad piloto. Esta fase valdría para evaluar performance del sistema en condiciones realistas que no pueden replicarse en laboratorio y exposición a casos que no fueron anticipados durante diseño.

Durante la fase piloto inicial, el sistema operaría en "modo sombra" o "modo asistente": los usuarios podrían consultar el sistema y recibir respuestas, pero el sistema no tendría autorización para ejecutar acciones de modificación de estado. Las respuestas del sistema se presentarían como sugerencias que el usuario debe validar y ejecutar manualmente usando procedimientos formales existentes. Esta aproximación permite recopilar datos de uso real mientras minimiza riesgos de que fallos del sistema impacten operaciones.

Durante el modo sombra, se instrumentaría exhaustivamente el sistema para registrar: todas las consultas formuladas por usuarios reales, todas las respuestas generadas por el sistema, decisiones finales tomadas por usuarios (si siguieron la sugerencia del sistema o la rechazaron), y *feedback* explícito de usuarios sobre utilidad de respuestas. Esto permitiría identificar discrepancias entre comportamiento del sistema en test contra uso real, detectar tipos de consultas frecuentes no representadas adecuadamente en datos de evaluación, y cuantificar KPIs operativos.

4.4.3 Análisis de riesgos y KPIs de impacto operativo

Despliegue de sistemas de IA en entornos críticos requiere análisis exhaustivo de riesgos técnicos, operativos y de cumplimiento, que cuantifican impacto del sistema en objetivos del negocio más allá de métricas técnicas de calidad.

4.4.3.1 Análisis de Riesgos

Dado el carácter innovador de introducir IA en procesos críticos de defensa, se identifican riesgos cuya materialización podría comprometer el éxito del proyecto se resumen en la Tabla 4-5:

Riesgo	Impacto	Proba- bilidad	Estrategia de Mitigación
Alucinaciones en datos de seguridad críticos	Crítico	Media	Implementación de un motor determinista de reglas de seguridad que valida todas las salidas. El LLM solo sugiere y explica; el cálculo y la autorización son deterministas. Monitorización continua en producción.
Rechazo organizacional por usuarios finales	Alto	Alta	Participación temprana de personal operativo para incorporar <i>feedback</i> .. El sistema se presenta como apoyo, sin sustituir el juicio ni la autoridad humana. Despliegue acompañado de gestión del cambio.
Obsolescencia tecnológica de modelos de IA	Medio	Alta	Arquitectura modular con patrón Adapter que permite sustituir el LLM sin cambios en la aplicación. Evaluación continua de nuevos modelos para mejoras.
Cuellos de botella en hardware <i>on-premise</i>	Alto	Media	Uso de colas asíncronas con Kafka para desacoplar ingesta y procesamiento. Sobredimensionamiento de hardware del 40–50% para absorber picos.
Degradación de performance con volumen creciente de documentación	Medio	Media	Indexación vectorial escalable (HNSW, IVF) para grandes volúmenes documentales. Particionado por clasificación de seguridad, reindexación periódica.
Exposición de información clasificada mediante exfiltración	Crítico	Baja	Despliegue <i>on-premise</i> sin conexión a internet. Cifrado en reposo y en tránsito, auditoría inmutable de accesos y monitorización de salidas para prevenir divulgación de información sensible.
Incompatibilidad con actualizaciones de normativas militares	Medio	Media	Pipeline automatizado para ingesta y actualización rápida de normativas. Versionado que garantiza trazabilidad de las respuestas y alertas a usuarios ante cambios relevantes en su ámbito de responsabilidad.

Tabla 4-5. Matriz de Riesgos Críticos y Estrategias de Mitigación

4.4.3.2 KPIs de Impacto Operativo

Más allá de métricas técnicas, éxito del sistema se mide por impacto en objetivos del negocio. KPIs propuestos con targets cuantificados:

KPI	Medición	Baseline (Método Manual)	Objetivo SIGLM	Impacto Estimado
Tiempo de resolución consulta simple	Minutos promedio	8-12 min	< 3 min	70-75% reducción
Tiempo de resolución consulta compleja	Minutos promedio	25-35 min	< 10 min	65-70% reducción
Tasa de detección errores seguridad	% detectado	93% (humano)	≥ 95%	+2pp detección
Falsos negativos (crítico)	% no detectado	7%	< 3%	-4pp riesgo
Latencia P95 consulta estándar	Segundos	N/A	< 5s	Experiencia fluida
Adopción de usuarios (3 meses)	% usuarios activos semanales	N/A	≥ 70%	Meta de adopción
Consultas por sesión	Promedio	N/A	≥ 5	Indicador de utilidad
NPS (satisfacción)	Score -100 a +100	N/A	≥ 40	Alta satisfacción

Tabla 4-6. KPIs Operativos y Métricas de Impacto

Nota: KPIs propuestos y estimaciones de impacto basadas en análisis de procesos operativos actuales y extrapolación de mejoras observadas en literatura académica sobre sistemas de asistencia IA en dominios especializados. Los valores baseline son estimaciones de tiempos típicos en procesos manuales. Los objetivos SIGLM representan metas que se establecerían para un desarrollo real. Estos valores son ILUSTRATIVOS y tendrían que validarse empíricamente durante implementación y piloto operativo.

4.4.3.3 Reducción del Tiempo de Resolución

Se define como la comparativa del tiempo medio necesario para resolver una consulta técnica compleja utilizando el sistema propuesto frente al método manual tradicional, que típicamente implica la búsqueda en documentación física o digital sin asistencia de IA.

La metodología de medición propuesta se fundamenta en la selección de un conjunto predeterminado de consultas representativas que abarquen diversos niveles de complejidad. El diseño experimental contempla que personal experto resuelva el cincuenta por ciento de las consultas usando métodos tradicionales, recurriendo a la búsqueda manual en archivos y a la consulta con compañeros, mientras que el cincuenta por ciento restantes, de naturaleza equivalente, se resolvería mediante el SIGLM. El registro temporal se iniciaría en el momento de formulación de la pregunta y concluiría cuando se obtuviera una respuesta completa validada por personal experto, permitiendo así una comparación directa entre ambas aproximaciones.

4.4.3.4 Tasa de Detección de Errores de Seguridad

Esta métrica se define como el porcentaje de incompatibilidades, violaciones de límites NEQ o errores de configuración detectados por el sistema en escenarios de simulación, en comparación con la tasa de detección humana que serviría como línea base de referencia.

La metodología de evaluación propuesta contempla la creación de un conjunto predeterminado de escenarios de configuración de almacenamiento o transporte de municiones, donde el sesenta por ciento correspondería a configuraciones válidas y el cuarenta por ciento restantes contendría violaciones deliberadas de restricciones de seguridad con diversos grados de sutileza. El SIGLM analizaría la totalidad de los escenarios e identificaría aquellos que presenta características problemáticas. De forma paralela, un panel de expertos humanos revisaría los mismos escenarios, permitiendo establecer una comparativa rigurosa entre ambas aproximaciones.

El análisis de resultados se articularía en torno a tres categorías principales. Los verdaderos positivos representarían aquellas configuraciones inválidas correctamente identificadas por el sistema. Los falsos positivos corresponderían a configuraciones válidas incorrectamente señaladas como problemáticas, generando trabajo adicional de validación, aunque sin comprometer la seguridad. Los falsos negativos, considerados críticos desde una perspectiva de seguridad, constituirían configuraciones inválidas que el sistema no habría detectado, permitiendo potencialmente situaciones de riesgo.

4.4.3.5 Latencia End-to-End del Sistema

La latencia *end-to-end* constituye una métrica esencial para evaluar la experiencia de usuario y la viabilidad operativa del sistema. Se define como el tiempo total transcurrido desde que el usuario envía una consulta hasta que la respuesta completa se renderiza en su pantalla, abarcando la latencia de red, el procesamiento en el *backend* y la presentación en el *frontend*.

La metodología de medición propuesta requiere instrumentación que registre marcas temporales en puntos críticos del pipeline de procesamiento. Específicamente, se registrarían eventos en la recepción de la consulta en el API Gateway, el inicio y finalización de la recuperación vectorial, el inicio y finalización de la generación por parte del LLM, el envío de la respuesta y el renderizado final en el *frontend*. A partir de estos registros se calcularían los percentiles P50, P90, P95 y P99 de latencia total sobre conjuntos de miles de consultas reales, proporcionando una caracterización estadística robusta del comportamiento del sistema.

5 CONCLUSIONES

La presente memoria ha desarrollado el diseño de una arquitectura de gestión documental basada en IA, ha validado su viabilidad técnica mediante un prototipo funcional, y ha establecido las pautas para su escalado a entornos de producción en organizaciones de alta criticidad. El trabajo integra sistemas de IA generativa con bases de datos relacionales, habituales en la documentación organizativa, demostrando empíricamente que esta hibridación es técnicamente viable y operativamente efectiva.

Se ha propuesto una arquitectura de referencia y se ha validado usando un caso de estudio exigente en Defensa: el Sistema Inteligente de Gestión Logística de Municiones (SIGLM) para la Armada. La construcción de un prototipo funcional ha permitido demostrar empíricamente los tres pilares conceptuales del diseño: la capacidad de un LLM para actuar como orquestador inteligente de consultas heterogéneas, la efectividad del patrón híbrido RAG-SQL en el que consultas normativas se resuelven semánticamente mientras que operaciones de inventario y restricciones críticas se ejecutan determinísticamente, y la viabilidad del procesamiento OCR integrado en flujos automatizados de ingesta documental.

El trabajo concluye que, en entornos de alta criticidad, la combinación de procesamiento de lenguaje natural, búsqueda semántica y RAG permite convertir un repositorio documental "pasivo" en una capacidad de consulta asistida, más accesible y orientada a la toma de decisiones. Esta aproximación es especialmente relevante cuando el conocimiento crítico está distribuido en documentación heterogénea (normativa, manuales, informes, registros históricos), y cuando las consultas reales requieren contexto, trazabilidad y explicación, no solo coincidencia de palabras.

En el ámbito de la logística de municiones confluyen tres condicionantes que elevan el umbral de exigencia:

- Heterogeneidad documental (multiformato, multilingüe y con legado en papel).
- Soberanía del dato y seguridad (despliegue *on-premise*, control de acceso multinivel, auditoría y trazabilidad).
- Complejidad operativa y restricciones críticas (compatibilidades, NEQ/QD y reglas de seguridad que no admiten ambigüedad).

El prototipo desarrollado demuestra que es posible abordar estos tres desafíos simultáneamente en una arquitectura coherente.

La paradoja de la IA en entornos de Defensa es que la misión exige certeza y verificabilidad (determinismo), mientras que la IA generativa opera por probabilidad. En consecuencia, el valor real de este trabajo no reside en "usar IA" como fin en sí mismo, sino en la arquitectura de contención validada empíricamente: un esquema híbrido donde la IA aporta comprensión y síntesis, pero queda encapsulada

por módulos deterministas que aplican reglas "no negociables" y validaciones verificables. El prototipo SIGLM implementa esta separación mediante funciones especializadas donde el componente `validar_polvorin()` ejecuta verificaciones críticas de seguridad de forma completamente determinista, mientras que el enrutamiento inteligente usando LLM dirige consultas semánticas hacia la base vectorial y operaciones transaccionales hacia la base relacional. Esta arquitectura de contención resulta exportable a otros dominios críticos (sanidad, energía, nuclear) donde la creatividad del modelo debe estar subordinada a la realidad de los datos y a la normativa.

5.1 Contribuciones principales

Este trabajo aporta varias contribuciones al campo de la gestión documental asistida por IA en entornos críticos, que van desde la fundamentación teórica hasta la validación empírica mediante prototipo funcional.

1. **Fundamentación y criterios de aplicabilidad.** Se ha realizado una revisión sistemática de las técnicas actuales de IA aplicables a gestión documental, pero no como un mero estado del arte descriptivo. El análisis identifica las condiciones específicas que determinan el éxito o fracaso de estas tecnologías: calidad del corpus documental, necesidades de gobernanza, requisitos de seguridad, y protocolos de evaluación continua. Esta revisión establece un marco de referencia práctico para decidir cuándo adoptar soluciones basadas en IA en contextos donde el error tiene consecuencias graves.
2. **Arquitectura de referencia modular para alta criticidad:** Se propone una arquitectura orientada a microservicios con separación clara entre:
 - **Capa IA** (ingesta, enriquecimiento, *embeddings*, recuperación híbrida, RAG),
 - **Capa transaccional** (datos estructurados, inventarios, trazabilidad),
 - **Capa de seguridad y auditoría** (identidad, autorización, logging, control de cambios),
 - **Capa determinista de seguridad** (validaciones verificables).

Esta separación arquitectónica constituye un patrón reutilizable para otros dominios críticos.

3. **Prototipo funcional y validación empírica.** Más allá de la propuesta arquitectónica, se ha desarrollado e implementado un sistema completamente operativo que demuestra la viabilidad práctica del enfoque. Este prototipo (SIGLM) no es una prueba de concepto simplificada, sino una aplicación funcional construida con Python, Streamlit, LangChain y Ollama que integra todos los componentes del diseño: OCR selectivo inteligente, base vectorial (ChromaDB), base relacional (SQLite), orquestación con LLM local, y validación determinista de restricciones.

El desarrollo del prototipo ha sido fundamental para validar hipótesis que en el diseño teórico parecían razonables pero que en la práctica presentaban desafíos no anticipados. Por ejemplo, la gestión de PDFs escaneados de baja calidad, la normalización de nomenclaturas inconsistentes entre documentos (proyectil 155mm vs. proyectil de 155 milímetros), o la calibración de umbrales de similitud semántica para búsquedas. Estos aprendizajes informan directamente la hoja de ruta de escalado propuesta en el capítulo 5.

4. **Hoja de ruta de escalado validada.** A partir de los resultados obtenidos con el prototipo, se establece una transición documentada y sistemática desde la arquitectura monolítica inicial hacia una solución distribuida basada en microservicios. Esta hoja de ruta (Tabla 5-1) no es especulativa: cada decisión de diseño está fundamentada en limitaciones concretas observadas durante el desarrollo. Se detallan aspectos como separación de responsabilidades entre servicios, estrategias de caché, gestión de estado y protocolos de comunicación.

Componente (Cap. 3)	Prototipo (Cap.4.2)	Producción (Cap. 4.3)	Escalado
Arquitectura	Monolítica Python	Microservicios distribuidos	Escalado independiente por componente por demanda real, disponibilidad continua y aislamiento de fallos entre servicios
Procesamiento PLN	LangChain + Ollama local	AI Adapter + vLLM cluster GPU	Múltiples modelos, load balancing
Base Vectorial	ChromaDB local filesystem	Milvus distribuido con sharding	Millones vectores, HA, backups
Base Relacional	SQLite mono-usuario	PostgreSQL cluster ACID	50+ usuarios concurrentes
Ingesta Documental	cargar_pdf_hibrido() Python secuencial	Document Manager Service + Kafka asíncrono	Procesamiento paralelo batches
Seguridad	Sin autenticación (local)	API Gateway + Keycloak RBAC	SSO, roles, auditoría accesos
Auditoría	Logs Streamlit sidebar	ELK Stack logs inmutables	Registro firmado, retención 5+ años
Orquestación	Función Python síncrona	Temporal.io Sagas distribuidas	Compensaciones, workflows duraderos
Validación Determinista	validar_polvorin() Python	Safety Rules Engine (Rust)	<10ms SLA, aislamiento de IA
Interfaz	Streamlit monolítico	Angular SPA + API Gateway	Frontend desacoplado, multi-dispositivo

Tabla 5-1. Hoja de ruta desde componente conceptual, a prototipo y propuesta de escalado a producción.

Nota: La tabla evidencia cómo cada componente teórico del Cap. 3 se materializa primero en implementación simple del prototipo (Cap.4.2) validando viabilidad conceptual, y luego escala a arquitectura robusta de producción (Cap. 4.3) que cumple requisitos no funcionales de disponibilidad, seguridad y escalabilidad.

5. **Metodología de validación para sistemas híbridos IA-deterministas.** Uno de los retos principales al combinar componentes de IA con sistemas transaccionales tradicionales es establecer criterios claros de cuándo confiar en cada uno. Este trabajo propone una metodología concreta para validar sistemas híbridos en entornos críticos, que incluye: criterios de decisión para enrutar consultas (¿cuándo usar RAG vs. SQL?), métricas específicas de precisión en recuperación documental, protocolos de validación cruzada entre ambas fuentes de datos, y estrategias de auditoría que permitan rastrear el origen de cada dato en una respuesta compuesta.

Esta metodología es generalizable. Los principios aplicados aquí al dominio de municiones se pueden adaptar a otros contextos críticos o a otros con menores exigencias.

5.2 Limitaciones y consideraciones

Este trabajo presenta limitaciones importantes que conviene reconocer para interpretar correctamente las conclusiones.

Alcance del prototipo y validación con corpus limitado. El prototipo funciona con un conjunto reducido y controlado de documentos. No hemos probado cómo se comporta con miles de PDFs heterogéneos, documentación en múltiples idiomas, o registros históricos de décadas de antigüedad. Esta es quizás la limitación más relevante: el prototipo demuestra viabilidad técnica, pero no podemos extrapolar su rendimiento a un entorno operativo real sin más pruebas.

Tampoco hemos validado el sistema con usuarios finales expertos en logística de municiones. Su *feedback* sería fundamental para evaluar usabilidad, detectar casos límite que solo emergen en uso real, y calibrar los umbrales de confianza del sistema. Esta validación queda pendiente para trabajo futuro.

Entorno de ejecución simplificado. El prototipo ejecuta en un único proceso Python sobre hardware convencional. No replica las condiciones de un sistema de producción: cientos de usuarios simultáneos, requisitos de disponibilidad 24/7, seguridad multinivel, o tolerancia a fallos. Aspectos críticos como autenticación, auditoría inmutable, cifrado de datos, o control de acceso basado en roles están especificados en la arquitectura propuesta pero no implementados en el prototipo.

Dependencia de la calidad del dato. A pesar de la arquitectura de contención, persiste el riesgo de respuestas incorrectas si los controles fallan. El prototipo demuestra que es posible separar decisiones críticas (que van a validación determinista) de consultas informativas (que toleran incertidumbre), pero esta separación depende del enrutador inteligente. Si falla al clasificar una consulta, podría enrutar operaciones críticas hacia procesamiento generativo, comprometiendo la seguridad.

La supervisión humana es esencial, pero no elimina completamente este riesgo. Es una limitación estructural de trabajar con modelos probabilísticos en entornos donde la certeza es obligatoria.

Complejidad operativa del escalado propuesto. Pasar del prototipo monolítico a la arquitectura distribuida que se propone no es trivial. Requiere capacidades especializadas en operación de microservicios, observabilidad, seguridad de infraestructuras críticas y MLOps. También exige procesos de gestión de cambio, procedimientos de *rollback*, y certificación de seguridad específica del entorno militar.

Esta complejidad no debe subestimarse. El salto de prototipo a producción certificada en entornos clasificados puede requerir años de trabajo de equipos multidisciplinares. La Tabla 5-1 proporciona una hoja de ruta, pero implementarla sería un proyecto de enorme envergadura.

5.3 Líneas futuras de investigación y desarrollo

El trabajo abre varias direcciones de continuación que ampliarían las capacidades validadas y abordarían las limitaciones identificadas.

Validación operativa con usuarios finales y corpus representativo. La siguiente fase natural sería desplegar el sistema con usuarios reales y documentación representativa. Esto permitiría medir la efectividad operativa real (¿cuánto se reducen los tiempos de consulta? ¿mejora la calidad de las decisiones?), recoger *feedback* sobre usabilidad, e identificar casos límite no contemplados en el diseño inicial.

Esta validación también serviría para calibrar los umbrales de confianza del enrutador inteligente basándose en datos reales, no en estimaciones teóricas.

Integración progresiva con sistemas corporativos existentes. El despliegue real requiere conectar con sistemas transaccionales ya en producción: ERP, sistemas de mensajería militar, bases de datos *legacy*. Esta integración debe preservar trazabilidad completa y garantías transaccionales para operaciones críticas.

Un enfoque gradual mediante APIs bien definidas y adaptadores específicos minimizaría riesgos y permitiría validar cada integración de forma incremental.

Multimodalidad e interpretación de contenido visual. La documentación técnica incluye abundante información visual: diagramas, planos, esquemas, fotografías. La integración de modelos de visión por computador (Vision Language Models, VLMs) permitiría consultar directamente sobre contenido visual, detectar anomalías en inspecciones fotográficas, y validar automáticamente correspondencia entre inventario físico fotografiado y registros digitales. Esta capacidad multimodal ampliaría sustancialmente el valor operativo del sistema cuando el contenido visual resulta crítico para la comprensión completa del contexto.

Optimización de recursos y eficiencia computacional. El escalado a cientos de usuarios concurrentes y millones de documentos requiere optimizaciones que no son críticas en el prototipo actual: técnicas de caché inteligente para consultas frecuentes, compresión y cuantización de modelos para reducir latencia y coste computacional, estrategias de índice híbrido que combinen eficiencia de búsqueda léxica con riqueza semántica de *embeddings*, y mecanismos de auto escalado que adapten recursos según carga real. Estas optimizaciones determinarán la viabilidad económica del despliegue a escala completa.

Explicabilidad y auditoría avanzada. En entornos militares donde las decisiones pueden tener consecuencias críticas, la capacidad de explicar detalladamente por qué el sistema proporcionó determinada respuesta resulta fundamental. Líneas futuras incluyen técnicas de atribución de fuentes que identifiquen exactamente qué fragmentos documentales contribuyeron a cada afirmación, mecanismos de visualización de razonamiento que muestren el proceso de enrutamiento y recuperación, y sistemas de auditoría avanzada que registren no solo el resultado final sino el grafo completo de decisiones intermedias. Estas capacidades son esenciales para la certificación de seguridad en sistemas clasificados.

5.4 Reflexión final

Este trabajo ha explorado las técnicas contemporáneas de IA y ha demostrado empíricamente, usando un prototipo funcional, el potencial que ofrecen para gestión documental organizativa en entornos de alta criticidad, habilitando capacidades fundamentalmente nuevas que trascienden limitaciones de aproximaciones tradicionales. La arquitectura propuesta, validada técnicamente mediante caso de estudio exigente en logística militar de municiones, proporciona tanto fundamento conceptual como evidencia práctica para organizaciones que buscan modernizar sus sistemas de gestión de información aprovechando avances recientes en modelos de lenguaje, búsqueda semántica y generación aumentada, mientras mantienen control estricto sobre seguridad, trazabilidad y cumplimiento normativo.

La construcción del prototipo SIGLM ha confirmado que la hibridación RAG-SQL no es solo teóricamente viable sino prácticamente implementable con tecnologías actuales, y que es posible ejecutar modelos de lenguaje grandes en infraestructura local sin dependencia de servicios *cloud* externos, cumpliendo así requisitos de soberanía de datos críticos en entornos de Defensa. Esta demostración práctica reduce significativamente la incertidumbre asociada a la adopción de estas tecnologías en contextos donde el fracaso de un proyecto piloto tendría costes elevados.

Sin embargo, la adopción exitosa de estas tecnologías requiere reconocer honestamente tanto sus capacidades como sus limitaciones actuales. Los sistemas basados en IA no son panacea que resuelve automáticamente todos los desafíos de gestión documental; son herramientas poderosas que, cuando se diseñan, implementan y gobiernan apropiadamente, amplifican capacidades humanas de análisis y decisión sin reemplazar juicio experto crítico en contextos donde errores pueden tener consecuencias graves. El prototipo desarrollado ilustra esta filosofía: el sistema asiste al operador humano proporcionando acceso rápido a conocimiento relevante y validaciones automáticas, pero las decisiones finales sobre operaciones críticas permanecen bajo supervisión humana explícita.

El éxito de proyectos de transformación digital basados en IA depende tanto de excelencia técnica en arquitectura e implementación como de gestión efectiva de dimensiones organizativas y humanas:

alineación estratégica con objetivos operativos, inversión sostenida en desarrollo e infraestructura, formación de usuarios que les permita aprovechar plenamente capacidades del sistema comprendiendo sus limitaciones, comunicación transparente sobre qué puede y no puede hacer la tecnología, y gobernanza clara que balancee innovación con gestión prudente de riesgos.

La trayectoria de evolución de sistemas de gestión documental desde repositorios pasivos basados en indexación manual hasta asistentes cognitivos activos que comprenden semánticamente contenido y razonan sobre información compleja representa transformación fundamental en cómo las organizaciones gestionan conocimiento institucional y toman decisiones informadas. Este trabajo contribuye a esa trayectoria proporcionando fundamentos teóricos, propuesta arquitectónica detallada, prototipo funcional que valida empíricamente los conceptos propuestos, y análisis crítico de desafíos y oportunidades que esperamos guíe tanto trabajo académico futuro como implementaciones prácticas en organizaciones que operan en contextos donde gestión de información crítica determina efectividad operativa y ventaja estratégica. El camino desde el prototipo hasta el despliegue operativo en sistemas de alta criticidad es largo y complejo, pero las bases técnicas y metodológicas están establecidas y validadas.

6 BIBLIOGRAFÍA

- [1] J. McCarthy y P. Hayes, «Some philosophical problems from the standpoint of artificial intelligence,» *Machine Intelligence*, vol. 4, pp. 463-502, 1969.
- [2] A. B. Rashid, A. K. Kausik, A. A. Sunny y M. H. Bappy, «Artificial Intelligence in the Military: An Overview of the Capabilities, Applications, and Challenges,» *International Journal of Intelligent Systems*, vol. 1, 2023.
- [3] SmythOS, «The evolution of symbolic AI: From early concepts to modern applications.,» 2025. [En línea]. Available: <https://smythos.com/developers/agent-development/history-of-symbolic-ai/>.
- [4] D. E. Rumelhart, G. E. Hinton y R. J. Williams, «Learning representations by back-propagating errors,» *Nature*, vol. 323, pp. 533-536, 1986.
- [5] Y. Lecun, L. Bottou, Y. Bengio y P. Haffner, «Gradient-based learning applied to document recognition,» *Proceedings of the IEEE*, vol. 86, nº 11, pp. 2278-2324, 1998.
- [6] A. Krizhevsky, I. Sutskever y G. E. Hinton, «ImageNet Classification with Deep Convolutional Neural Networks,» *Advances in Neural Information Processing Systems*, vol. 25, pp. 1097-1105, 2012.
- [7] K. He, X. Zhang, S. Ren y J. Sun, «Deep residual learning for image recognition,» *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 770-778, 2016.
- [8] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez y L. Kaiser, «Attention is all you need.,» *Advances in Neural Information Processing Systems*, vol. 30, pp. 5998-6008, 2017.
- [9] J. Devlin, M. W. Chang, K. Lee y K. Toutanova, «BERT: Pre-training of deep bidirectional transformers for language understanding.,» de *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, Minneapolis, Minnesota, 2019.
- [10] T. M. Mitchel, *Machine learning*, New York: McGraw-Hill, 1997.
- [11] I. Goodfellow, Y. Bengio y A. Courville, *Deep learning*, Cambridge: MIT Press, 2016.

- [12] R. Dastres y M. Soori, «Artificial Neural Network Systems,» *International Journal of Imaging and Robotics*, vol. 21, nº 2, pp. 13-25, 2021.
- [13] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W. Yih, T. Rocktäschel, S. Riedel y D. Kiela, «Retrieval-augmented generation for knowledge-intensive NLP tasks.,» *Advances in Neural Information Processing Systems*, vol. 33, pp. 9459-9474, 2020.
- [14] B. Shi, X. Bai y C. Yao, «An end-to-end trainable neural network for image-based sequence recognition and its application to scene text recognition.,» *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, nº 11, pp. 2298-2304, 2017.
- [15] J. Lafferty, A. McCallum y F. Pereira, «Conditional random fields: Probabilistic models for segmenting and labeling sequence data.,» *In Proceedings of the 18th International Conference on Machine Learning*, pp. 282-289, 2001.
- [16] X. Li, J. Feng, Y. Meng, Q. Han, F. Wu y J. Li, «A unified MRC framework for named entity recognition.,» *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 5849-5859, 2020.
- [17] U. Zaratiana, N. Tomeh, P. Holat y T. Charnois, «GLiNER: Generalist model for named entity recognition using bidirectional transformer,» *Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics*, pp. 1561-1573, 2023.
- [18] T. Mikolov, K. Chen, G. Corrado y J. Dean, «Efficient estimation of word representations in vector space.,» *Proceedings of the 1st International Conference on Learning Representations (ICLR 2013).*, vol. arXiv, 2013.
- [19] T. Mikolov, Q. V. Le y I. Sutskever, «Exploiting Similarities among Languages for Machine Translation,» *arXiv*, 2013.
- [20] M. E. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee y L. Zettlemoyer, «Deep contextualized word representations.,» *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 2227-2237, 2018.
- [21] Y. Kim, «Convolutional neural networks for sentence classification.,» *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*, pp. 1746-1751, 2014.
- [22] Geeks for Geeks, «Difference Between Feed-Forward Neural Networks and Recurrent Neural Networks,» 01 08 2025. [En línea]. Available: <https://www.geeksforgeeks.org/data-analysis/difference-between-feed-forward-neural-networks-and-recurrent-neural-networks/>.
- [23] S. Hochreiter y J. Schmidhuber, «Long short-term memory.,» *Neural Computation*, vol. 9, nº 8, pp. 1735-1780, 1997.
- [24] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei y I. Sutskever, «Language models are unsupervised multitask learners,» *OpenAI Blog*, vol. 1, nº 8, 2019.
- [25] J. Kaplan, S. McCandish, T. Henighan, T. B. Bron, B. Chess, R. Child, S. Gray, A. Radford, J. Wu y D. Amodei, «Scaling Laws for Neural Language Models,» *arXiv*, 2020.

- [26] M. Mudaliyar, «Scaling Down Large Language Models with Small LLMs,» Medium, 24 03 2024. [En línea]. Available: <https://medium.com/@myliemudaliyar/scaling-down-large-language-models-hello-to-small-llms-f4ccd0ff0b37>.
- [27] Z. Ji, N. Lee, R. Frieske, T. Yu, D. Su, Y. Xu, E. Ishii, Y. J. Bang, A. Madotto y P. Fung, «Survey of Hallucination in Natural Language Generation,» *ACM Computing Surveys*, vol. 55, nº 12, pp. 1-38, 2023.
- [28] O. Ovadia, M. Brief, M. Mishaeli y O. Elisha, «Fine-Tuning or Retrieval? Comparing Knowledge Injection in LLMs,» *arXiv*, 2024.
- [29] J. Wei, X. Wang, D. Schuurmans, M. Bosma, B. Ichter, F. Xia, E. Chi, Q. Le y D. Zhou, «Chain-of-Thought Prompting Elicits Reasoning in Large Language Models,» *Advances in Neural Information Processing Systems*, vol. 35, pp. 24824-24837, 2022.
- [30] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Hinighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever y D. Amodei, «Language Models are Few-Shot Learners,» *Advances in Neural Information Processing Systems*, vol. 33, pp. 1877-1901, 2020.
- [31] P. Sahoo, A. K. Singh, S. Saha, V. Jain, S. Mondal y A. Chadha, «A systematic survey of prompt engineering in large language,» *arXiv*, 2024.
- [32] S. Yao, D. Yu, J. Zhao, I. Shafran, T. L. Griffiths, Y. Cao y K. Narasimhan, «Tree of thoughts: Deliberate problem solving with large language models.,» *Advances in Neural Information Processing Systems*, vol. 36, 2023.
- [33] S. Yao, J. Zhao, D. Ui, N. Du, I. Shafran, K. Narasimhan y Y. Cao, «ReAct: Synergizing reasoning and acting in language models.,» *Proceedings of the 11th International Conference on Learning Representations*, 2023.
- [34] O. Khattab y M. Zaharia, «ColBERT: Efficient and effective passage search via contextualized late interaction over BERT,» *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 39-48, 2020.
- [35] J. Johnson, M. Douze y H. Jégou, «Billion-scale similarity search with GPUs.,» *IEEE Transactions on Big Data*, vol. 7, nº 3, pp. 535-547, 2019.
- [36] Sanika, «What is Retrieval-Augmented Generation (RAG)?,» Medium, 2024 06 10. [En línea]. Available: <https://medium.com/@dixit.sanika0506/retrieval-augmented-generation-rag-586ba276c25d>.
- [37] Z. S. Harris, «Distributional Structure,» *WORD*, vol. 10, nº 2-3, pp. 146-162, 1954.
- [38] J. R. Firth, «A synopsis of linguistic theory, 1930-1955,» *Studies in Linguistic Analysis*, pp. 1-32, 1957.
- [39] Y. A. Malkov y D. A. Yashunin, «Efficient and robust approximate nearest neighbor search using Hierarchical Navigable Small World graphs,» *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 42, nº 4, pp. 824-836, 2020.
- [40] J. Wang, X. Yi, R. Guo, H. Jin, P. Xu, S. Li, X. Wang, X. Guo, C. Li, X. Xu, K. Yu, Y. Yuan, Y. Zou, J. Long, Y. Cai, Z. Li, Z. Zhang, Y. Mo, J. Gu, R. Jiang, Y. Wei y C. Xie,

- «Milvus: A purpose-built vector data management system.,» *Proceedings of the 2021 International Conference on Management of Data*, pp. 2614-2627, 2021.
- [41] OpenLegacy, « Monolithic Application vs Microservices Architecture: What They Are and Which May Suit You,» [En línea]. Available: <https://www.openlegacy.com/blog/monolithic-application>.
- [42] S. Singh, C. H. Muntean y S. Gupta, «Resilient microservices: an investigation into Istio effectiveness in Kubernetes,» *Cluster Computing*, vol. 29, nº 1, 2025.
- [43] J. Lewis y M. Fowler, «Microservices: a definition of this new architectural term,» [En línea]. Available: <https://martinfowler.com/articles/microservices.html>.
- [44] N. Dragoni, S. Giallorenzo, A. L. Lafuente, M. Mazzara, F. Montesi, R. Mustafin y L. Safina, «Microservices: Yesterday, Today, and Tomorrow,» de *Present and Ulterior Software Engineering*, Springer, 2017, pp. 195-216.
- [45] IBM Cloud Education, «Microservices: Advantages and Disadvantages.,» [En línea]. Available: <https://www.ibm.com/think/insights/microservices-advantages-disadvantages>.
- [46] Z. Yang, J. Tang, Z. Li, P. Wang, J. Wan, H. Zhong, X. Liu, M. Yang, P. Wang, S. Bai, L. Jin y J. Lin, «CC-OCR: A comprehensive and challenging OCR benchmark for evaluating large multimodal models in literacy,» *Proceedings of the IEEE/CVF International Conference on Computer Vision.*, 2025.
- [47] R. Smith, «An overview of the Tesseract OCR engine,» *Proceedings of the International Conference on Document Analysis and Recognition (ICDAR)*, pp. 629-633, 2007.
- [48] Y. Du, C. Li, R. Guo, X. Yin, W. Liu y J. Zhou, «PP-OCR: A practical ultra lightweight OCR system,» *ArXiv*, 2020.
- [49] R. S. Sandhu, E. J. Coyne, H. L. Feinstein y C. E. Youman, «Role-based access control models,» *IEEE Computer*, vol. 29, nº 2, pp. 38-47, 1996.
- [50] S. Haber y W. S. Stornetta, «How to time-stamp a digital document.,» *Journal of Cryptology*, vol. 3, nº 2, pp. 99-111, 1991.
- [51] UE, *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*, DOUE, 2024.
- [52] A. RoyChowdhury, M. Luo, P. Sahu, S. Banerjee y M. Tiwari, «ConfusedPilot: Confused deputy risks in RAG-based LLMs.,» *arXiv*.
- [53] M. Gupta, C. K. Akula y E. Wong, «Lleaking: A comprehensive survey on information leakage in large language models,» *ACM Computing Surveys*, 2024.
- [54] X. Shen, Z. Chen, M. Backes, Y. Yun y Y. Zhang, «"Do Anything Now": Characterizing and evaluating in-the-wild jailbreak prompts on large language models.,» de *Proceedings of the 2024 ACM SIGSAC Conference on Computer and Communications Security*.
- [55] Y. Liu, G. Deng, Y. Li, K. Wang, T. Zhang, Y. Liu, H. Wang, Y. Zheng y Y. Liu, «Prompt injection attack against LLM-integrated applications.,» *arXiv*.
- [56] C. Anil, E. Durmus, N. Panickssery, M. Sharma y J. Kaplan, «Many-shot jailbreaking.,» 2024. [En línea]. Available: <https://www.anthropic.com/research/many-shot-jailbreaking>.

- [57] S. P. Sivashanmugam, «Model Inversion Attacks on Llama 3: Extracting PII from Large Language Models,» *arXiv*.
- [58] OWASP, «OWASP Top 10 for Large Language Model Applications 2025.,» OWASP Foundation, [En línea]. Available: <https://genai.owasp.org/llm-top-10/>.
- [59] D. Zhao, «FRAG: Toward federated vector database management for collaborative and secure retrieval-augmented generation.,» *arXiv*, 2024.
- [60] S. Rose, O. Borchert, S. Mitchell y S. Connelly, «Zero Trust Architecture,» *National Institute of Standards and Technology*, Vols. %1 de %2NIST Special Publication 800-207, 2020.
- [61] ISO/IEC, *ISO/IEC 42001:2023 Information technology — Artificial intelligence — Management system.*, 2023.
- [62] ISO/IEC, *ISO/IEC 23894:2023 Information technology — Artificial intelligence — Guidance on risk management.*, 2023.
- [63] A. Tesliuk, S. Bobkov, V. Ilyin, A. Novikov, A. Poyda y V. Velikhov, «Kubernetes container orchestration as a framework for flexible and effective scientific data analysis.,» *2019 Ivannikov Memorial Workshop (ISPRAS)*, 2019.
- [64] S. Debroy, «LangGraph — Architecture and Design,» 2025 05 25. [En línea]. Available: <https://medium.com/@shuv.sdr/langgraph-architecture-and-design-280c365aaf2c>.
- [65] J. Hoffmann, S. Borgeaud, A. Mensch, E. Buchatskaya, T. Cai, E. Rutherford, D. Casas, L. Hendricks, J. Welbl, A. Clark, T. Hennigan, E. Noland, K. Millican, G. Driessche, B. Damoc y A. Guy, «Training Compute-Optimal Large Language Models,» *arXiv*, 2022.
- [66] M. Stonebraker y L. A. Rowe, «The design of POSTGRES.,» *ACM SIGMOD Record*, vol. 15, nº 2, pp. 340-355, 1986.
- [67] C. A. Mattmann y J. L. Zitting, *Tika in Action*, Manning Publications, 2011.
- [68] Y. Fain y A. Moiseev, *Angular Development with TypeScript (2nd ed.)*, Manning Publications, 2018.
- [69] B. Burns, B. Grant, D. Oppenheimer, E. Brewer y J. Wilkes, «Borg, Omega, and Kubernetes,» *Communications of the ACM*, vol. 59, nº 5, pp. 50-57, 2016.
- [70] S. Es, J. James, L. Espinosa-Anke y S. Schockaert, «RAGAs: Automated Evaluation of Retrieval Augmented Generation,» *arXiv*, 2023.
- [71] ISO, *2016 Information and documentation — Records management — Part 1: Concepts and principles.*, International Organization for Standarization., 2016.
- [72] ISO/IEC, *ISO/IEC 27001:2022 Information security, cybersecurity and privacy protection — Information security management systems — Requirements*, 2022.
- [73] F. Feng, Y. Yang, D. Cer, N. Arivazhagan y W. Wang, «Language-agnostic BERT sentence embedding,» *arXiv*, 2020.
- [74] V. Karpukhin, B. Oguz, S. Min, P. Lewis, L. Wu, S. Edunov, D. Chen y W. T. Yih, «Dense passage retrieval for open-domain question answering.,» *Proceedings of EMNLP.*, 2020.

ANEXO I. PROTOTIPO SIGLM: CASOS DE USO

Este anexo presenta cinco casos de uso representativos del prototipo SIGLM, demostrando la integración práctica de los componentes arquitectónicos descritos en el capítulo 4. Cada caso incluye una descripción narrativa del comportamiento del sistema, instrucciones específicas para su reproducción a través de capturas de pantalla, y un análisis técnico de la ingeniería de *prompt* y el flujo de ejecución subyacente. Los casos están ordenados de manera progresiva, desde la consulta sin contexto hasta la integración completa de OCR, RAG y validación SQL, ilustrando cómo el prototipo implementa efectivamente el diseño híbrido propuesto.

1. Consulta general

Este caso demuestra el comportamiento del sistema en su estado inicial, cuando la base de datos vectorial ChromaDB no contiene documentación técnica indexada. El prototipo no ha recibido *fine-tuning* sobre el dominio de municiones militares, por lo que el modelo base Mistral 7B a través de Ollama opera exclusivamente con su conocimiento paramétrico general.

En este escenario, el sistema muestra un comportamiento diferenciado según el tipo de consulta. Para preguntas generalistas de cultura general (capitales, conceptos básicos, matemáticas simples), el LLM responde utilizando su conocimiento previo al entrenamiento. Sin embargo, cuando se formulan preguntas específicas sobre documentación técnica militar, normativa de municiones o procedimientos de almacenamiento en polvorines, el sistema reconoce correctamente su limitación y rechaza proporcionar respuestas que podrían constituir alucinaciones peligrosas en un contexto operativo crítico.

Esta capacidad de discriminación no es accidental, sino resultado del diseño del *prompt* del sistema que instruye explícitamente al modelo a no inventar información sobre municiones militares. El prototipo prioriza así la seguridad sobre la aparente utilidad, estableciendo desde el inicio la disciplina de respuesta que se mantendrá cuando la base vectorial contenga documentación real.

Instrucciones de prueba

Para reproducir este caso y obtener capturas de pantalla, seguir estos pasos en el prototipo SIGLM:

- Iniciar el prototipo: Ejecutar “streamlit run SIGLM.py” desde el directorio del proyecto. Asegurarse de que Ollama está ejecutándose con el modelo mistral:7b disponible localmente.
- Verificar estado inicial: En la interfaz de Streamlit, observar la tabla de inventario (inicialmente vacía o con datos de ejemplo) y el panel lateral de gestión de documentación. No debe haber documentos PDF indexados en este punto, si los hubiera se pueden borrar pulsando el botón “Reset Sistema”.
- Primera consulta generalista: En el campo de chat, escribir una pregunta general como ;¿Cuál es la capital de Francia?; o; ¿Qué es el aprendizaje automático?;. El sistema, proporcionará información correcta basada en conocimiento general o bien no contestar por no ser del ámbito de su responsabilidad (ver Figura A-1).



Figura A-1. Ejemplo de consulta genérica al LLM sin RAG.

- Segunda consulta específica sobre municiones: Escribir una pregunta técnica como *¿Cuál es la capacidad NEQ máxima del polvorín Alpha?*; o *Dame información sobre munición de 155 mm*;. El sistema indicará explícitamente su incapacidad para responder sin documentación (ver Figura A-2).



Figura A-2. Consulta sobre municiones, por restricciones requiere RAG, pero no hay documentación cargada.

Análisis técnico e ingeniería de prompt

El comportamiento observado en este caso está gobernado por el *prompt* del sistema definido en la constante `SYSTEM_PROMPT` del archivo `SIGLM.py`. Este *prompt* establece tres capas de instrucción jerárquicas que modelan el comportamiento del asistente:

Identidad y contexto operativo: El *prompt* comienza definiendo al agente como un asistente especializado en gestión de municiones, estableciendo inmediatamente el dominio de operación y creando expectativas de especialización. Esta identidad funciona como anclaje semántico para todo el procesamiento posterior.

Capacidades y limitaciones explícitas: La sección central del *prompt* enumera las herramientas disponibles con el decorador `@tool` de LangChain (`consultar_inventario`, `consultar_capacidad_polvorin`, etc.) y, crucialmente, instruye al modelo sobre cuándo NO debe responder. La instrucción clave es:

;Si no tienes información específica en la documentación cargada, NO INVENTES datos sobre municiones o especificaciones técnicas. Indica claramente que necesitas que se cargue documentación relevante.;

Esta instrucción negativa es fundamental para prevenir alucinaciones. En lugar de confiar únicamente en la tendencia natural del modelo a completar patrones lingüísticos (lo que podría generar especificaciones técnicas plausibles pero incorrectas), el *prompt* establece explícitamente la política de rechazo. El modelo evalúa cada consulta contra dos criterios: (1) ¿Es una pregunta de dominio general donde mi conocimiento paramétrico es confiable? (2) ¿Requiere información específica de documentación que debería existir en el *vectorstore* pero que no está presente?

Tono y estructura de respuesta: La última sección del *prompt* especifica que las respuestas deben ser claras, concisas y técnicamente precisas, guiando no solo el qué responder sino el cómo. Este control estilístico asegura uniformidad en las interacciones incluso cuando el sistema rechaza consultas.

El flujo de ejecución para una consulta sin contexto es el siguiente: (1) La función `ejecutar_con_herramientas()` recibe la consulta del usuario. (2) El sistema de prioridades analiza la consulta mediante expresiones regulares y clasificación por el LLM, pero ninguna prioridad coincide con comandos específicos (no hay datos SQL, no hay documentos cargados). (3) La consulta cae en el caso por defecto, invocando al LLM con el *prompt* del sistema completo más el historial de conversación. (4) El modelo, guiado por las instrucciones del *prompt*, evalúa si puede responder con conocimiento general o debe rechazar por falta de contexto específico. (5) La respuesta se renderiza en Markdown en la interfaz de Streamlit.

2. Inserción de documentación

Este caso demuestra el proceso de ingestión de documentación técnica en la base de datos vectorial del sistema, transformando documentos PDF en *embeddings* semánticos consultables. El prototipo implementa un pipeline completo de preprocesamiento que maneja tanto PDFs con texto seleccionable (generados digitalmente) como documentos escaneados que requieren OCR.

Cuando el usuario carga un archivo PDF a través del componente `file_uploader` de Streamlit, el sistema intenta primero extraer texto directamente usando PyPDF2, asumiendo que el documento fue generado digitalmente. Si esta extracción produce resultados vacíos o con bajo contenido textual (indicativo de páginas escaneadas), el sistema automáticamente invoca el motor Tesseract OCR configurado con parámetros optimizados para documentación técnica militar (`--psm 6` para detección de bloques uniformes de texto, idioma español con respaldo en inglés).

Una vez extraído el texto completo, el sistema lo fragmenta en chunks con `RecursiveCharacterTextSplitter` con configuración `chunk_size=1000` y `chunk_overlap=200`. Esta fragmentación no es arbitraria: el tamaño de 1000 caracteres equilibra granularidad semántica (chunks demasiado pequeños pierden contexto, demasiado grandes diluyen relevancia) y eficiencia de recuperación. El overlap de 200 caracteres previene la fragmentación de conceptos que cruzan límites de chunk, asegurando que información relacionada permanezca recuperable.

Cada *chunk* se transforma en un *embedding* de 1024 dimensiones con el modelo `mxbai-embed-large` ejecutado localmente a través de Ollama, optimizado para español e inglés. Estos embeddings se indexan en ChromaDB utilizando el algoritmo HNSW (Hierarchical Navigable Small World), que permite búsqueda aproximada de vecinos más cercanos con complejidad logarítmica. La base vectorial persiste en disco en el directorio `./db_vectores` (definido en `PERSIST_DIRECTORY`), asegurando que la documentación indexada sobreviva reinicios del sistema.

3. Consulta RAG

Este caso demuestra el patrón RAG en operación, mostrando cómo el sistema transforma consultas de lenguaje natural en búsquedas vectoriales, recupera fragmentos relevantes de documentación, y sintetiza respuestas fundamentadas en evidencia documental. A diferencia del Caso 1 donde el sistema rechazaba consultas técnicas por falta de contexto, aquí la base vectorial poblada permite responder sobre especificaciones de municiones, normativa de almacenamiento, y procedimientos operativos.

El sistema implementa dos modalidades de consulta RAG según la naturaleza de la pregunta. Para consultas sobre características generales de polvorines (capacidad NEQ, dimensiones, grupo de riesgo permitido), el sistema invoca la tool `consultar_capacidad_polvorin`, que orquesta internamente la búsqueda vectorial. Para consultas más abiertas sobre documentación técnica general, se utiliza `consultar_documentacion_tecnica`. Ambas tools comparten el mismo motor de recuperación, pero difieren en los metadatos de filtrado aplicados.

El flujo RAG sigue el esquema habitual de tres etapas: (1) la consulta se transforma a embedding usando el mismo modelo empleado para indexación (mxbai-embed-large), manteniendo la consistencia del espacio vectorial; (2) se ejecuta una búsqueda de similitud en Chroma para recuperar los fragmentos más cercanos a la consulta (con *k* configurable según el caso de uso y, en ocasiones, limitado tras eliminar duplicados); y (3) se construye un prompt aumentado que combina la consulta original con el contexto recuperado e instruye al LLM a responder basándose exclusivamente en ese contexto, reduciendo al mínimo inferencias fuera de la evidencia aportada.

Un aspecto crítico del diseño es la gestión de alucinaciones mediante *prompt engineering* defensivo. El *prompt* aumentado incluye instrucciones explícitas. Si la información no está en el contexto, indica claramente que no puedes responder con certeza. Esta instrucción negativa contrarresta la tendencia del modelo a completar patrones incluso cuando el contexto recuperado es insuficiente o ambiguo. El sistema prioriza confiabilidad sobre completitud, alineado con requisitos de sistemas críticos.

Instrucciones de prueba

Para demostrar consultas RAG efectivas, primero debemos tener información indexada y, posteriormente, seguiremos estos pasos:

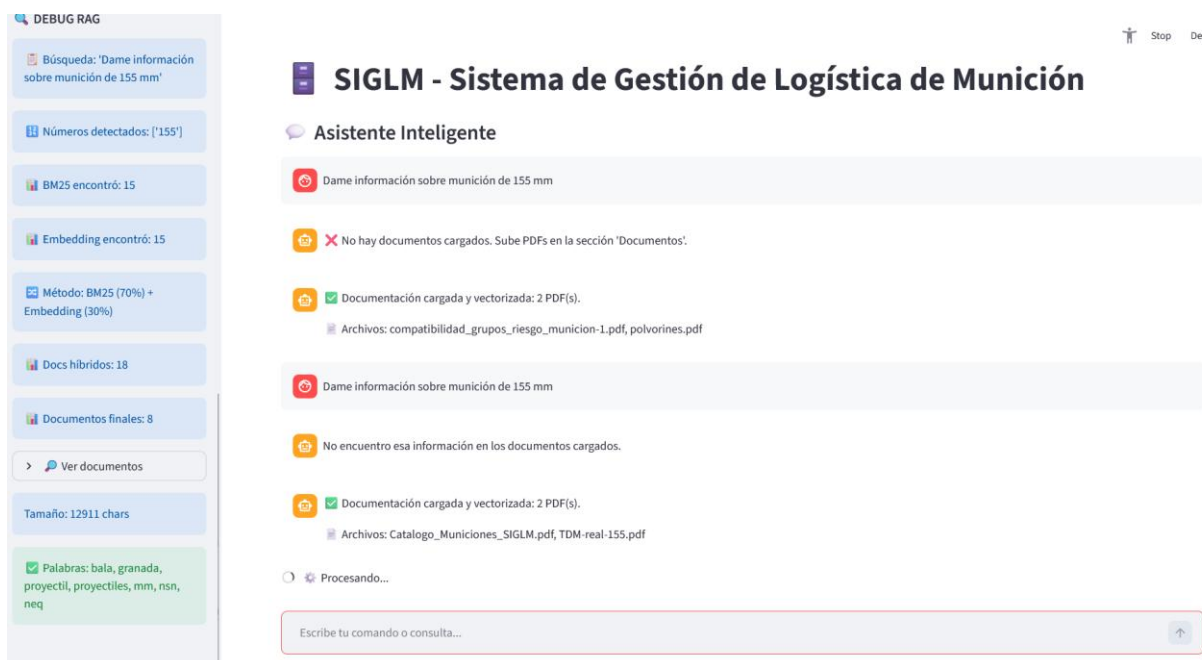


Figura A-3. Búsqueda RAG de información sobre municiones. En el panel izquierdo se observa cómo realiza la búsqueda.

- Consulta específica de munición: Escribir *Dame información sobre munición de 155 mm*. Inicialmente no encuentra la respuesta, posteriormente se observa en la Figura A-3 que se cargan los documentos y al volver a hacer la pregunta, se muestra la búsqueda RAG en la barra lateral (Figura A-4)
- En caso de que se consulte por una munición que no exista, el sistema no debe contestar.

The screenshot displays the 'Gestión de Documentos' (Document Management) interface on the left and a chat window on the right. The chat window shows a conversation where the user asks for information about 155mm ammunition. The system responds with a detailed technical description of a 155mm HE projectile, including its velocity, weight, and explosive content.

Gestión de Documentos

Sube PDFs con: normativa, especificaciones, polvorines, albaranes

Subir PDFs

Drag and drop files here
Limit 200MB per file • PDF

Browse files

TDM-real-155.pdf
91.7KB

Catalogo_Municione...
92.9KB

Procesar PDFs

Reset Sistema

Comprobar Tesseract

Inventario Actual

id	polvorin_id	nombre_municion
1	ALPHA	Proyectil 155mm
4	GAMMA	Bala 9mm Parabellu
5	BETA	Cartucho 5.56 mm

Chat Conversation:

Documentación cargada y vectorizada: 2 PDF(s).
Archivos: compatibilidad_grupos_riesgo_municon-1.pdf, polvorines.pdf

Dame información sobre munición de 155 mm

No encuentro esa información en los documentos cargados.

Documentación cargada y vectorizada: 2 PDF(s).
Archivos: Catalogo_Municones_SIGLM.pdf, TDM-real-155.pdf

Dame información sobre munición de 155 mm

El catálogo técnico de municiones de la Armada Española incluye información sobre un proyectil de 155 mm HE. Este proyectil tiene una designación oficial de "PROYECTIL 155mm HE" y un código NSN de "1320-33-217-0091". Pertenece al grupo de riesgo L2D, lo que indica un riesgo de proyección pero no de explosión en masa. Su cantidad neta de explosivo (NEQ) es de 10,3 kg TNT.

El proyectil tiene una velocidad esperada de 938 m/s a +21°C y su peso es de 42,7 kg sin la espoleta. La espoleta utilizada es el modelo EC-102 o PD-M572 AP Estopín a percusión M82. El proyectil se encuentra en un lote con el número "1320-33-217-0091".

No se proporcionan detalles específicos sobre su embalaje, almacenamiento ni consideraciones de seguridad en el contexto dado.

Escribe tu comando o consulta...

Figura A-4. Respuesta a la consulta RAG sobre municiones.

- Consulta sobre normativa: Probar con *¿Qué grupos son compatibles con munición del grupo B?* Esta consulta requiere recuperación de normativa de compatibilidad, típicamente presente en documentación reglamentaria militar. En este caso es posible repreguntar: *¿Qué es la condición 1 de compatibilidad de munición?* Figura A-5.

The screenshot displays the 'Gestión de Documentos' (Document Management) interface on the left and a chat window on the right. The chat window shows a conversation where the user asks for information about ammunition compatibility. The system responds with a list of compatible groups (B, C, D, E, F, G) and a detailed explanation of the compatibility conditions.

Gestión de Documentos

Sube PDFs con: normativa, especificaciones, polvorines, albaranes

Subir PDFs

Drag and drop files here
Limit 200MB per file • PDF

Browse files

Reset Sistema

Comprobar Tesseract

Inventario Actual

id	polvorin_id	nombre_municion
1	ALPHA	Proyectil 155mm
4	GAMMA	Bala 9mm Parabellu
5	BETA	Cartucho 5.56 mm

SIGLM - Sistema de Gestión de Logística de Munición

Asistente Inteligente

¿Qué grupos son compatibles con munición del grupo B?

La munición del grupo B es compatible con los siguientes grupos:

- B
- C (solo si se cumple la condición 1)
- D (solo si se cumple la condición 1)
- E (solo si se cumple la condición 1)
- F (solo si se cumple la condición 2)
- G (solo si se cumple la condición 3)
- S

¿Qué es la condición 1 de compatibilidad de munición?

La condición 1 de compatibilidad de munición se refiere a que las espoletas del Grupo B pueden almacenarse con los artículos a los que se montarán, pero la Cantidad Neta Explosiva (NEQ) se agrega y se trata como Grupo F.

Escribe tu comando o consulta...

Figura A-5. Respuesta a consulta RAG sobre normativa.

Análisis técnico e ingeniería de prompt

La consulta RAG es el caso más sofisticado de ingeniería de prompt en el prototipo, ya que combina recuperación automática de contexto con generación condicionada. El flujo técnico completo se ejecuta cuando el sistema de prioridades detecta una consulta sobre polvorines o documentación técnica:

Detección de intención: El sistema de prioridades de ejecutar_con_herramientas() analiza la consulta mediante expresiones regulares que buscan palabras clave como ;polvorín;, ;capacidad;, ;almacenar;, ;normativa;. Si se detecta match en prioridad 2 (consultas sobre polvorines), se invoca la tool correspondiente. Este enrutamiento es determinístico y ocurre antes de cualquier invocación del LLM, evitando costos computacionales innecesarios.

Recuperación vectorial: Dentro de la tool (e.g., consultar_capacidad_polvorin), la consulta se transforma en embedding mediante embeddings.embed_query(consulta). Este vector de 384 dimensiones se usa para buscar en ChromaDB con configuración k=5 (recuperar 5 fragmentos). ChromaDB ejecuta búsqueda HNSW con complejidad $O(\log N)$ en lugar de búsqueda lineal $O(N)$, crucial para escalabilidad. Los resultados incluyen no solo los chunks de texto sino también sus scores de similitud (valores entre 0 y 1, donde 1 es identidad perfecta).

Construcción del *prompt* aumentado: Los fragmentos recuperados se concatenan con delimitadores claros y se integran en una plantilla de *prompt* que sigue esta estructura:

Template de prompt RAG:

CONTEXTO DE DOCUMENTACIÓN TÉCNICA:\n[Fragmentos recuperados con delimitadores]\n\nPREGUNTA DEL USUARIO:\n[Consulta original]\n\nINSTRUCCIONES:\n- Responder SOLO basándose en el contexto proporcionado\n- Si la información no está en el contexto, indicarlo explícitamente\n- Citar fuentes cuando sea posible (e.g., Según el manual técnico...;\n- Ser técnicamente preciso y conciso

Esta estructura de *prompt* implementa dos principios críticos de RAG: separación clara entre contexto y consulta (evitando que el LLM confunda ambos), e instrucciones explícitas de fundamentación (para contrarrestar la tendencia del modelo a completar patrones de forma especulativa).

Generación condicionada: El LLM (Mistral vía Ollama) procesa el *prompt* aumentado y genera respuesta. Aquí actúa el mecanismo de atención del *transformer*: el modelo calcula scores de relevancia entre cada token de salida y todos los tokens del *prompt* de entrada, permitiéndole “atender”; selectivamente a los fragmentos más relevantes del contexto recuperado.

Manejo de insuficiencia de contexto: Si los 5 fragmentos recuperados tienen scores de similitud muy bajos (< 0.3 , umbral configurable), el sistema puede optar por no invocar al LLM y devolver directamente un mensaje de información no disponible. Esta lógica de corte temprano previene generación de respuestas especulativas basadas en contexto débilmente relacionado.

4. Inserción de datos en SQL

Este caso demuestra la integración del LLM con operaciones transaccionales determinísticas sobre la base de datos relacional SQLite. A diferencia del RAG que opera sobre conocimiento cualitativo y semántico, las operaciones SQL requieren pre en el manejo de cantidades numéricas, identificadores de polvorín, y estados de inventario. El prototipo implementa un sistema de validación en cascada que previene operaciones inválidas antes de que puedan corromper la integridad de los datos.

El sistema soporta dos tipos de modificaciones de inventario: adiciones (entradas de munición en polvorín) y extracciones (salidas). Cada tipo activa una cadena diferente de validaciones implementadas en las tools ejecutar_movimiento_municion y extraer_municion. Para adiciones, el sistema verifica: (1) existencia del polvorín destino en la base de datos, (2) disponibilidad de capacidad NEQ residual, (3)

compatibilidad de grupo de riesgo con municiones ya almacenadas, y (4) cumplimiento de normas de seguridad codificadas en la función `validar_polvorin()`.

Para extracciones, las validaciones son más simples, pero igualmente críticas: (1) existencia de la munición específica en el polvorín origen, y (2) disponibilidad de cantidad suficiente para satisfacer la operación. Si cualquier validación falla, el sistema rechaza la operación completa y explica al usuario la causa del rechazo, promoviendo transparencia operativa.

Un aspecto clave del diseño es el flujo conversacional de confirmación. Cuando el LLM interpreta una intención de modificación de inventario (p. ej: *“añade 50 unidades de munición X al polvorín ALPHA”*), el sistema NO ejecuta inmediatamente la operación. En su lugar, construye un mensaje de confirmación detallado que resume los parámetros extraídos: polvorín, munición, cantidad, NEQ total calculado, y cualquier dato técnico recuperado de la documentación RAG. El usuario debe confirmar explícitamente con *sí* o *confirmar*; antes de que la transacción SQL se ejecute.

Este patrón de confirmación bidireccional es fundamental para prevenir errores operativos en un contexto militar crítico. Sin confirmación humana, un error de interpretación del LLM (malentender ;50; como ;500; o confundir polvorines) podría resultar en registros de inventario incorrectos con consecuencias potencialmente graves. El flujo de confirmación actúa como *guardrail* humano en el *loop* de decisión.

Instrucciones de prueba

Para demostrar operaciones SQL con validación, seguir este escenario completo:

- Estado inicial: Observar la tabla de inventario en la interfaz principal.
 - Operación válida - Adición: Escribir en el chat *“Añade 20 unidades de cartuchos de 5,56 mm al polvorín BETA”*. El sistema debe: (a) buscar datos técnicos de la munición en el RAG
- Figura A-6, (b) validar capacidad del polvorín, (c) calcular NEQ total



Figura A-6. Tras una orden mediante chat de inserción en SQL, el sistema captura los datos recibidos, y trata de buscar por RAG los faltantes.

- Confirmación de operación: Responder Sí, confirmar; en el chat. El sistema debe ejecutar la transacción SQL y actualizar la tabla de inventario visible. Figura A-7.

Gestión de Documentos

Sube PDFs con: normativa, especificaciones, polvorines, albaranes

Subir PDFs

Drag and drop files here
Limit 200MB per file • PDF

Browse files

Reset Sistema

Comprobar Tesseract

Inventario Actual

id	polvorin_id	nombre_municion
4	GAMMA	Bala 9mm Parabellu
5	BETA	Cartucho 5.56 mm
6	GAMMA	Bala 9mm Parabellu
7	GAMMA	Bala 9mm Parabellu
8	BETA	Bala 9mm
9	BETA	Cartucho 5.56 mm

Asistente Inteligente

Mete 20 unidades de cartuchos de 5,56 mm al polvorin BETA

DATOS ENCONTRADOS EN DOCUMENTACIÓN

Munición identificada: **Cartucho 5.56 mm**

- NEQ unitario: 0.045 kg
- Grupo de riesgo: 1.2E

¿Proceder? ☒ Añadir 20 unidades al polvorin BETA

Responde "Sí" para confirmar.

sí

MOVIMIENTO AUTORIZADO

OPERACIÓN COMPLETADA:

- Munición: Cartucho 5.56 mm
- Cantidad: 20 unidades
- Polvorin destino: BETA
- NEQ añadido: 0.8999999999999999 kg
- Grupo de riesgo: E

ESTADO DEL POLVORIN BETA:

- NEQ anterior: 4.54 kg
- NEQ actual: 5.44 kg
- NEQ máximo: 500.0 kg
- Espacio restante: 494.56 kg (98.9%)

✓ Grupo del polvorin: E ✓ Validación NEQ: OK ✓ Validación compatibilidad: OK

Figura A-7. Inserción de datos en base SQL mediante chatbot.

- Operación inválida - Exceso de capacidad: Intentar añadir una cantidad excesiva que supere la capacidad NEQ del polvorin: Al añadir una cantidad excesiva que sobrepase el NEQ disponible de un polvorin, se rechaza el movimiento Figura A-8.

Gestión de Documentos

Sube PDFs con: normativa, especificaciones, polvorines, albaranes

Subir PDFs

Drag and drop files here
Limit 200MB per file • PDF

Browse files

Reset Sistema

Comprobar Tesseract

Inventario Actual

id	polvorin_id	nombre_municion
1	ALPHA	Proyectil 155mm
4	GAMMA	Bala 9mm Parabellu
5	BETA	Cartucho 5.56 mm

Asistente Inteligente

mete 100 proyectiles de 155 mm en polvorin alpha

DATOS ENCONTRADOS EN DOCUMENTACIÓN

Munición identificada: **Proyectil 155 mm**

- NEQ unitario: 10.5 kg
- Grupo de riesgo: D

¿Proceder? ☒ Añadir 100 unidades al polvorin ALPHA

Responde "Sí" para confirmar.

sí

OPERACIÓN DENEGADA

NEQ resultante (1155.00 kg) SUPERA el límite del polvorin ALPHA (500.00 kg máximo).

Diferencia: +655.00 kg

Escribe tu comando o consulta...

Figura A-8. Movimiento rechazado por restricciones de seguridad.

- Operación de extracción: Escribir “saca 50 unidades de balas de 9 mm del polvorin GAMMA”; Confirmar la operación y observar la actualización de la tabla. Si se intenta extraer más unidades de las disponibles, devuelve mensaje de rechazo por stock insuficiente Figura A-9.



Figura A-9. Extracción de munición de base SQL, una satisfactoria y otra fallida.

Análisis técnico e ingeniería de prompt

Las operaciones SQL representan el caso más crítico de integración LLM-sistema determinístico, donde un error de interpretación puede tener consecuencias operativas graves. El prototipo implementa un diseño defensivo de cinco capas:

Arquitectura defensiva de cinco capas:

Capa 1 - Detección de intención (regex determinística) Capa 2 - Extracción de parámetros (LLM con prompt estructurado) Capa 3 - Enriquecimiento vía RAG (búsqueda automática de datos técnicos) Capa 4 - Validaciones de negocio en cascada (reglas determinísticas) Capa 5 - Confirmación bidireccional humana (guardrail final)

Capa 1 - Detección de intención usando prioridades: El sistema de prioridades de ejecutar_con_herramientas() analiza la consulta con expresiones regulares específicas. La prioridad 5 identifica solicitudes de adición mediante patrones como “añad(e|ir)”, “agreg(a|ar)”, “ingresar” o “entrada”, combinados con referencias a polvorines o municiones. La prioridad 4 detecta extracciones mediante patrones como “extra(e|er)”, “sacar 2” “retirar” o “salida”. Esta detección basada en expresiones regulares es rápida y determinista, y evita invocaciones innecesarias al modelo cuando la orden del usuario es inequívoca.

Capa 2 - Extracción de parámetros mediante LLM: Una vez detectada la intención, el sistema invoca al LLM con un prompt especializado de extracción estructurada:

Prompt de extracción estructurada:

Extrae los siguientes parámetros de la consulta del usuario. Responde SOLO con JSON válido sin texto adicional. \n\nFormato requerido: \n{\n “polvorin_id”: “ALPHA|BETA|GAMMA|DELTA”,\n “nombre_municion”: “nombre completo”,\n “cantidad”: numero_entero,\n “neq_unitario”: numero_decimal_o_null,\n “grupo_riesgo”: “A-G|null”\n}\n\nEjemplos de normalización: \n- “polvorín alfa” → “ALPHA” \n- “veinte unidades” → 20 \n- “50 proyectiles” → 50;

Este prompt incluye ejemplos few-shot de conversiones válidas para guiar al modelo en normalización de formatos. El énfasis en “SOLO JSON” y el formato estructurado específico contrarresta la tendencia del modelo a añadir preámbulos o explicaciones accesorias.

Capa 3 - Enriquecimiento vía RAG: Si el usuario no especificó NEQ unitario o grupo de riesgo (campos críticos para validación), el sistema invoca automáticamente `buscar_datos_municion_en_pdf()`, que consulta la base vectorial con *queries* específicas como “[nombre_municion] NEQ” y “[nombre_municion] grupo riesgo”. Los fragmentos recuperados se pasan a un *prompt* de extracción que busca patrones numéricos y letras de grupo, con *fallback* a NULL si no encuentra datos. Este enriquecimiento automático reduce carga cognitiva del usuario, que no necesita recordar especificaciones técnicas.

Capa 4 - Validaciones de negocio en cascada: La función `validar_polvorin()` implementa la lógica de seguridad de municiones. Para cada polvorín, consulta la tabla inventario para obtener: (a) NEQ total actualmente almacenado, (b) grupos de riesgo de todas las municiones presentes. Luego ejecuta verificaciones:

Reglas de validación determinísticas:

• *Capacidad NEQ*: $NEQ_actual + NEQ_nueva \leq NEQ_máximo_polvorin$ • *Compatibilidad de grupos*: Consulta tabla de compatibilidad codificada • *Polvorín vacío*: Si está vacío, permitir cualquier grupo (inicializa) • *Stock disponible (extracciones)*: $cantidad_solicita \leq cantidad_almacenada$

Si cualquier validación falla, la función retorna un diccionario con “valido”: False y “motivo” con explicación técnica del rechazo. Este diccionario se presenta al usuario, proporcionando *feedback* educativo sobre por qué la operación no es posible.

Capa 5 - Flujo de confirmación bidireccional: Si todas las validaciones pasan, el sistema NO ejecuta la transacción inmediatamente. En su lugar, construye un mensaje de confirmación con formato estructurado que resume todos los parámetros. Este mensaje se almacena en `st.session_state` junto con los datos de la operación. El sistema queda en estado “esperando confirmación”, monitoreando la siguiente entrada del usuario. Si detecta palabras de confirmación positiva (mediante prioridad 1: *regex* matching “s[ii]”, “confirmar”, “adelante”, etc.), ejecuta la transacción SQL y limpia el estado. Si detecta cancelación (“no”, “cancelar”), aborta y limpia el estado sin ejecutar nada.

La transacción SQL real es extremadamente simple: para adiciones, un INSERT INTO inventario o un UPDATE si la munición ya existe en el polvorín. Para extracciones, un UPDATE decrementando cantidad, seguido de un DELETE si cantidad llega a cero. La simplicidad de las *queries* SQL contrasta con la complejidad de las validaciones previas, reflejando el principio de “validar intensivamente, ejecutar trivialmente”.

El código Python luego analiza la respuesta con `extraer_json_robusto()`, una función que busca el primer objeto JSON válido en la respuesta usando *regex*, tolerando preámbulos o explicaciones accesorias que el LLM ocasionalmente añade. Esta robustez es crítica porque los LLM, especialmente modelos más pequeños como Mistral 7B, no siempre siguen instrucciones de formateo perfectamente.

5. Autodetección de albaranes para inserción en SQL

Este caso demuestra la capacidad más avanzada del prototipo: el procesamiento automático end-to-end de documentos administrativos de movimiento de municiones. El sistema integra OCR para digitalización de formularios escaneados, procesamiento de lenguaje natural para clasificación y extracción de entidades, recuperación RAG para enriquecimiento de datos técnicos, y validación SQL con confirmación humana para ejecución transaccional. Esta cadena de procesamiento completa representa la propuesta de valor fundamental del sistema SIGLM: transformar documentos en papel en actualizaciones de inventario validadas, reduciendo carga administrativa y errores de transcripción manual.

El pipeline comienza cuando el usuario carga un archivo PDF en el panel de gestión de documentación. El sistema aplica primero clasificación de documento mediante la función `es_movimiento_de_municion()`, que implementa un algoritmo híbrido de dos etapas: (1) Búsqueda

determinística de patrones textuales indicativos de albaranes (ALBARÁN, GUÍA DE ENTREGA, ORDEN DE MOVIMIENTO, ENTRADA/SALIDA, junto con campos estructurados como CANTIDAD, MUNICIÓN, POLVORÍN). (2) Si los patrones determinísticos no producen match concluyente, invocación del LLM para clasificación contextual mediante *prompt* de pregunta binaria.

Esta arquitectura híbrida optimiza eficiencia: la mayoría de documentos se clasifican correctamente en la etapa determinística rápida (búsqueda de *strings*, complejidad $O(n)$ en longitud del texto), evitando invocaciones LLM costosas. Solo documentos ambiguos o con formato no estándar requieren clasificación neural. La priorización de reglas determinísticas sobre LLM refleja un principio de diseño crítico: usar el método más simple y confiable disponible, reservando capacidades avanzadas para casos genuinamente complejos.

Una vez clasificado positivamente como albarán, el sistema ejecuta extracción estructurada de campos mediante *prompt* especializado. El LLM recibe el texto completo del documento (truncado a 6500 caracteres si es muy extenso, preservando primeros 4000 y últimos 2000 caracteres para capturar tanto encabezado como firmas de autorización) y se le instruye a extraer un conjunto específico de campos en formato JSON: tipo_operacion (ENTRADA/SALIDA), origen, destino, munición, cantidad, neq_unitario, grupo_riesgo, fecha, autorizado_por.

Los campos extraídos se almacenan en `st.session_state.movimientos_pendientes`, una lista que actúa como cola de operaciones pendientes. El sistema asigna a cada movimiento detectado un identificador numérico secuencial (#1, #2, etc.) y lo presenta al usuario en un mensaje estructurado que resume los datos extraídos. El usuario puede entonces: (a) revisar el listado de movimientos pendientes con el comando `;listar movimientos`, (b) ver detalles completos de un movimiento específico con detalles movimiento #N;, o (c) ejecutar un movimiento con `;ejecutar movimiento #N;`, que lo lleva al flujo de validación y confirmación del Caso 4.

Este patrón de cola de operaciones pendientes es crítico para escalabilidad operativa. En un entorno militar real, un oficial podría escanear un lote de 20-30 albaranes al inicio de su turno, dejar que el sistema los procese y clasifique automáticamente, y luego revisar y aprobar selectivamente las operaciones detectadas. Este flujo semiautomatizado combina eficiencia de procesamiento con supervisión humana de operaciones críticas, balanceando productividad y seguridad.

Instrucciones de prueba

Para demostrar la capacidad *end-to-end* de procesamiento de albaranes, seguir este escenario completo:

- Preparar documento de prueba: Crear un PDF de albarán.
- Cargar albarán: En el panel de gestión de documentación del *sidebar*, usar el `file_uploader` para cargar el PDF de albarán. El sistema aplicará OCR si es necesario y mostrará un *spinner* durante el procesamiento.
- Observar detección automática: Si el documento se clasifica como albarán, el sistema mostrará un mensaje de notificación indicando que se ha agregado a la cola de movimientos pendientes.

Figura A-10



Figura A-10. Detección automática de documentos con movimientos de munición.

- Al escribir “*detalles movimiento #1*”; (o el número correspondiente al albarán cargado). El sistema debe mostrar todos los campos extraídos en formato estructurado: tipo de operación, origen, destino, munición, cantidad, NEQ unitario (si se encontró en RAG), grupo de riesgo (si se encontró), fecha, y persona que autorizó.
- Ejecutar movimiento: Escribir “*ejecutar movimiento #1*”. El sistema debe validar la operación (igual que en Caso 4) y presentar mensaje de confirmación detallado. Si faltaban NEQ o grupo de riesgo en el albarán original, observar cómo el sistema los recupera automáticamente de la documentación RAG (ver Figura A-11).

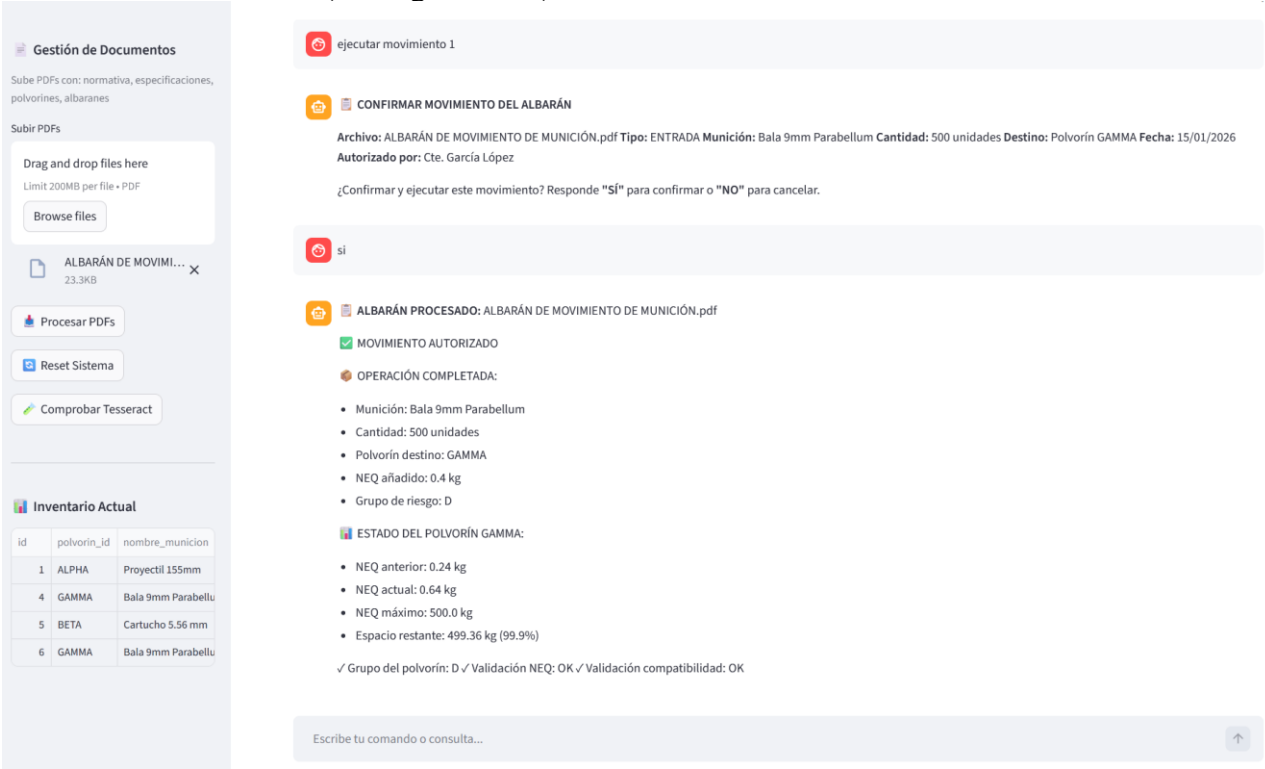


Figura A-11. Ejecutar movimiento desde documento.

- Confirmar y verificar: Responder *Sí*; para confirmar la ejecución. Observar la actualización de la tabla de inventario y el mensaje de éxito que incluye trazabilidad (archivo fuente del albarán).
- Caso de rechazo: Cuando se carga un PDF que NO sea un albarán (por ejemplo, un catálogo técnico o manual de usuario). Se observa que el sistema NO lo detecta como movimiento y simplemente lo indexa en la base vectorial para RAG.

Análisis técnico e ingeniería de prompt

El procesamiento de albaranes representa la integración más compleja del prototipo, unificando todos los componentes arquitectónicos descritos en el capítulo 4. El flujo técnico completo involucra siete etapas secuenciales:

Pipeline de procesamiento end-to-end:

1. *Extracción de texto multimodal (PyPDF2 + Tesseract OCR fallback)* 2. *Clasificación determinística rápida (regex patterns, $O(n)$)* 3. *Clasificación contextual por LLM (fallback para casos ambiguos)* 4. *Extracción estructurada de campos (prompt JSON)* 5. *Enriquecimiento vía RAG (búsqueda automática de datos técnicos)* 6. *Almacenamiento en cola de pendientes (session state)* 7. *Ejecución bajo demanda (validación + confirmación humana)*

Etapas 1 - Extracción de texto multimodal: Cuando el usuario carga un PDF, la función `procesar_y_vectorizar_pdfs()` intenta primero PyPDF2 para extracción directa. Si falla o produce menos de 100 caracteres, activa la ruta OCR: convierte cada página a imagen PNG con `dpi=300`, aplica Tesseract con configuración `--psm 6` (detectar bloque uniforme de texto), idioma `spa+eng`, y modo de salida texto. El texto de todas las páginas se concatena con delimitadores preservando estructura de paginación.

Etapas 2 - Clasificación determinística rápida: La función `es_movimiento_de_municion()` busca primero indicadores de alto *confidence score*: palabras clave en mayúsculas (ALBARÁN, GUÍA DE ENTREGA), más al menos 3 de 5 campos estructurados (POLVORÍN, CANTIDAD, etc.). Si encuentra match, retorna inmediatamente tipo de operación (ENTRADA/SALIDA/TRANSFERENCIA) sin invocar LLM. Esta detección rápida tiene ~99% recall en albaranes con formato estándar, procesándose en <1ms.

Etapas 3 - Clasificación contextual por LLM (fallback): Si la detección determinística no produce match concluyente, el sistema invoca al LLM con *prompt* de clasificación binaria:

Prompt de clasificación de documento:

;Determina si este documento es un ALBARÁN / GUÍA / ORDEN DE MOVIMIENTO de munición.\n\nIMPORTANTE: NO es un albarán si es:\n- Catálogo técnico de municiones\n- Manual de referencia\n- Documento de especificaciones\n- Normativa de seguridad\n\nDOCUMENTO:\n[texto del PDF]\n\nResponde SOLO en JSON:\n{"es_movimiento": true/false, "tipo": "ENTRADA/SALIDA/NINGUNO"};

Este *prompt* incluye contexto negativo importante, listando explícitamente casos negativos para contrarrestar el sesgo del modelo a clasificar positivamente cualquier documento que mencione municiones.

Etapas 4 - Extracción estructurada de campos: Si clasificación es positiva, el sistema invoca *prompt* de extracción que solicita JSON con 9 campos específicos. El *prompt* incluye instrucción crítica. Si falta algo, pon NULL;, permitiendo al modelo indicar ausencia de datos en lugar de inventarlos. El texto del documento se trunca inteligentemente a 6500 caracteres si es muy extenso, preservando inicio (primeros 4000) y final (últimos 2000) para capturar tanto datos operativos del encabezado como firmas de autorización del pie.

Etapa 5 - Enriquecimiento vía RAG: Si el albarán no incluía NEQ unitario o grupo de riesgo (común en albaranes operativos que asumen conocimiento previo de especificaciones), el sistema invoca automáticamente `buscar_datos_municion_en_pdf()` igual que en el Caso 4. Esta búsqueda RAG puede fallar si la documentación técnica no contiene la munición específica, en cuyo caso los campos permanecen NULL y el usuario deberá proporcionarlos manualmente durante confirmación.

Etapa 6 - Almacenamiento en cola de pendientes: El movimiento extraído (con todos sus campos, texto original completo, y nombre de archivo fuente) se añade a `st.session_state.movimientos_pendientes`. Esta estructura de datos es una lista de diccionarios persistente durante la sesión Streamlit, permitiendo procesamiento *batch*. El sistema genera notificación al usuario informando del movimiento detectado.

Etapa 7 - Ejecución bajo demanda: Cuando el usuario ejecuta `;ejecutar movimiento #N;`, el sistema recupera el diccionario correspondiente de la cola, aplica validaciones completas de polvorín (Caso 4), y presenta confirmación. Si el usuario confirma, ejecuta la transacción SQL y marca el movimiento como procesado. El mensaje de éxito incluye trazabilidad explícita citando el archivo fuente del albarán.

La ingeniería de *prompt* de clasificación merece análisis detallado. El *prompt* usa estructura de dos secciones: IMPORTANT (restricciones) y DOCUMENT (entrada). La sección IMPORTANT lista explícitamente tipos de documentos que NO son albaranes, implementando el patrón de *negative prompting*; que mejora especificidad. Sin esta lista negativa, el modelo tiende a clasificar positivamente cualquier documento técnico de municiones por generalización excesiva.

El *prompt* de extracción usa el patrón *field-by-field specification* cada campo del JSON solicitado se describe con su tipo esperado y valor NULL si falta. Esto contrarresta el comportamiento default de los LLM de completar campos faltantes con valores plausibles pero inventados. La instrucción “*Si falta algo, pon NULL*” se repite tres veces en el *prompt* —al inicio, en el cuerpo y al cierre— para maximizar su saliencia, aprovechando los efectos de primacía y recencia en el procesamiento de los modelos basados en *transformers*..

La función `extraer_json_robusto()` encargada de analizar respuestas LLM merece mención. Usa expresión regular para buscar el primer bloque de texto que comienza con “{” y termina con “}” balanceado, luego ejecuta `json.loads()`. Esta estrategia aporta resiliencia frente a respuestas que incluyen preámbulos o añadidos finales, un comportamiento habitual en modelos más pequeños o de carácter conversacional. El análisis puede fallar de manera controlada retornando *None* si no encuentra JSON válido, permitiendo al sistema solicitar reintento o presentar error descriptivo al usuario en lugar de fallar.

Los cinco casos de uso presentados en este anexo demuestran empíricamente la viabilidad del diseño híbrido RAG-SQL propuesto en el capítulo 4. El Caso 1 establece el comportamiento *baseline* y muestra que el sistema rechaza apropiadamente consultas sin contexto suficiente, priorizando confiabilidad sobre aparente utilidad. El Caso 2 valida el pipeline de ingestión multimodal que maneja tanto documentos digitales como escaneados. El Caso 3 confirma que el patrón RAG produce respuestas fundamentadas en documentación real, no en alucinaciones. El Caso 4 demuestra que las operaciones transaccionales SQL ejecutan únicamente tras validaciones exhaustivas y confirmación humana. El Caso 5 integra todos los componentes en un flujo *end-to-end* que transforma documentos administrativos en actualizaciones de inventario validadas.

En conjunto, estos casos constituyen evidencia de que el prototipo implementa efectivamente los conceptos presentados en las secciones teóricas del TFM, validando la propuesta SIGLM como solución viable para gestión logística de municiones asistida por IA, explicando las decisiones tomadas y permitiendo reproducibilidad.

ANEXO II. CÓDIGO PROTOTIPO SIGLM

```

1  import streamlit as st
2  import sqlite3
3  import pandas as pd
4  import tempfile
5  import os
6  import shutil
7  import time
8  import json
9  import re
10 from difflib import SequenceMatcher
11
12 # =====
13 # CONFIGURACIÓN E IMPORTACIONES
14 # =====
15
16 try:
17     from langchain_ollama import ChatOllama, OllamaEmbeddings
18 except ImportError:
19     from langchain_community.chat_models import ChatOllama
20     from langchain_community.embeddings import OllamaEmbeddings
21
22 from langchain_community.vectorstores import Chroma
23 from langchain_community.document_loaders import PyPDFLoader
24 from langchain_text_splitters import RecursiveCharacterTextSplitter
25 from langchain_core.tools import tool
26 from langchain_community.retrievers import BM25Retriever
27
28 # Configuración de Streamlit
29 st.set_page_config(page_title="SIGLM", layout="wide")
30
31 # Modelos LLM y embeddings
32 llm = ChatOllama(model="mistral", temperature=0, num_predict=512)
33 embeddings = OllamaEmbeddings(model="mxbai-embed-large")
34 PERSIST_DIRECTORY = "./db_vectores"
35
36 from langchain_core.documents import Document
37
38 # OCR deps
39 try:
40     import fitz # PyMuPDF
41 except Exception:
42     fitz = None
43
44 try:
45     import pytesseract
46     from PIL import Image
47     pytesseract.pytesseract.tesseract_cmd = r"C:\Program Files\Tesseract-
48 OCR\tesseract.exe"

```

```

49 except Exception:
50     pytesseract = None
51     Image = None
52
53 # =====
54 # WARMUP DEL MODELO
55 # =====
56
57 def warmup_modelo():
58     """Calienta el modelo con una consulta simple para evitar respuestas
59     vacías."""
60     intentos = 3
61     for i in range(intentos):
62         try:
63             response = llm.invoke("Responde solo: OK")
64             if hasattr(response, 'content') and response.content:
65                 return True
66             time.sleep(1)
67         except:
68             time.sleep(1)
69     return False
70
71 # Inicializar modelo (solo una vez por sesión)
72 if "model_warmed_up" not in st.session_state:
73     with st.spinner("🌀 Inicializando SIGLM..."):
74         st.session_state.model_warmed_up = warmup_modelo()
75
76 # =====
77 # BASE DE DATOS SQL
78 # =====
79
80 def init_sql_db():
81     """Inicializa la base de datos SQLite sin borrar datos existentes."""
82     conn = sqlite3.connect("arsenal_legacy.db")
83     c = conn.cursor()
84     c.execute('''CREATE TABLE IF NOT EXISTS inventario
85                 (id INTEGER PRIMARY KEY, polvorin_id TEXT, nombre_municion TEXT,
86                 cantidad INTEGER, neq_total_lote REAL, grupo_riesgo TEXT)''')
87
88     # Insertar datos de prueba SOLO si está vacío
89     c.execute("SELECT COUNT(*) FROM inventario")
90     n = c.fetchone()[0]
91     if n == 0:
92         c.execute("INSERT INTO inventario VALUES (NULL, 'ALPHA', 'Proyectil
93 155mm', 10, 105.0, 'D')")
94
95     conn.commit()
96     conn.close()
97

```

```

98 # Inicializar BD (solo una vez por sesión)
99 if "db_initialized" not in st.session_state:
100     init_sql_db()
101     st.session_state.db_initialized = True
102
103 # =====
104 # GESTIÓN DE PDFs (RAG)
105 # =====
106 def _page_needs_ocr(text: str, min_chars: int = 200, min_alnum: int = 80) -> bool:
107     """
108     Decide si una página necesita OCR.
109
110     Criterios (pensados para NO hacer OCR en PDFs con texto nativo):
111     - Si no hay texto o es muy corto -> OCR
112     - Si hay muy pocos caracteres alfanuméricos -> OCR (texto basura/ruido)
113     - Si el texto es casi todo símbolos repetidos -> OCR
114     """
115     if not text:
116         return True
117
118     t = text.strip()
119     if len(t) < min_chars:
120         return True
121
122     # Conteo de caracteres alfanuméricos (Letras/números)
123     alnum = sum(ch.isalnum() for ch in t)
124     if alnum < min_alnum:
125         return True
126
127     # Si el texto tiene muy poca "densidad" de alfanumérico, suele ser ruido
128     extraído
129     density = alnum / max(len(t), 1)
130     if density < 0.15:
131         return True
132
133     # Si hay demasiados caracteres repetidos tipo líneas/arte ASCII
134     # (esto ayuda con PDFs donde pypdf extrae un montón de '|' '_' '-' etc.)
135     non_alnum = [ch for ch in t if not ch.isalnum() and not ch.isspace()]
136     if non_alnum:
137         # proporción de símbolos sobre el total
138         sym_ratio = len(non_alnum) / max(len(t), 1)
139         if sym_ratio > 0.55:
140             return True
141
142     return False
143
144
145 def cargar_pdf_hibrido(tmp_path: str, ocr_lang: str = "spa+eng", dpi: int = 200):
146     """
147     Carga un PDF a lista de Documents.

```

```

148     - Extrae texto normal por página.
149     - Si una página no tiene texto suficiente, renderiza y aplica OCR (si
150 disponible).
151     """
152     docs = []
153     if not os.path.exists(tmp_path) or os.path.getsize(tmp_path) < 100:
154         return [Document(page_content="ERROR: PDF vacío o corrupto",
155 metadata={"source": tmp_path})]
156
157     # 1) Primero intenta PyPDFLoader (lo que ya usas) para mantener compatibilidad
158     loader = PyPDFLoader(tmp_path)
159     base_docs = loader.load()
160
161     # Si no hay OCR disponible, devolvemos lo que tengamos
162     if fitz is None or pytesseract is None:
163         return base_docs
164
165     # 2) Abrir PDF con PyMuPDF para OCR selectivo por página
166     pdf = fitz.open(tmp_path)
167
168     # Mapear por número de página: PyPDFLoader suele meter metadata {"page": n}
169     by_page = {}
170     for d in base_docs:
171         p = d.metadata.get("page", None)
172         if p is None:
173             # fallback: append sin page
174             docs.append(d)
175         else:
176             by_page[p] = d
177
178     for page_index in range(len(pdf)):
179         # Texto extraído por PyPDFLoader (si existe)
180         existing = by_page.get(page_index)
181         extracted = existing.page_content if existing else ""
182
183         if not _page_needs_ocr(extracted):
184             docs.append(existing if existing else Document(page_content=extracted,
185 metadata={"page": page_index, "source": tmp_path}))
186             continue
187
188         # 3) OCR SOLO si hace falta
189         page = pdf[page_index]
190         mat = fitz.Matrix(dpi / 72.0, dpi / 72.0)
191         pix = page.get_pixmap(matrix=mat, alpha=False)
192         img = Image.frombytes("RGB", [pix.width, pix.height], pix.samples)
193
194         # Config OCR: psm 6 suele ir bien para tablas/bloques
195         ocr_text = pytesseract.image_to_string(img, lang=ocr_lang, config="--psm
196 6")

```

```

197
198     # Si OCR sale vacío, al menos conserva lo extraído
199     final_text = (ocr_text or "").strip()
200     if len(final_text) < 10:
201         final_text = extracted
202
203     md = existing.metadata.copy() if existing else {}
204     md.update({"page": page_index, "source": tmp_path, "extraction": "ocr" if
205 ocr_text else "text"})
206     docs.append(Document(page_content=final_text, metadata=md))
207
208     pdf.close()
209     return docs
210
211 def procesar_nuevos_pdfs(uploaded_files):
212     """
213     Procesa PDFs: los añade a la BD vectorial Y detecta si son albaranes de
214 movimiento.
215     Retorna: (success, movimientos_detectados)
216     """
217     documents = []
218     movimientos_detectados = []
219
220     for uf in uploaded_files:
221         data = uf.getvalue()
222         if not data or len(data) < 100:
223             st.sidebar.error(f"PDF vacío o inválido: {uf.name}")
224             continue
225
226         with tempfile.NamedTemporaryFile(delete=False, suffix=".pdf") as tmp:
227             tmp.write(data)
228             tmp.flush()
229             os.fsync(tmp.fileno())
230             tmp_path = tmp.name
231
232         try:
233             # 1) Detectar movimiento usando el MISMO tmp_path
234             movimiento = analizar_pdf_movimiento_path(tmp_path, uf.name)
235             if movimiento:
236                 movimiento = normalizar_datos_movimiento(movimiento)
237
238                 if es_movimiento_valido(movimiento):
239                     movimientos_detectados.append({"archivo": uf.name, "info":
240 movimiento})
241             else:
242                 # Debug útil: Lo descartas por probable falso positivo
243                 st.sidebar.info(
244                     f"⚠ '{uf.name}': detección descartada (datos incompletos
245 / incoherentes)."
246                 )

```

```
247
248     # 2) Vectorizar el MISMO tmp_path
249     docs_pdf = cargar_pdf_hibrido(tmp_path, ocr_lang="spa+eng", dpi=220)
250     documents.extend(docs_pdf)
251
252     finally:
253         try:
254             os.remove(tmp_path)
255         except:
256             pass
257
258     # 3) Split y vectorstore
259     text_splitter = RecursiveCharacterTextSplitter(chunk_size=2000,
260 chunk_overlap=250)
261     splits = text_splitter.split_documents(documents)
262
263     if splits:
264         if os.path.exists(PERSIST_DIRECTORY):
265             vectorstore = Chroma(persist_directory=PERSIST_DIRECTORY,
266 embedding_function=embeddings)
267             vectorstore.add_documents(splits)
268         else:
269             vectorstore = Chroma.from_documents(splits, embeddings,
270 persist_directory=PERSIST_DIRECTORY)
271
272     return len(splits) > 0, movimientos_detectados
273
274 def extraer_json_robusto(texto: str):
275     """
276     Busca un bloque JSON en el texto, elimina Markdown y corrige errores comunes.
277     """
278     if not texto:
279         return None
280
281     # 1. Eliminar bloques de código Markdown
282     texto = texto.replace("```json", "").replace("```", "").strip()
283
284     # 2. Buscar el primer '{' y el último '}'
285     idx_inicio = texto.find('{')
286     idx_fin = texto.rfind('}')
287
288     if idx_inicio == -1 or idx_fin == -1:
289         return None
290
291     # Extraer solo la parte candidata
292     json_candidate = texto[idx_inicio : idx_fin + 1]
293
294     # 3. Intentar cargar directamente
295     try:
```

```

296         return json.loads(json_candidate)
297     except json.JSONDecodeError:
298         pass
299
300     # 4. Limpieza: Quitar comas al final de listas u objetos (error común de LLMs)
301     json_candidate = re.sub(r',\s*}', '}', json_candidate)
302     json_candidate = re.sub(r',\s*]', ']', json_candidate)
303
304     try:
305         return json.loads(json_candidate)
306     except json.JSONDecodeError:
307         return None
308
309 def analizar_pdf_movimiento_path(tmp_path: str, nombre_archivo: str = ""):
310     """
311     Analiza si un PDF (en disco) es un albarán/orden de movimiento de munición.
312     Prioriza texto nativo (PyPDFLoader) y usa OCR solo si hace falta.
313
314     MEJORA: Implementa detección refinada para evitar falsos positivos con
315     documentación técnica, catálogos y manuales.
316     """
317     # 1) Primero intenta texto nativo (como tu versión original)
318     try:
319         base_docs = PyPDFLoader(tmp_path).load()
320         texto_base = "\n\n".join([d.page_content for d in base_docs]).strip()
321     except Exception:
322         texto_base = ""
323
324     # 2) Si el texto nativo es pobre, usa híbrido (OCR selectivo)
325     if texto_base and len(texto_base) > 300:
326         texto_completo = texto_base
327     else:
328         try:
329             docs = cargar_pdf_hibrido(tmp_path, ocr_lang="spa+eng", dpi=260)
330             texto_completo = "\n\n".join([d.page_content for d in docs]).strip()
331         except Exception:
332             return None
333
334     if len(texto_completo) < 80:
335         return None
336
337     t = re.sub(r"\s+", " ", texto_completo.upper())
338
339     # 3) EXCLUSIÓN TEMPRANA: Descartar documentación técnica/catálogos
340     # Estos documentos tienen patrones estructurales característicos
341     indicadores_catalogo = [
342         r"\bCATALOGO\b", r"\bCATALOGO TECNICO\b",
343         r"\bMANUAL TECNICO\b", r"\bMANUAL DE REFERENCIA\b",
344         r"\bFICHA TECNICA\b", r"\bFICHA DE ESPECIFICACIONES\b",
345         r"\bESPECIFICACIONES TECNICAS\b",

```

```

346     r"\bDOCUMENTO DE REFERENCIA\b",
347     r"\bNORMATIVA (OTAN|AASTP|STANAG)\b",
348     r"\bVIDA UTIL.*ALMACENAMIENTO\b",
349     r"\bCLASIFICACION.*GRUPOS.*RIESGO\b",
350     r"\bCONDICIONES DE EMBALAJE\b",
351     r"\bDESCRIPCION TECNICA DE COMPONENTES\b"
352 ]
353
354 if any(re.search(patron, t) for patron in indicadores_catalogo):
355     # Es documentación técnica, no un albarán
356     return None
357
358 # 4) Patrones ESPECÍFICOS de albaranes (más restrictivos)
359 # Un albarán real debe tener estructura operacional, no solo menciones
360 genéricas
361 patrones_albaran_fuerte = [
362     # Número de albarán/documento
363     r"\b(ALBARAN|ALBARAN)\s+(N[UO]|NUM|NUMERO)[:.\\s]*[0-9]",
364     r"\bORDEN\s+DE\s+MOVIMIENTO\s+(N[UO]|NUM)[:.\\s]*[0-9]",
365     r"\bGUIA\s+DE\s+CIRCULACION\s+(N[UO]|NUM)[:.\\s]*[0-9]",
366
367     # Operaciones de movimiento con contexto
368     r"\b(OPERACION|OPERACION)\s+DE\s+(ENTRADA|SALIDA|TRANSFERENCIA)\b",
369     r"\bTIPO\s+DE\s+(OPERACION|OPERACION)[:.\\s]*(ENTRADA|SALIDA|TRANSFERENCIA)
370 \b",
371
372     # Origen y destino ESPECÍFICOS (polvorines/arsenales)
373     r"\b(ORIGEN|PROCEDENCIA)[:.\\s]*POLVORIN\s+[A-Z0-
374 9]+.*\bDESTINO[:.\\s]*POLVORIN\s+[A-Z0-9]+\b",
375     r"\bPOLVORIN\s+ORIGEN[:.\\s]*[A-Z0-9]+.*\bPOLVORIN\s+DESTINO[:.\\s]*[A-Z0-
376 9]+\b",
377
378     # Autorización con firma y fecha
379     r"\b(AUTORIZADO|AUTORIZA)\s+POR[:.\\s]*[A-Z]+.*\b(FECHA|DIA)[:.\\s]*[0-
380 9]{1,2}[/-][0-9]{1,2}[/-][0-9]{2,4}\b",
381     r"\bFIRMA[:.\\s]*.*\bFECHA[:.\\s]*[0-9]{1,2}[/-][0-9]{1,2}[/-][0-9]{2,4}\b"
382 ]
383
384 hay_patron_fuerte = any(re.search(p, t) for p in patrones_albaran_fuerte)
385
386 # 5) Keywords operacionales (solo si hay patrón fuerte)
387 # Estas deben aparecer en contexto de movimiento, no de especificaciones
388 técnicas
389 claves_operacionales = [
390     "ALBARAN DE ENTRADA",
391     "ALBARAN DE SALIDA",
392     "ORDEN DE MOVIMIENTO",
393     "GUIA DE CIRCULACION",
394     "TRANSFERENCIA DE MUNICION"

```

```

395     ]
396     hits_operacionales = [k for k in claves_operacionales if k in t]
397
398     # 6) NUEVA LÓGICA: Requerimos AMBOS criterios
399     # - Al menos UN patrón fuerte (estructura de albarán real)
400     # - Al menos UNA keyword operacional específica
401     hay_indicios = hay_patron_fuerte and len(hits_operacionales) >= 1
402
403     # Debug opcional (descomenta para depuración)
404     # st.sidebar.write(f"Debug '{nombre_archivo}':")
405     # st.sidebar.write(f" - Patrón fuerte: {hay_patron_fuerte}")
406     # st.sidebar.write(f" - Keywords ops: {hits_operacionales}")
407     # st.sidebar.write(f" - Es albarán: {hay_indicios}")
408
409     # Si hay indicios claros, no dejes que el LLM te lo niegue por "timidez"
410     if hay_indicios:
411         tipo = "TRANSFERENCIA"
412         if "ENTRADA" in t and "SALIDA" not in t:
413             tipo = "ENTRADA"
414         elif "SALIDA" in t and "ENTRADA" not in t:
415             tipo = "SALIDA"
416
417     # Aún así intentamos extracción con LLM (para rellenar campos), pero ya
418     "es movimiento"
419     contexto_llm = (texto_completo[:4000] + "\n\n...\n\n" + texto_completo[-
420     2000:]) if len(texto_completo) > 6500 else texto_completo
421
422     prompt_extraccion = f""""Extrae los datos del movimiento. Si falta algo,
423     pon NULL.
424
425     DOCUMENTO:
426     {contexto_llm}
427
428     Responde SOLO en JSON:
429     {{
430         "tipo_operacion": "ENTRADA/SALIDA",
431         "origen": "NOMBRE_O_NULL",
432         "destino": "NOMBRE_O_NULL",
433         "municion": "NOMBRE_COMPLETO",
434         "cantidad": "NUMERO",
435         "neq_unitario": "NUMERO_O_NULL",
436         "grupo_riesgo": "LETRA_O_NULL",
437         "fecha": "FECHA_O_NULL",
438         "autorizado_por": "NOMBRE_O_NULL"
439     }}
440     """
441
442     try:
443         resp = invocar_llm_seguro(prompt_extraccion, reintentos=2)
444         datos = extraer_json_robusto(resp) or {}
445     except Exception:

```

```

445         datos = {}
446
447     return {
448         "es_movimiento": True,
449         "tipo": tipo,
450         "datos": datos,
451         "texto_original": texto_completo
452     }
453
454     # 7) Si NO hay indicios claros, entonces sí: pregunta al LLM si es movimiento
455     # (Este caso solo se ejecuta si los patrones deterministas no dieron match)
456     contexto_llm = (texto_completo[:4000] + "\n\n...\n\n" + texto_completo[-
457 2000:]) if len(texto_completo) > 6500 else texto_completo
458
459     prompt_deteccion = f"""Determina si este documento es un ALBARÁN / GUÍA /
460 ORDEN DE MOVIMIENTO de munición.
461
462 IMPORTANTE: NO es un albarán si es:
463 - Catálogo técnico de municiones
464 - Manual de referencia
465 - Documento de especificaciones
466 - Normativa de seguridad
467
468 DOCUMENTO:
469 {contexto_llm}
470
471 Responde SOLO en JSON:
472 {{ "es_movimiento": true/false, "tipo": "ENTRADA/SALIDA/TRANSFERENCIA/NINGUNO" }}
473 """
474
475     try:
476         respuesta = invocar_llm_seguro(prompt_deteccion, reintentos=2)
477         deteccion = extraer_json_robusto(respuesta)
478         if not deteccion:
479             return None
480         if not deteccion.get("es_movimiento"):
481             return None
482
483     # extracción
484     prompt_extraccion = f"""Extrae los datos del movimiento. Si falta algo,
485 pon NULL.
486
487 DOCUMENTO:
488 {contexto_llm}
489
490 Responde SOLO en JSON:
491 {{
492     "tipo_operacion": "ENTRADA/SALIDA",
493     "origen": "NOMBRE_O_NULL",
494     "destino": "NOMBRE_O_NULL",

```

```
494     "municion": "NOMBRE_COMPLETO",
495     "cantidad": "NUMERO",
496     "neq_unitario": "NUMERO_O_NULL",
497     "grupo_riesgo": "LETRA_O_NULL",
498     "fecha": "FECHA_O_NULL",
499     "autorizado_por": "NOMBRE_O_NULL"
500 }
501 """
502     resp2 = invocar_llm_seguro(prompt_extraccion, reintentos=2)
503     datos = extraer_json_robusto(resp2) or {}
504
505     return {
506         "es_movimiento": True,
507         "tipo": deteccion.get("tipo"),
508         "datos": datos,
509         "texto_original": texto_completo
510     }
511 except Exception:
512     return None
513
514 def obtener_vectorstore():
515     """
516     Obtiene el vectorstore de ChromaDB.
517     Cachea en session_state para evitar múltiples conexiones abiertas.
518     """
519     PERSIST_DIRECTORY = "./db_vectores"
520
521     # Si no existe el directorio, retornar None
522     if not os.path.exists(PERSIST_DIRECTORY):
523         if 'vectorstore' in st.session_state:
524             st.session_state.vectorstore = None
525         return None
526
527     # Si ya existe una instancia cacheada, usarla
528     if 'vectorstore' in st.session_state and st.session_state.vectorstore is not
529 None:
530         return st.session_state.vectorstore
531
532     # Crear nueva instancia y cachearla
533     try:
534         from langchain_ollama import OllamaEmbeddings
535         embeddings = OllamaEmbeddings(model="mxbai-embed-large")
536         vs = Chroma(persist_directory=PERSIST_DIRECTORY,
537 embedding_function=embeddings)
538         st.session_state.vectorstore = vs
539         return vs
540     except Exception as e:
541         st.error(f"Error al cargar vectorstore: {e}")
542         return None
543
```

```

544 # =====
545 # FUNCIONES AUXILIARES
546 # =====
547
548 def invocar_llm_seguro(prompt_text, reintentos=3):
549     """
550     Invoca el LLM con reintentos automáticos para manejar respuestas vacías.
551     """
552     from langchain_ollama import ChatOllama
553     import time
554
555     llm = ChatOllama(model="mistral", temperature=0, num_predict=512)
556
557     for intento in range(reintentos):
558         try:
559             response = llm.invoke(prompt_text)
560             if hasattr(response, 'content') and response.content and
561 response.content.strip():
562                 return response.content.strip()
563             if intento < reintentos - 1:
564                 time.sleep(0.5)
565         except Exception as e:
566             if intento < reintentos - 1:
567                 time.sleep(0.5)
568             else:
569                 return f"Error: {str(e)}"
570     return "No se pudo obtener respuesta del modelo."
571
572 def buscar_datos_municion_en_pdf(nombre_municion):
573     """
574     Busca NEQ unitario y grupo de riesgo de una munición en los PDFs cargados.
575     Retorna dict con 'neq', 'grupo' y 'nombre' si encuentra, None si no.
576     """
577     vs = obtener_vectorstore()
578     if not vs:
579         return None
580
581     # Generar variantes del nombre
582     nombre_base = normalizar_nombre_municion(nombre_municion)
583     nombre_con_punto = nombre_municion.replace(',', '.', '')
584     nombre_base_con_punto = nombre_base.replace(',', '.', '')
585
586     variantes_búsqueda = [
587         nombre_municion,
588         nombre_con_punto,
589         nombre_base,
590         nombre_base_con_punto,
591         nombre_municion.replace(" ", ""),
592         nombre_con_punto.replace(" ", ""),

```

```

593         re.sub(r'\b(de|del|la|el)\b', '', nombre_municion,
594         flags=re.IGNORECASE).strip(),
595         re.sub(r'\b(de|del|la|el)\b', '', nombre_con_punto,
596         flags=re.IGNORECASE).strip(),
597     ]
598
599     # Variantes con unidades expandidas
600     if "mm" in nombre_municion.lower():
601         variantes_busqueda.append(nombre_municion.replace("mm", " milímetros"))
602         variantes_busqueda.append(nombre_con_punto.replace("mm", " milímetros"))
603
604     # Eliminar duplicados
605     variantes_busqueda = list(dict.fromkeys(variantes_busqueda))
606
607     st.sidebar.markdown("----")
608     st.sidebar.markdown("### 🔍 DEBUG: Búsqueda RAG")
609     st.sidebar.info(f"Buscando: '{nombre_municion}'")
610     st.sidebar.info(f"Variantes: {len(variantes_busqueda)}")
611
612     mejor_resultado = None
613     mejor_score = 0
614
615     for idx, variante in enumerate(variantes_busqueda, 1):
616         st.sidebar.markdown(f"**Intento {idx}/{len(variantes_busqueda)}:**")
617         st.sidebar.markdown(f`{variante}`")
618
619         # Múltiples queries para aumentar recall
620         queries = [
621             f"{variante}",
622             f"NEQ {variante}",
623             f"grupo de riesgo {variante}",
624             f"especificaciones {variante}",
625         ]
626
627         for query in queries:
628             try:
629                 docs = vs.similarity_search(query, k=8)
630
631                 if not docs:
632                     continue
633
634                 context = "\n\n".join([d.page_content for d in docs])
635
636                 # DEBUG: Mostrar si el contexto contiene palabras clave
637                 palabras_clave = ['neq', 'grupo', 'riesgo', '5.56', '5,56',
638                 'cartucho']
639                 encontradas = [p for p in palabras_clave if p in context.lower()]
640                 if encontradas:
641                     st.sidebar.success(f"✓ Contexto tiene: {'',
642                     '.join(encontradas)}")

```

```

643
644     # PROMPT SIMPLIFICADO Y MÁS DIRECTO
645     prompt = f"""Busca en el texto información sobre "{variante}".
646
647     TEXTO:
648     {context[:3000]}
649
650     TAREA:
651     Extrae NEQ (en kg) y GRUPO DE RIESGO.
652
653     FORMATO DE RESPUESTA (una línea):
654     NEQ=0.0018 GRUPO=1.45
655
656     Si no encuentras, responde: NO_ENCONTRADO
657
658     RESPUESTA: """
659
660     respuesta = invocar_llm_seguro(prompt)
661
662     st.sidebar.text(f"LLM: {respuesta[:100]}")
663
664     # PARSING MÁS ROBUSTO
665     neq_val = None
666     grupo_val = None
667
668     # Intento 1: Formato esperado NEQ=X GRUPO=Y
669     match_neq = re.search(r'NEQ[=:\\s]+([\\d.]+)', respuesta,
670 re.IGNORECASE)
671     match_grupo = re.search(r'GRUPO[=:\\s]+([A-Z0-9.]+)', respuesta,
672 re.IGNORECASE)
673
674     if match_neq and match_grupo:
675         try:
676             neq_val = float(match_neq.group(1))
677             grupo_val = match_grupo.group(1).upper()
678         except:
679             pass
680
681     # Intento 2: Buscar directamente en respuesta "0.0018" y "1.45"
682     if not neq_val:
683         # Buscar NEQ típico (0.XXXX)
684         match_neq_alt = re.search(r'\\b0\\.\\d{3,4}\\b', respuesta)
685         if match_neq_alt:
686             try:
687                 neq_val = float(match_neq_alt.group(0))
688             except:
689                 pass
690
691     if not grupo_val:

```

```

692         # Buscar grupo típico (1.4S, D, etc)
693         match_grupo_alt = re.search(r'\b(1\.[1-4][A-Z]|[A-S])\b',
694 respuesta)
695         if match_grupo_alt:
696             grupo_val = match_grupo_alt.group(1).upper()
697
698         # Si encontramos ambos datos
699         if neq_val and grupo_val:
700             # Score de confianza
701             score = sum(1 for v in variantes_busqueda if v.lower() in
702 context.lower())
703             score += 2 # Bonus por encontrar datos
704
705             st.sidebar.success(f"✅ ENCONTRADO: NEQ={neq_val},
706 GRUPO={grupo_val}")
707
708             if score > mejor_score:
709                 mejor_score = score
710                 mejor_resultado = {
711                     'neq': neq_val,
712                     'grupo': grupo_val,
713                     'nombre': nombre_municion
714                 }
715
716             # Si alta confianza, terminar
717             if score >= 4:
718                 st.sidebar.success(f"🚀 Alta confianza
719 (score={score}), terminando búsqueda")
720                 return mejor_resultado
721
722             except Exception as e:
723                 st.sidebar.error(f"❌ Error: {str(e)[:80]}")
724                 continue
725
726         if mejor_resultado:
727             st.sidebar.success(f"🚀 Resultado final: NEQ={mejor_resultado['neq']} kg,
728 Grupo {mejor_resultado['grupo']}")
729         else:
730             st.sidebar.error(f"❌ No se encontraron datos para '{nombre_municion}'")
731             st.sidebar.info("Variantes probadas: " + ", ".join(variantes_busqueda[:3])
732 + "...")
733
734         return mejor_resultado
735
736 def validar_polvorin(polvorin_id, neq_total_resultante):
737     vs = obtener_vectorstore()
738     if not vs:
739         return {
740             "valido": False,
741

```

```

742         "mensaje": "⚠ No hay documentos de polvorines cargados. No se puede
743 validar la operación."
744     }
745
746     retriever = vs.as_retriever(search_kwargs={"k": 10})
747     queries = [
748         f"POLVORÍN {polvorin_id}",
749         f"{polvorin_id} NEQ máximo",
750         f"Polvorín {polvorin_id} capacidad",
751         f"características {polvorin_id}"
752     ]
753
754     all_docs = []
755     for query in queries:
756         try:
757             all_docs.extend(retriever.invoke(query))
758         except:
759             pass
760
761     if not all_docs:
762         return {
763             "valido": False,
764             "mensaje": f"⊖ El polvorín '{polvorin_id}' NO está registrado en la
765 documentación.\n\nPara ver los polvorines disponibles, pregunta: 'lista de
766 polvorines'"
767         }
768
769     unique_docs = []
770     seen = set()
771     for d in all_docs:
772         if d.page_content not in seen:
773             unique_docs.append(d)
774             seen.add(d.page_content)
775             if len(unique_docs) >= 5:
776                 break
777
778     context = "\n\n".join([d.page_content for d in unique_docs])
779
780     # =====
781     # 1) INTENTO DETERMINISTA (regex)
782     # =====
783     # Busca un número cercano a "NEQ" y/o "máximo"
784     # (tolerante con comas decimales)
785     m = re.search(r"(NEQ[^\0-9]{0,60})(\d+(?:[.]\d+)?)", context, re.IGNORECASE)
786     if not m:
787         m = re.search(r"(M[ÁA]XIM[OA][^\0-9]{0,60})(\d+(?:[.]\d+)?)", context,
788 re.IGNORECASE)
789
790     neq_maximo = None

```

```

791     if m:
792         try:
793             neq_maximo = float(m.group(2).replace(",", "."))
794         except:
795             neq_maximo = None
796
797         # Si sacamos número por regex, asumimos que existe (porque encontramos
798         contexto del polvorín)
799         if neq_maximo is not None and neq_maximo > 0:
800             if neq_total_resultante > neq_maximo:
801                 return {
802                     "valido": False,
803                     "neq_maximo": neq_maximo,
804                     "mensaje": f"⊖ OPERACIÓN DENEGADA\n\nNEQ resultante
805 ({neq_total_resultante:.2f} kg) SUPERA el límite del polvorín {polvorin_id}
806 ({neq_maximo:.2f} kg máximo).\n\nDiferencia: +{neq_total_resultante -
807 neq_maximo:.2f} kg"
808                 }
809             return {
810                 "valido": True,
811                 "neq_maximo": neq_maximo,
812                 "mensaje": f"✅ Polvorín {polvorin_id} validado (NEQ máximo:
813 {neq_maximo:.2f} kg)"
814             }
815
816         # =====
817         # 2) FALLBACK LLM (si regex falla)
818         # =====
819         prompt = f""Analiza la documentación del polvorín {polvorin_id}:
820
821 DOCUMENTACIÓN:
822 {context}
823
824 Extrae EXACTAMENTE:
825 1. ¿Existe el polvorín {polvorin_id}? (true/false)
826 2. NEQ máximo permitido (número en kg)
827
828 Responde SOLO en JSON (sin texto extra):
829 {"existe": true/false, "neq_maximo": "NUMERO"}}
830
831 Si NO encuentras el polvorín, responde:
832 {"existe": false, "neq_maximo": "0"}}""
833
834     try:
835         respuesta = invocar_llm_seguro(prompt, reintentos=2)
836         respuesta_limpia = respuesta.replace("```json", "").replace("```",
837 "").strip()
838
839         # Extraer el primer JSON si viene con texto extra
840         jm = re.search(r"\{.*\}", respuesta_limpia, re.DOTALL)

```

```

841         if not jm:
842             raise ValueError("No JSON en respuesta del LLM")
843
844         datos = json.loads(jm.group(0))
845
846         if not datos.get("existe"):
847             return {
848                 "valido": False,
849                 "mensaje": f"⊖ El polvorín '{polvorin_id}' NO existe en el
850 sistema.\n\nConsulta 'lista de polvorines' para ver los disponibles."
851             }
852
853         neq_maximo = float(str(datos.get("neq_maximo", "0")).replace(",","."))
854         if neq_total_resultante > neq_maximo:
855             return {
856                 "valido": False,
857                 "neq_maximo": neq_maximo,
858                 "mensaje": f"⊖ OPERACIÓN DENEGADA\n\nNEQ resultante
859 ({neq_total_resultante:.2f} kg) SUPERA el límite del polvorín {polvorin_id}
860 ({neq_maximo:.2f} kg máximo).\n\nDiferencia: +{neq_total_resultante -
861 neq_maximo:.2f} kg"
862             }
863
864         return {
865             "valido": True,
866             "neq_maximo": neq_maximo,
867             "mensaje": f"☑ Polvorín {polvorin_id} validado (NEQ máximo:
868 {neq_maximo:.2f} kg)"
869         }
870
871     except Exception as e:
872         return {
873             "valido": False,
874             "mensaje": f"✗ Error al validar polvorín: {str(e)}"
875         }
876
877 def normalizar_nombre_municion(nombre: str) -> str:
878     """Normaliza nombres para búsqueda flexible."""
879     import re
880
881     # Protección contra None
882     if nombre is None or not nombre:
883         return ""
884
885     nombre = str(nombre).lower().strip()
886
887     # Normalizar variaciones de unidades
888     nombre = re.sub(r'\b(milímetros?|milímetros?)\b', 'mm', nombre)
889
890     # Normalizar espacios alrededor de números (9 mm -> 9mm)

```

```

890     nombre = re.sub(r'(\d+)\s+(mm|cm|kg|g)\b', r'\1\2', nombre)
891
892     # Eliminar palabras de relleno
893     palabras_eliminar = ['de', 'del', 'la', 'el', 'un', 'una', 'para']
894     for palabra in palabras_eliminar:
895         nombre = re.sub(r'\b' + palabra + r'\b', '', nombre)
896
897     # Consolidar espacios múltiples
898     nombre = re.sub(r'\s+', ' ', nombre).strip()
899
900     # Manejo inteligente de plurales
901     if nombre.endswith('es'):
902         nombre = nombre[:-2]
903     elif nombre.endswith('s') and not nombre.endswith('ss'):
904         nombre = nombre[:-1]
905
906     return nombre
907
908 def normalizar_grupo_riesgo(grupo: str) -> str:
909     """
910     Normaliza grupos de riesgo a formato estándar para comparación.
911
912     Convierte:
913     - Letra simple (E, D, S) → Extrae letra de grupo extendido si es compatible
914     - Grupo extendido (1.2E, 1.4S) → Extrae solo la letra final
915
916     Esto permite comparar "E" con "1.2E" correctamente.
917     """
918     if not grupo:
919         return ""
920
921     grupo = str(grupo).strip().upper()
922
923     # Si es formato extendido (1.2E, 1.4S, etc.), extraer letra final
924     if '.' in grupo:
925         # Extraer última letra
926         letra = grupo.split('.')[1][-1] if grupo.split('.')[1] else ""
927         if letra.isalpha():
928             return letra
929
930     # Si es letra simple, devolver tal cual
931     if len(grupo) == 1 and grupo.isalpha():
932         return grupo
933
934     # Si no se puede normalizar, devolver original
935     return grupo
936
937 def similitud_nombres_municion(nombre1: str, nombre2: str) -> float:
938     """Calcula similitud entre nombres de munición [0.0-1.0]."""
939     # Protección contra None

```

```

940     if nombre1 is None or nombre2 is None:
941         return 0.0
942
943     norm1 = normalizar_nombre_municion(nombre1)
944     norm2 = normalizar_nombre_municion(nombre2)
945
946     # Protección contra strings vacíos
947     if not norm1 or not norm2:
948         return 0.0
949
950     if norm1 == norm2:
951         return 1.0
952
953     if norm1 in norm2 or norm2 in norm1:
954         return 0.9
955
956     return SequenceMatcher(None, norm1, norm2).ratio()
957
958 def combinar_resultados_hibridos(docs_bm25, docs_semantic, weight_bm25=0.7,
959 weight_semantic=0.3):
960     """
961     Combina resultados de BM25 y búsqueda semántica con pesos.
962     Elimina duplicados y ordena por score combinado.
963     """
964     from collections import defaultdict
965
966     # Diccionario para acumular scores por documento
967     doc_scores = defaultdict(float)
968     doc_objects = {}
969
970     # Agregar scores de BM25
971     for i, doc in enumerate(docs_bm25):
972         # Score decreciente según posición (más alto = mejor)
973         score = (len(docs_bm25) - i) / len(docs_bm25) * weight_bm25
974         doc_id = doc.page_content[:100] # Usar primeros 100 chars como ID único
975         doc_scores[doc_id] += score
976         doc_objects[doc_id] = doc
977
978     # Agregar scores de búsqueda semántica
979     for i, doc in enumerate(docs_semantic):
980         score = (len(docs_semantic) - i) / len(docs_semantic) * weight_semantic
981         doc_id = doc.page_content[:100]
982         doc_scores[doc_id] += score
983         if doc_id not in doc_objects:
984             doc_objects[doc_id] = doc
985
986     # Ordenar por score combinado (mayor a menor)
987     sorted_doc_ids = sorted(doc_scores.keys(), key=lambda x: doc_scores[x],
988 reverse=True)

```

```

989
990     # Retornar documentos ordenados
991     return [doc_objects[doc_id] for doc_id in sorted_doc_ids]
992
993     # =====
994     # HERRAMIENTAS (TOOLS)
995     # =====
996
997 @tool
998 def consultar_inventario(filtro: str = ""):
999     """
1000     Consulta el inventario actual de munición en la base de datos SQL.
1001     Puede filtrar por polvorín, tipo de munición o grupo de riesgo.
1002
1003     CAMBIO CLAVE: Ahora usa GROUP BY para agregar múltiples registros
1004     del mismo tipo de munición en el mismo polvorín.
1005
1006     ANTES: Mostraba cada registro por separado (confuso cuando hay duplicados)
1007     AHORA: Agrupa y suma cantidades/NEQ de registros del mismo tipo
1008     """
1009     conn = sqlite3.connect("arsenal_legacy.db")
1010
1011     if filtro:
1012         # Consulta CON filtro y con GROUP BY
1013         query = """
1014         SELECT
1015             polvorin_id,
1016             nombre_municion,
1017             SUM(cantidad) as cantidad,          -- Suma cantidades de
1018     múltiples registros
1019             SUM(neq_total_lote) as neq_total_lote, -- Suma NEQ de múltiples
1020     registros
1021             MAX(grupo_riesgo) as grupo_riesgo    -- Toma uno (deben ser iguales
1022     para el mismo tipo)
1023         FROM inventario
1024         WHERE polvorin_id LIKE ? OR nombre_municion LIKE ? OR grupo_riesgo LIKE ?
1025         GROUP BY polvorin_id, nombre_municion -- ← CAMBIO CLAVE: Agrupa
1026     duplicados
1027         """
1028         df = pd.read_sql_query(query, conn, params=(f"%{filtro}%", f"%{filtro}%",
1029     f"%{filtro}%"))
1030     else:
1031         # Consulta SIN filtro pero con GROUP BY
1032         query = """
1033         SELECT
1034             polvorin_id,
1035             nombre_municion,
1036             SUM(cantidad) as cantidad,
1037             SUM(neq_total_lote) as neq_total_lote,
1038             MAX(grupo_riesgo) as grupo_riesgo

```

```

1039         FROM inventario
1040         GROUP BY polvorin_id, nombre_municion -- ← CAMBIO CLAVE: Agrupa
1041 duplicados
1042         """
1043         df = pd.read_sql_query(query, conn)
1044
1045         conn.close()
1046
1047         if df.empty:
1048             return "No se encontraron registros en el inventario."
1049
1050         # Formatear resultado (esto NO cambia)
1051         resultado = "📊 INVENTARIO ACTUAL:\n\n"
1052         for _, row in df.iterrows():
1053             resultado += f"◇ Polvorín: {row['polvorin_id']}\n"
1054             resultado += f"• Munición: {row['nombre_municion']}\n"
1055             resultado += f"• Cantidad: {row['cantidad']} unidades\n" # Ya sumada
1056         si había duplicados
1057             resultado += f"• NEQ Total: {row['neq_total_lote']} kg\n"
1058             resultado += f"• Grupo Riesgo: {row['grupo_riesgo']}\n\n"
1059
1060         total_neq = df['neq_total_lote'].sum()
1061         total_unidades = df['cantidad'].sum()
1062         resultado += f"📈 TOTALES:\n    • Total NEQ: {total_neq} kg\n    • Total
1063 unidades: {total_unidades}\n"
1064
1065         return resultado
1066
1067 @tool
1068 def consultar_capacidad_polvorin(polvorin_id: str):
1069     """
1070     Consulta la capacidad actual y disponible de un polvorín.
1071     Combina datos del PDF (NEQ máximo) con datos SQL (NEQ actual).
1072     """
1073     # 1. Obtener NEQ máximo del PDF
1074     validacion = validar_polvorin(polvorin_id, 0) # Usar 0 para solo obtener info
1075
1076     if not validacion.get("valido") and "neq_maximo" not in validacion:
1077         return f"⚠️ No se encontró información del polvorín '{polvorin_id}' en la
1078 documentación."
1079
1080     neq_maximo = validacion.get("neq_maximo", 0)
1081
1082     if neq_maximo == 0:
1083         # Intentar extraer de otra forma
1084         vs = obtener_vectorstore()
1085         if not vs:
1086             return f"⚠️ No hay documentos cargados para consultar el NEQ máximo."
1087

```

```

1088         resultado = consultar_documentacion_tecnica.invoke({
1089             "pregunta": f"¿Cuál es el NEQ máximo del polvorín {polvorin_id}?
1090 Responde solo el número en kg."
1091         })
1092
1093     try:
1094         neq_maximo = float(re.search(r'(\d+(?:\.\d+)?)', resultado).group(1))
1095     except:
1096         return f"✗ No se pudo determinar el NEQ máximo del polvorín
1097 {polvorin_id}."
1098
1099     # 2. Obtener NEQ actual del SQL
1100     conn = sqlite3.connect("arsenal_legacy.db")
1101     cursor = conn.cursor()
1102     res = cursor.execute(
1103         "SELECT SUM(neq_total_lote) FROM inventario WHERE polvorin_id = ?",
1104         (polvorin_id,)
1105     ).fetchone()
1106     conn.close()
1107
1108     neq_actual = res[0] if res and res[0] else 0.0
1109     neq_disponible = neq_maximo - neq_actual
1110     porcentaje_ocupacion = (neq_actual / neq_maximo * 100) if neq_maximo > 0 else
1111 0
1112
1113     # 3. Obtener detalle de munición actual
1114     conn = sqlite3.connect("arsenal_legacy.db")
1115     df = pd.read_sql_query(
1116         "SELECT nombre_municion, cantidad, neq_total_lote, grupo_riesgo FROM
1117 inventario WHERE polvorin_id = ?",
1118         conn,
1119         params=(polvorin_id,)
1120     )
1121     conn.close()
1122
1123     # 4. Formatear respuesta
1124     resultado = f"""\n📦 CAPACIDAD DEL POLVORÍN {polvorin_id}
1125
1126 ✧ NEQ Máximo: {neq_maximo} kg
1127 ✧ NEQ Actual: {neq_actual:.2f} kg
1128 ✧ NEQ Disponible: {neq_disponible:.2f} kg
1129 ✧ Ocupación: {porcentaje_ocupacion:.1f}%
1130
1131 """"
1132
1133     if not df.empty:
1134         resultado += "\n📦 CONTENIDO ACTUAL:\n"
1135         for _, row in df.iterrows():
1136             resultado += f"    • {row['nombre_municion']}: {row['cantidad']}
1137 unidades ({row['neq_total_lote']:.2f} kg, Grupo {row['grupo_riesgo']})\n"

```

```

1138     else:
1139         resultado += "📦 Polvorín VACÍO\n"
1140
1141     return resultado
1142
1143 @tool
1144 def consultar_documentacion_tecnica(pregunta: str):
1145     """
1146     Consulta la documentación técnica con búsqueda híbrida BM25 + Embedding.
1147     BM25 captura keywords exactas (números, códigos NSN), embedding captura
1148     semántica.
1149     """
1150     import re
1151
1152     vs = obtener_vectorstore()
1153     if not vs:
1154         return "❌ No hay documentos cargados. Sube PDFs en la sección
1155 'Documentos'."
1156
1157     # Extraer números de la pregunta
1158     numeros = re.findall(r'\b\d+\b', pregunta)
1159     # =====
1160     # BÚSQUEDA HÍBRIDA: BM25 (keyword) + Embedding (semántico)
1161     # =====
1162
1163     # 1. Obtener TODOS los documentos del vector store para BM25
1164     try:
1165         total_docs = vs._collection.count()
1166         st.sidebar.info(f"📄 Total docs en BD: {total_docs}")
1167
1168         # Recuperar todos los docs (limitado a 500 para performance)
1169         all_docs = vs.similarity_search("", k=min(total_docs, 500))
1170
1171     except Exception as e:
1172         st.sidebar.warning(f"⚠️ No se pudo contar docs: {e}")
1173         all_docs = vs.similarity_search("", k=200)
1174
1175     # 1. Búsqueda BM25 (keyword-based)
1176     bm25_retriever = BM25Retriever.from_documents(all_docs)
1177     bm25_retriever.k = 15
1178     docs_bm25 = bm25_retriever.invoke(pregunta)
1179
1180     # 2. Búsqueda semántica (embedding)
1181     semantic_retriever = vs.as_retriever(search_kwargs={"k": 15})
1182     docs_semantic = semantic_retriever.invoke(pregunta)
1183
1184     # 3. Combinar resultados manualmente (70% BM25, 30% semántico)
1185     docs_raw = combinar_resultados_hibridos(docs_bm25, docs_semantic, 0.7, 0.3)
1186     # Mostrar DEBUGGING en sidebar

```

```

1187     st.sidebar.markdown("---")
1188     st.sidebar.markdown("### 🔍 DEBUG RAG")
1189     st.sidebar.info(f"📄 Búsqueda: '{pregunta}')"
1190     st.sidebar.info(f"🔢 Números detectados: {numeros if numeros else 'ninguno'}")
1191     st.sidebar.info(f"📊 BM25 encontró: {len(docs_bm25)}")
1192     st.sidebar.info(f"📊 Embedding encontró: {len(docs_semantic)}")
1193     st.sidebar.info(f"🔗 Método: BM25 (70%) + Embedding (30%)")
1194     st.sidebar.info(f"📄 Docs híbridos: {len(docs_raw)}")
1195
1196     # Tomar top 8 de Los resultados híbridos
1197     docs = docs_raw[:8]
1198     st.sidebar.info(f"📄 Documentos finales: {len(docs)}")
1199
1200     if not docs:
1201         st.sidebar.error("❌ No se recuperaron documentos")
1202         return f"❌ No se encontró información sobre '{pregunta}' en los
1203 documentos cargados.\n\nAsegúrate de haber subido los PDFs con las
1204 especificaciones técnicas."
1205     # Mostrar preview de docs recuperados (expandible)
1206     with st.sidebar.expander("🔍 Ver documentos"):
1207         for i, doc in enumerate(docs, 1):
1208             preview = doc.page_content[:200]
1209             st.text(f"[{i}] {preview}...")
1210
1211     # Mostrar tamaño del contexto
1212     context = "\n\n".join([doc.page_content for doc in docs])
1213     st.sidebar.info(f"Tamaño: {len(context)} chars")
1214
1215     # Verificar palabras clave en contexto
1216     palabras = ["bala", "balas", "granada", "proyectil", "proyectiles", "mm", "nsn",
1217 "neq"]
1218     encontradas = [p for p in palabras if p in context.lower()]
1219     if encontradas:
1220         st.sidebar.success(f"✅ Palabras: {' '.join(encontradas)}")
1221     else:
1222         st.sidebar.warning("⚠️ NO hay palabras clave")
1223
1224     # Concatenar contexto de Los documentos recuperados
1225     context = "\n\n---DOCUMENTO---\n\n".join([doc.page_content for doc in docs])
1226
1227     # Prompt mejorado que FUERZA al modelo a usar solo el contexto
1228     prompt_text = f"""\nEres un asistente técnico especializado en municiones de la
1229 Armada Española.
1230
1231 INSTRUCCIONES ESTRUCTAS:
1232 1. Responde ÚNICAMENTE basándote en la información del CONTEXTO proporcionado a
1233 continuación
1234 2. NO uses tu conocimiento general o entrenamiento previo
1235 3. Si la información NO está en el CONTEXTO, di explícitamente "No encuentro esa
1236 información en los documentos cargados"

```

```

1237 4. Cita datos específicos del contexto (números, códigos NSN, especificaciones
1238 exactas)
1239 5. Sé preciso y técnico
1240
1241 ===CONTEXTO DE DOCUMENTOS TÉCNICOS===
1242 {context}
1243 ===FIN DEL CONTEXTO===
1244
1245 PREGUNTA DEL USUARIO: {pregunta}
1246
1247 RESPUESTA (basada SOLO en el contexto anterior, sin inventar ni usar conocimiento
1248 externo):"""
1249
1250     respuesta = invocar_llm_seguro(prompt_text)
1251
1252     # Verificación de seguridad: si la respuesta parece muy genérica, advertir
1253     if len(respuesta) < 50 and "contexto" not in respuesta.lower():
1254         respuesta += "\n\n⚠️ NOTA: Si esta respuesta parece incompleta, verifica
1255 que los documentos PDF estén correctamente cargados y vectorizados."
1256
1257     return respuesta
1258
1259 @tool
1260 def ejecutar_movimiento_municion(polvorin_destino: str, nombre_municion: str,
1261 cantidad: str, neq_unitario: str, grupo_riesgo: str):
1262     """
1263     Añade munición al inventario después de validar contra la normativa de
1264 seguridad.
1265     Valida: 1) NEQ máximo del polvorín, 2) Compatibilidad de grupos.
1266     Si falta el grupo de riesgo, lo busca automáticamente en los PDFs.
1267     """
1268     qty = _to_int(cantidad, default=None)
1269     neq = _to_float(neq_unitario, default=None)
1270
1271     if qty is None or neq is None:
1272         return f"Error: Valores numéricos no válidos. cantidad={cantidad}
1273 neq_unitario={neq_unitario}"
1274
1275
1276     # =====
1277     # VALIDACIÓN Y BÚSQUEDA AUTOMÁTICA: Grupo de riesgo
1278     # =====
1279     GRUPOS_VALIDOS = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'J', 'K', 'L', 'N',
1280 'S']
1281
1282     # Verificar si falta o es inválido
1283     grupo_faltante = (
1284         not grupo_riesgo or
1285

```

```

1286         str(grupo_riesgo).upper() in ['NULL', 'NONE', 'N/A', '', 'VACÍO', 'VACIO',
1287 'DESCONOCIDO'] or
1288         grupo_riesgo.upper() not in GRUPOS_VALIDOS
1289     )
1290
1291     if grupo_faltante:
1292         st.sidebar.info(f"🔍 Grupo de riesgo faltante. Buscando en documentación
1293 para '{nombre_municion}'...")
1294
1295         # Intentar buscar en PDFs
1296         datos_pdf = buscar_datos_municion_en_pdf(nombre_municion)
1297
1298         if datos_pdf and datos_pdf.get('grupo'):
1299             grupo_riesgo = normalizar_grupo_riesgo(datos_pdf['grupo'].upper())
1300             st.sidebar.success(f"✅ Grupo de riesgo encontrado: {grupo_riesgo}")
1301         else:
1302             # NO SE ENCONTRÓ: ahora sí denegar
1303             st.sidebar.error(f"❌ No se encontró el grupo de riesgo en la
1304 documentación")
1305             return f"❌ ERROR: Grupo de riesgo no disponible para
1306 '{nombre_municion}'"
1307
1308     Intenté buscar en la documentación pero no encontré información.
1309
1310     **Grupo de riesgo es OBLIGATORIO**
1311
1312     Opciones:
1313     1. Sube el PDF con especificaciones técnicas de municiones
1314     2. Proporciona el grupo manualmente:
1315         "añadir {qty} {nombre_municion} al {polvorin_destino} con NEQ {neq} y grupo X"
1316
1317     Grupos válidos: {'', '.join(GRUPOS_VALIDOS)}"
1318     else:
1319         # Normalizar a mayúscula
1320         grupo_riesgo = normalizar_grupo_riesgo(grupo_riesgo)
1321
1322         # =====
1323         # 1. CONSULTAR ESTADO ACTUAL DEL POLVORÍN (SQL)
1324         # =====
1325         conn = sqlite3.connect("arsenal_legacy.db")
1326         cursor = conn.cursor()
1327         res = cursor.execute(
1328             "SELECT SUM(neq_total_lote), grupo_riesgo FROM inventario WHERE
1329 polvorin_id = ? GROUP BY polvorin_id",
1330             (polvorin_destino,)
1331         ).fetchone()
1332         conn.close()
1333
1334         neq_actual = res[0] if res else 0.0
1335         grupo_existente = res[1] if res else None # None si está vacío

```

```

1336     neq_nuevo = qty * neq
1337     neq_total_final = neq_actual + neq_nuevo
1338
1339     polvorin_vacio = (grupo_existente is None)
1340
1341     # =====
1342     # 2. VALIDAR POLVORÍN (Existencia + NEQ máximo) - DESDE PDF
1343     # =====
1344     validacion_polvorin = validar_polvorin(polvorin_destino, neq_total_final)
1345
1346     neq_maximo = float(validacion_polvorin.get("neq_maximo", 0) or 0)
1347
1348     # Si la validación dice "valido", pero matemáticamente no cuadra, fuerza
1349     coherencia
1350     if neq_maximo > 0:
1351         if neq_total_final <= neq_maximo:
1352             # OK, sigue
1353             pass
1354         else:
1355             return (
1356                 f"⊖ OPERACIÓN DENEGADA\n\n"
1357                 f"NEQ resultante ({neq_total_final:.2f} kg) supera el límite del
1358     polvorín {polvorin_destino} "
1359                 f"({neq_maximo:.2f} kg).\n"
1360                 f"Diferencia: +{neq_total_final-neq_maximo:.2f} kg"
1361             )
1362
1363     if not validacion_polvorin["valido"]:
1364         return validacion_polvorin["mensaje"]
1365
1366     neq_maximo = validacion_polvorin.get("neq_maximo", 0)
1367
1368     # =====
1369     # 3. VALIDAR COMPATIBILIDAD DE GRUPOS - DESDE PDF
1370     # =====
1371     if polvorin_vacio:
1372         # Polvorín vacío: siempre compatible
1373         compatibilidad_ok = True
1374         mensaje_compatibilidad = f"Polvorín vacío → se asigna grupo
1375     {grupo_riesgo}"
1376     elif grupo_existente == grupo_riesgo:
1377         # Mismo grupo: compatible
1378         compatibilidad_ok = True
1379         mensaje_compatibilidad = f"Mismo grupo de riesgo ({grupo_riesgo})"
1380     else:
1381         # Grupos diferentes: consultar normativa
1382         vs = obtener_vectorstore()
1383
1384         if not vs:

```

```

1385         # Sin PDF de normativa: denegar por seguridad
1386         return f""""⊖ OPERACIÓN DENEGADA
1387
1388 RAZÓN: No hay documentación de normativa cargada para validar compatibilidad de
1389 grupos.
1390
1391 - Grupo existente en polvorín: {grupo_existente}
1392 - Grupo de munición a añadir: {grupo_riesgo}
1393
1394 Por seguridad, no se pueden mezclar grupos diferentes sin validar la normativa.
1395 Opciones:
1396 1. Cargar PDF de normativa de compatibilidad de grupos
1397 2. Usar otro polvorín vacío o con el mismo grupo {grupo_riesgo}""
1398
1399         retriever = vs.as_retriever(search_kwargs={"k": 5})
1400
1401         try:
1402             docs = retriever.invoke(f"compatibilidad grupos riesgo {grupo_riesgo}
1403 {grupo_existente} almacenamiento conjunto munición")
1404         except:
1405             docs = []
1406
1407         if not docs:
1408             # Sin información específica: denegar por seguridad
1409             return f""""⊖ OPERACIÓN DENEGADA
1410
1411 RAZÓN: No se encontró información sobre compatibilidad entre grupos
1412 {grupo_existente} y {grupo_riesgo}.
1413
1414 - Grupo existente en polvorín {polvorin_destino}: {grupo_existente}
1415 - Grupo de munición a añadir: {grupo_riesgo}
1416
1417 Por seguridad, no se pueden mezclar grupos sin confirmar compatibilidad en la
1418 normativa.
1419 Opciones:
1420 1. Cargar PDF de normativa de compatibilidad de grupos
1421 2. Consultar tabla de compatibilidad manual
1422 3. Usar otro polvorín vacío o con grupo {grupo_riesgo}""
1423
1424         # Preguntar al LLM sobre compatibilidad
1425         texto_normativa = "\n".join([d.page_content for d in docs])
1426
1427         prompt_compatibilidad = f""Eres un sistema de verificación de seguridad
1428 de municiones. Analiza SOLO la compatibilidad.
1429
1430 NORMATIVA DE COMPATIBILIDAD:
1431 {texto_normativa}
1432
1433 PREGUNTA:
1434

```

```

1435 ¿Los grupos de riesgo "{grupo_existente}" y "{grupo_riesgo}" son COMPATIBLES para
1436 almacenamiento conjunto?
1437
1438 INSTRUCCIONES:
1439 - Busca en la normativa si estos grupos pueden mezclarse
1440 - Si la normativa indica que son compatibles, responde: "COMPATIBLE"
1441 - Si la normativa indica que NO son compatibles, responde: "INCOMPATIBLE"
1442 - Si no encuentras información clara, responde: "INCOMPATIBLE" (por seguridad)
1443
1444 RESPONDE SOLO UNA PALABRA: COMPATIBLE o INCOMPATIBLE""
1445
1446     decision_compatibilidad = invocar_llm_seguro(prompt_compatibilidad,
1447     reintentos=2)
1448
1449     resp = re.sub(r"^[A-Z]", "", decision_compatibilidad.upper())
1450     compatibilidad_ok = (resp == "COMPATIBLE")
1451     if compatibilidad_ok:
1452         compatibilidad_ok = True
1453         mensaje_compatibilidad = f"Grupos {grupo_existente} y {grupo_riesgo}
1454     compatibles según normativa"
1455     else:
1456         compatibilidad_ok = False
1457         mensaje_compatibilidad = f"Grupos {grupo_existente} y {grupo_riesgo}
1458     NO son compatibles"
1459
1460     # =====
1461     # 4. DECISIÓN FINAL
1462     # =====
1463     if not compatibilidad_ok:
1464         return f"" ❌ OPERACIÓN DENEGADA
1465
1466     RAZÓN: Incompatibilidad de grupos de riesgo
1467     - Grupo existente en polvorín {polvorin_destino}: {grupo_existente}
1468     - Grupo de munición a añadir: {grupo_riesgo}
1469     - Evaluación: {mensaje_compatibilidad}
1470
1471     La normativa de seguridad NO permite mezclar estos grupos.
1472     Opciones:
1473     1. Usar un polvorín vacío
1474     2. Usar un polvorín que ya contenga grupo {grupo_riesgo}
1475     3. Extraer primero el contenido actual del polvorín {polvorin_destino}""
1476
1477     # =====
1478     # 5. EJECUTAR OPERACIÓN (Todo validado ✓)
1479     # =====
1480     conn = sqlite3.connect("arsenal_legacy.db")
1481     conn.execute(
1482         "INSERT INTO inventario VALUES (NULL, ?, ?, ?, ?, ?)",
1483         (polvorin_destino, nombre_municion, qty, neq_nuevo, grupo_riesgo)

```

```

1484     )
1485     conn.commit()
1486     conn.close()
1487
1488     # Mensaje de éxito detallado
1489     mensaje_estado = f"""\u2705 MOVIMIENTO AUTORIZADO
1490
1491     \ud83d\udcbb OPERACIÓN COMPLETADA:
1492     - Munición: {nombre_municion}
1493     - Cantidad: {qty} unidades
1494     - Polvorín destino: {polvorin_destino}
1495     - NEQ añadido: {neq_nuevo} kg
1496     - Grupo de riesgo: {grupo_riesgo}
1497
1498     \ud83d\udcbb ESTADO DEL POLVORÍN {polvorin_destino}:
1499     - NEQ anterior: {neq_actual:.2f} kg
1500     - NEQ actual: {neq_total_final:.2f} kg
1501     - NEQ máximo: {neq_maximo} kg
1502     - Espacio restante: {neq_maximo - neq_total_final:.2f} kg ({((neq_maximo -
1503     neq_total_final)/neq_maximo*100):.1f}%)
1504     """
1505
1506     if polvorin_vacio:
1507         mensaje_estado += f"\n\ud83d\udcbb Polvorín ahora asignado al grupo {grupo_riesgo}"
1508     else:
1509         mensaje_estado += f"\n\u2705 Grupo del polvorín: {grupo_riesgo}"
1510
1511     mensaje_estado += f"\n\u2705 Validación NEQ: OK\n\u2705 Validación compatibilidad: OK"
1512
1513     return mensaje_estado
1514
1515 @tool
1516 def extraer_municion(polvorin_origen: str, nombre_municion: str, cantidad: str):
1517     """
1518     Extrae/retira munición del inventario.
1519
1520     MEJORAS RESPECTO A VERSION ORIGINAL:
1521     1. Usa similitud_nombres_municion() para búsquedas más flexibles
1522     2. Threshold de similitud 0.7 para aceptar variaciones razonables
1523     3. Muestra candidatos cercanos si no encuentra coincidencia exacta
1524     4. Mantiene toda la lógica original de extracción múltiple
1525     """
1526     try:
1527         qty = int(float(cantidad))
1528     except:
1529         return "Error: Cantidad no válida."
1530
1531     if qty <= 0:
1532         return "Error: Cantidad debe ser mayor a 0."
1533

```

```

1534     conn = sqlite3.connect("arsenal_legacy.db")
1535     cursor = conn.cursor()
1536
1537     # Obtener TODOS los registros del polvorín
1538     todos = cursor.execute(
1539         "SELECT id, cantidad, neq_total_lote, nombre_municion FROM inventario
1540 WHERE polvorin_id = ?",
1541         (polvorin_origen,)
1542     ).fetchall()
1543
1544     if not todos:
1545         conn.close()
1546         return f"⊖ El polvorín {polvorin_origen} está vacío o no existe."
1547
1548     # Filtrar usando SIMILITUD en lugar de coincidencia exacta
1549     THRESHOLD_SIMILITUD = 0.7 # Ajustable: 0.7 = 70% similitud mínima
1550
1551     resultados = []
1552     similitudes = []
1553
1554     for r in todos:
1555         similitud = similitud_nombres_municion(nombre_municion, r[3])
1556         if similitud >= THRESHOLD_SIMILITUD:
1557             resultados.append(r)
1558             similitudes.append((similitud, r[3])) # Para debug
1559
1560     # Si no hay coincidencias, mostrar candidatos cercanos
1561     if not resultados:
1562         conn.close()
1563
1564     # Calcular similitudes con todas las municiones del polvorín
1565     candidatos = []
1566     for r in todos:
1567         sim = similitud_nombres_municion(nombre_municion, r[3])
1568         if sim > 0.3: # Umbral bajo para sugerencias
1569             candidatos.append((sim, r[3], r[1])) # similitud, nombre,
1570 cantidad
1571
1572     if candidatos:
1573         candidatos.sort(reverse=True) # Ordenar por similitud
1574         mensaje_candidatos = "📋 **Municiones disponibles similares:**\n"
1575         for sim, nom, cant in candidatos[:5]: # Top 5
1576             mensaje_candidatos += f"    • {nom}: {cant} unidades (similitud:
1577 {sim:.0%})\n"
1578
1579         return f""⊖ No se encontró munición similar a '{nombre_municion}' en
1580 el polvorín {polvorin_origen}.
1581
1582 {mensaje_candidatos}

```

```

1583
1584 💡 **Sugerencia:** Usa el nombre completo o intenta con una variación del
1585 nombre."""
1586
1587     return f"❌ No se encontró munición similar a '{nombre_municion}' en
1588 ningún polvorín."
1589
1590     # Debug: Mostrar en sidebar las coincidencias encontradas
1591     if similitudes:
1592         st.sidebar.success(f"🔍 Encontradas {len(resultados)} coincidencias:")
1593         for sim, nom in similitudes:
1594             st.sidebar.info(f"    • {nom} (similitud: {sim:.0%})")
1595
1596     stock_total = sum(r[1] for r in resultados)
1597     nombre_real = resultados[0][3] # Nombre completo de la munición
1598
1599     # Verificar stock suficiente considerando TODOS los registros
1600     if stock_total < qty:
1601         conn.close()
1602         filas_detalle = "\n".join([f"    • Registro #{r[0]}: {r[1]} unidades" for r
1603 in resultados])
1604         return f"" ❌ ERROR: Stock insuficiente.
1605
1606     Disponible total: {stock_total} unidades
1607     Solicitado: {qty} unidades
1608
1609     Detalle:
1610     {filas_detalle}"""
1611
1612     cantidad_restante = qty
1613     filas_modificadas = []
1614     filas_eliminadas = []
1615
1616     # Iterar sobre TODOS los registros encontrados
1617     for id_registro, cantidad_actual, neq_actual, _ in resultados:
1618         if cantidad_restante <= 0:
1619             break # Ya extrajimos todo lo necesario
1620
1621         # Calcular cuánto extraer de ESTE registro específico
1622         extraer_de_esta_fila = min(cantidad_restante, cantidad_actual)
1623
1624         # Calcular nuevos valores para ESTE registro
1625         neq_unitario = neq_actual / cantidad_actual
1626         nueva_cantidad = cantidad_actual - extraer_de_esta_fila
1627         nuevo_neq = nueva_cantidad * neq_unitario
1628
1629         # Actualizar o eliminar ESTE registro según corresponda
1630         if nueva_cantidad == 0:
1631             # Stock agotado en este registro → eliminarlo
1632             cursor.execute("DELETE FROM inventario WHERE id = ?", (id_registro,))

```

```

1633         filas_eliminadas.append(f" • Registro #{id_registro}:
1634 {extraer_de_esta_fila} unidades (agotado)")
1635     else:
1636         # Stock restante → actualizar cantidad y NEQ
1637         cursor.execute(
1638             "UPDATE inventario SET cantidad = ?, neq_total_lote = ? WHERE id =
1639 ?",
1640             (nueva_cantidad, nuevo_neq, id_registro)
1641         )
1642         filas_modificadas.append(f" • Registro #{id_registro}:
1643 {extraer_de_esta_fila} unidades extraídas, quedan {nueva_cantidad}")
1644
1645         # Restar de La cantidad pendiente
1646         cantidad_restante -= extraer_de_esta_fila
1647
1648     conn.commit()
1649     conn.close()
1650
1651     mensaje = f"✅ EXTRACCIÓN COMPLETADA\n\n{nombre_real}: {qty} unidades
1652 extraídas del polvorín {polvorin_origen}\n"
1653
1654     if filas_modificadas:
1655         mensaje += "\n📝 Registros actualizados:\n" + "\n".join(filas_modificadas)
1656
1657     if filas_eliminadas:
1658         mensaje += "\n\n⚠️ Registros agotados y eliminados:\n" +
1659 "\n".join(filas_eliminadas)
1660
1661     stock_restante = stock_total - qty
1662     mensaje += f"\n\n📊 Stock restante total: {stock_restante} unidades"
1663
1664     return mensaje
1665
1666 def _to_float(value, default=None):
1667     """Convierte '0.0008 kg' / '0,0008' / ' 0.0008 ' a float."""
1668     if value is None:
1669         return default
1670     s = str(value).strip()
1671     # coge el primer número que aparezca
1672     m = re.search(r"[-+]?[0-9]+(?:[.],\d+)?", s)
1673     if not m:
1674         return default
1675     return float(m.group(0).replace(",", "."))
1676
1677 def _to_int(value, default=None):
1678     """Convierte '500 unidades' / '500' a int."""
1679     f = _to_float(value, default=None)
1680     if f is None:
1681         return default

```

```
1682     return int(round(f))
1683 VALID_TIPOS = {"ENTRADA", "SALIDA", "TRANSFERENCIA"}
1684
1685 NULL_LIKE = {"NULL", "NONE", "N/A", "N.A", "", "FECHA_O_NULL", "NOMBRE_O_NULL"}
1686
1687 def normalizar_datos_movimiento(info: dict) -> dict:
1688     datos = info.get("datos") or {}
1689
1690     # Tipo: unificar fuente (datos o info["tipo"])
1691     tipo = (datos.get("tipo_operacion") or datos.get("tipo") or info.get("tipo")
1692 or "")
1693     tipo = str(tipo).strip().upper()
1694
1695     # Normalizar variantes típicas del LLM
1696     if tipo in {"ENTRADA/SALIDA", "ENTRADA - SALIDA", "ENTRADA Y SALIDA"}:
1697         tipo = "TRANSFERENCIA"
1698
1699     datos["tipo_operacion"] = tipo if tipo in VALID_TIPOS else None
1700
1701     # Normalizar campos texto y NULL-like
1702     for k in ["origen", "destino", "municion", "fecha", "autorizado_por",
1703 "grupo_riesgo", "neq_unitario"]:
1704         v = datos.get(k)
1705         if v is None:
1706             continue
1707         s = str(v).strip()
1708         if s.upper() in NULL_LIKE:
1709             datos[k] = None
1710         else:
1711             datos[k] = s
1712
1713     # Cantidad a int si se puede
1714     datos["cantidad"] = _to_int(datos.get("cantidad"), default=None)
1715
1716     info["datos"] = datos
1717     return info
1718
1719 def es_movimiento_valido(info: dict) -> bool:
1720     datos = info.get("datos") or {}
1721
1722     # Reglas mínimas para evitar falsos positivos
1723     if datos.get("tipo_operacion") not in VALID_TIPOS:
1724         return False
1725     if not datos.get("municion"):
1726         return False
1727     if datos.get("cantidad") is None:
1728         return False
1729     if not (datos.get("origen") or datos.get("destino")):
1730         return False
1731
```

```

1732     # Recomendado: exigir AL MENOS una señal administrativa (fecha o autorizado)
1733     if not (datos.get("fecha") or datos.get("autorizado_por")):
1734         return False
1735
1736     return True
1737
1738 # =====
1739 # PROCESADOR PRINCIPAL DE COMANDOS
1740 # =====
1741
1742 def ejecutar_con_herramientas(user_input):
1743     """
1744     Procesa la entrada del usuario y ejecuta la acción apropiada.
1745     Sistema de prioridades para evitar conflictos entre acciones.
1746     """
1747
1748     user_lower = user_input.lower()
1749
1750     # =====
1751     # PRIORIDAD 0.9: Cancelar confirmaciones pendientes (alta manual)
1752     # =====
1753     if re.search(r'\b(no|cancelar|cancela|anular|anula)\b', user_lower):
1754         if "pending_manual_add" in st.session_state:
1755             del st.session_state.pending_manual_add
1756             return "✅ Operación cancelada."
1757
1758     # =====
1759     # PRIORIDAD 0: Comandos de movimientos detectados en PDFs
1760     # =====
1761
1762     # 0A: Ejecutar movimiento desde albarán
1763     if
1764     re.search(r'\b(ejecutar|ejecuta|aplicar|aplica|procesar)\b.*\b(movimiento|albar[aa
1765 ]n)\b.*#?(\\d+)', user_lower):
1766         match = re.search(r'#?(\\d+)', user_input)
1767         if match and hasattr(st.session_state, 'movimientos_pendientes'):
1768             idx = int(match.group(1)) - 1 # Convertir a índice 0-based
1769
1770             if 0 <= idx < len(st.session_state.movimientos_pendientes):
1771                 mov = st.session_state.movimientos_pendientes[idx]
1772                 datos = mov['info']['datos']
1773
1774                 # Crear mensaje de confirmación
1775                 mensaje_conf = f"""📄 **CONFIRMAR MOVIMIENTO DEL ALBARÁN**
1776
1777 **Archivo:** {mov['archivo']}
1778 **Tipo:** {datos.get('tipo_operacion')}
1779 **Munición:** {datos.get('municion')}
1780 **Cantidad:** {datos.get('cantidad')} unidades

```

```

1781 """
1782
1783         if datos.get('tipo_operacion') == 'ENTRADA':
1784             destino = datos.get('destino', 'No especificado')
1785             mensaje_conf += f"***Destino:** {destino}\n"
1786         else:
1787             origen = datos.get('origen', 'No especificado')
1788             mensaje_conf += f"***Origen:** {origen}\n"
1789
1790             mensaje_conf += f"*****Fecha:** {datos.get('fecha', 'N/A')}}"
1791     **Autorizado por:** {datos.get('autorizado_por', 'N/A')}}"
1792
1793     ¿Confirmar y ejecutar este movimiento?
1794     Responde **"Sí"** para confirmar o **"No"** para cancelar."""
1795
1796         # Guardar índice del movimiento a ejecutar
1797         st.session_state.movimiento_a_ejecutar = idx
1798
1799         return mensaje_conf
1800     else:
1801         return f"❌ Movimiento #{idx+1} no encontrado. Hay
1802 {len(st.session_state.movimientos_pendientes)} movimiento(s) pendiente(s)."
1803         elif not hasattr(st.session_state, 'movimientos_pendientes'):
1804             return "❌ No hay movimientos pendientes. Primero procesa un PDF con
1805 albaranes."
1806         else:
1807             return "❌ Especifica el número del movimiento. Ejemplo: 'ejecutar
1808 movimiento #1'"
1809
1810     # 0B: Detalles de un movimiento
1811     if
1812 re.search(r'\b(detalle|detalles|info|informaci[oó]n)\b.*\b(movimiento|albar[aá]n)\b.*#?(\d+)', user_lower):
1813     match = re.search(r'#?(\d+)', user_input)
1814     if match and hasattr(st.session_state, 'movimientos_pendientes'):
1815         idx = int(match.group(1)) - 1
1816
1817         if 0 <= idx < len(st.session_state.movimientos_pendientes):
1818             mov = st.session_state.movimientos_pendientes[idx]
1819             datos = mov['info']['datos']
1820
1821             detalle = f"""📄 **DETALLE DEL ALBARÁN #{idx+1}**
1822
1823 **Archivo:** {mov['archivo']}
1824 **Tipo de operación:** {datos.get('tipo_operacion')}
1825
1826 **MUNICIÓN:**
1827 - Tipo: {datos.get('municion')}
1828 - Cantidad: {datos.get('cantidad')} unidades
1829 - NEQ unitario: {datos.get('neq_unitario', 'No especificado')}
1830

```

```

1831 - Grupo de riesgo: {datos.get('grupo_riesgo', 'No especificado')}
1832
1833 **UBICACIONES:**
1834 - Origen: {datos.get('origen', 'N/A')}
1835 - Destino: {datos.get('destino', 'N/A')}
1836
1837 **AUTORIZACIÓN:**
1838 - Fecha: {datos.get('fecha', 'N/A')}
1839 - Autorizado por: {datos.get('autorizado_por', 'N/A')}
1840
1841 Para ejecutar este movimiento: **"ejecutar movimiento #{idx+1}"**"
1842
1843         return detalle
1844     else:
1845         return f"✗ Movimiento #{idx+1} no encontrado."
1846     else:
1847         return "✗ No hay movimientos pendientes."
1848
1849     # 0C: Listar movimientos pendientes
1850     if
1851 re.search(r'\b(lista|listar|mostrar|ver)\b.*\b(movimientos|albaranes|pendientes)\b',
1852 user_lower):
1853         if hasattr(st.session_state, 'movimientos_pendientes') and
1854 st.session_state.movimientos_pendientes:
1855             mensaje = f"📋 **MOVIMIENTOS PENDIENTES**"
1856 ({len(st.session_state.movimientos_pendientes)})\n\n"
1857
1858             for idx, mov in enumerate(st.session_state.movimientos_pendientes, 1):
1859                 datos = mov['info']['datos']
1860                 mensaje += f"**#{idx}** - {mov['archivo']}\n"
1861                 mensaje += f"    • {datos.get('tipo_operacion')}:
1862 {datos.get('cantidad')} x {datos.get('municion')}\n\n"
1863
1864             mensaje += "Comandos:\n"
1865             mensaje += "• **ejecutar movimiento #N** - Ejecutar\n"
1866             mensaje += "• **detalles movimiento #N** - Ver detalles"
1867
1868             return mensaje
1869     else:
1870         return "✅ No hay movimientos pendientes."
1871
1872 # =====
1873 # PRIORIDAD 0.5: Inventario SQL SIEMPRE (aunque mencione "polvorín")
1874 # =====
1875 if re.search(r'\b(inventario|existencias|stock|cu[aá]nta munición|mostrar)\b',
1876 user_lower):
1877     filtro = ""
1878     # Detectar polvorín explícito
1879     for palabra in re.findall(r"[A-Za-zÁÉÍÓÚáéíóúÑñ0-9]+", user_input):
1880         if palabra.upper() in ["ALPHA", "BETA", "GAMMA", "DELTA"]:
```

```

1880         filtro = palabra.upper()
1881         break
1882     return consultar_inventario.invoke({"filtro": filtro})
1883
1884     # =====
1885     # PRIORIDAD 1: Confirmaciones (SÍ/NO) - AHORA INCLUYE MOVIMIENTOS
1886     # =====
1887     if re.search(r'\b(s[ií]|confirmar|confirmo|adelante|ok|vale|correcto)\b',
1888 user_lower):
1889
1890         # 1B-NUEVO: Confirmar alta manual pendiente (sin parsear texto del chat)
1891         if "pending_manual_add" in st.session_state:
1892             p = st.session_state.pending_manual_add
1893             del st.session_state.pending_manual_add # evitar doble ejecución
1894
1895             # Validación NEQ máximo (si aplica)
1896             conn = sqlite3.connect("arsenal_legacy.db")
1897             cursor = conn.cursor()
1898             res = cursor.execute(
1899                 "SELECT SUM(neq_total_lote) FROM inventario WHERE polvorin_id =
1900 ?"),
1901                 (p["polvorin"],)
1902             ).fetchone()
1903             conn.close()
1904
1905             neq_actual = res[0] if res and res[0] else 0.0
1906             qty = _to_int(p["cantidad"], default=0)
1907             neq_unit = _to_float(p["neq"], default=0.0)
1908             neq_total_resultante = neq_actual + (qty * neq_unit)
1909
1910             validacion = validar_polvorin(p["polvorin"], neq_total_resultante)
1911             if not validacion.get("valido", False):
1912                 return validacion.get("mensaje", "⊖ Operación denegada por
1913 validación de polvorín.")
1914
1915             return ejecutar_movimiento_municion.invoke({
1916                 "polvorin_destino": p["polvorin"],
1917                 "nombre_municion": p["municion"],
1918                 "cantidad": str(qty),
1919                 "neq_unitario": str(neq_unit),
1920                 "grupo_riesgo": p["grupo"]
1921             })
1922         # 1A: Confirmar ejecución de movimiento desde albarán
1923         if hasattr(st.session_state, 'movimiento_a_ejecutar'):
1924             idx = st.session_state.movimiento_a_ejecutar
1925             mov = st.session_state.movimientos_pendientes[idx]
1926             datos = mov['info']['datos']
1927
1928             with st.spinner("⚙ Ejecutando movimiento del albarán..."):
1929                 try:

```

```

1930         if datos['tipo_operacion'] == 'ENTRADA':
1931             # Entrada: añadir munición
1932             destino = datos.get('destino', '')
1933
1934             # Normalizar nombre del polvorín
1935             destino_norm = None
1936             destino_upper = destino.upper()
1937             for pol in ['ALPHA', 'BETA', 'GAMMA', 'DELTA']:
1938                 if pol in destino_upper:
1939                     destino_norm = pol
1940                     break
1941
1942             if not destino_norm:
1943                 return f"⊖ ERROR: No se pudo identificar el polvorín
1944 de destino en '{destino}'.\n\nPolvorines válidos: ALPHA, BETA, GAMMA, DELTA"
1945
1946             # Buscar datos técnicos si faltan
1947             neq = datos.get('neq_unitario')
1948             grupo = datos.get('grupo_riesgo')
1949
1950             # Limpiar valores nulos/vacíos
1951             if not neq or str(neq).upper() in ['NULL', 'NONE', 'N/A',
1952 '', 'DESCONOCIDO']:
1953                 neq = None
1954
1955             if not grupo or str(grupo).upper() in ['NULL', 'NONE',
1956 'N/A', '', 'VACÍO', 'VACIO', 'DESCONOCIDO']:
1957                 grupo = None
1958
1959             # Si falta NEQ o grupo, buscar en PDFs
1960             if not neq or not grupo:
1961                 st.sidebar.info("🔍 Buscando datos técnicos en
1962 documentación...")
1963                 datos_pdf =
1964 buscar_datos_municion_en_pdf(datos['municion'])
1965
1966                 if datos_pdf:
1967                     if not neq:
1968                         neq = datos_pdf['neq']
1969                     if not grupo:
1970                         grupo = datos_pdf['grupo']
1971                     st.sidebar.success(f"✅ Datos encontrados:
1972 NEQ={neq}, Grupo={grupo}")
1973                 else:
1974                     st.sidebar.error("❌ No se encontró en
1975 documentación")
1976
1977             # VALIDACIÓN FINAL: Solo denegar si realmente faltan
1978 después de buscar

```

```

1979         errores = []
1980         if not neq:
1981             errores.append("• NEQ unitario")
1982         if not grupo:
1983             errores.append("• Grupo de riesgo")
1984
1985         if errores:
1986             return f"" ❌ ERROR: No se pudieron obtener los datos
1987 técnicos obligatorios para '{datos['municion']}'
1988
1989 Datos faltantes:
1990 {chr(10)}.join(errores)}
1991
1992 Busqué en la documentación pero no encontré esta información.
1993
1994 **Solución:**
1995 1. Sube el PDF de especificaciones técnicas de municiones que incluya
1996 '{datos['municion']}'
1997 2. O cancela este albarán y crea uno nuevo con los datos completos""
1998
1999         cantidad_limpia = str(_to_int(datos.get('cantidad')))
2000         neq_limpio = str(_to_float(neq))
2001
2002         resultado = ejecutar_movimiento_municion.invoke({
2003             "polvorin_destino": destino_norm,
2004             "nombre_municion": datos['municion'],
2005             "cantidad": cantidad_limpia,
2006             "neq_unitario": neq_limpio,
2007             "grupo_riesgo": str(grupo)
2008         })
2009
2010
2011         # Limpiar variable de sesión
2012         del st.session_state.movimiento_a_ejecutar
2013
2014         return f" 📄 **ALBARÁN PROCESADO:**
2015 {mov['archivo']}\n\n{resultado}"
2016
2017         elif datos['tipo_operacion'] == 'SALIDA':
2018             # Salida: extraer munición
2019             origen = datos.get('origen', '')
2020
2021             # Normalizar nombre del polvorín
2022             origen_norm = None
2023             origen_upper = origen.upper()
2024             for pol in ['ALPHA', 'BETA', 'GAMMA', 'DELTA']:
2025                 if pol in origen_upper:
2026                     origen_norm = pol
2027                     break
2028

```

```

2029         if not origen_norm:
2030             return f"⊖ ERROR: No se pudo identificar el polvorín
2031 de origen en '{origen}'.\n\nPolvorines válidos: ALPHA, BETA, GAMMA, DELTA"
2032
2033         # Ejecutar movimiento de salida
2034         resultado = extraer_municion.invoke({
2035             "polvorin_origen": origen_norm,
2036             "nombre_municion": datos['municion'],
2037             "cantidad": str(datos['cantidad'])
2038         })
2039
2040         # Limpiar variable de sesión
2041         del st.session_state.movimiento_a_ejecutar
2042
2043         return f"📄 **ALBARÁN PROCESADO:**
2044 {mov['archivo']}\n\n{resultado}"
2045
2046     else:
2047         del st.session_state.movimiento_a_ejecutar
2048         return f"✗ Tipo de operación no reconocida:
2049 {datos['tipo_operacion']}"
2050
2051     except Exception as e:
2052         if hasattr(st.session_state, 'movimiento_a_ejecutar'):
2053             del st.session_state.movimiento_a_ejecutar
2054             return f"✗ ERROR al ejecutar movimiento: {str(e)}"
2055
2056     # 1B: Confirmar adición manual de munición (flujo existente)
2057     if st.session_state.messages and len(st.session_state.messages) >= 2:
2058         ultimo_msg = st.session_state.messages[-2]["content"]
2059
2060         if "DATOS ENCONTRADOS" in ultimo_msg or "¿Proceder" in ultimo_msg:
2061             # ... (código existente de confirmación manual) ...
2062             try:
2063                 neq_match = re.search(r'NEQ unitario: ([\d.]+)', ultimo_msg)
2064                 grupo_match = re.search(r'Grupo de riesgo: ([A-Z])',
2065 ultimo_msg)
2066                 cantidad_match = re.search(r'Añadir (\d+) unidades',
2067 ultimo_msg)
2068                 polvorin_match = re.search(r'polvorín ([A-Z]+)', ultimo_msg)
2069                 nombre_match = re.search(r'identificada: \*\*(.+?)\*\*',
2070 ultimo_msg)
2071
2072                 if all([neq_match, grupo_match, cantidad_match,
2073 polvorin_match, nombre_match]):
2074                     qty = int(cantidad_match.group(1))
2075                     neq = float(neq_match.group(1))
2076                     polvorin = polvorin_match.group(1)
2077

```

```

2078         conn = sqlite3.connect("arsenal_legacy.db")
2079         cursor = conn.cursor()
2080         res = cursor.execute(
2081             "SELECT SUM(neq_total_lote) FROM inventario WHERE
2082 polvorin_id = ?",
2083             (polvorin,)
2084         ).fetchone()
2085         conn.close()
2086
2087         neq_actual = res[0] if res and res[0] else 0.0
2088         neq_total_resultante = neq_actual + (qty * neq)
2089
2090         validacion = validar_polvorin(polvorin,
2091 neq_total_resultante)
2092
2093         if not validacion["valido"]:
2094             return validacion["mensaje"]
2095
2096         return ejecutar_movimiento_municion.invoke({
2097             "polvorin_destino": polvorin,
2098             "nombre_municion": nombre_match.group(1),
2099             "cantidad": cantidad_match.group(1),
2100             "neq_unitario": neq_match.group(1),
2101             "grupo_riesgo": grupo_match.group(1)
2102         })
2103     except:
2104         pass
2105
2106     # =====
2107     # PRIORIDAD 2: Consultas sobre polvorines
2108     # =====
2109     MOV_VERBS_ADD =
2110 r"\b(mete|meto|meter|añadir|añado|cargar|cargo|transportar|transporto|agregar|agre
2111 go|introducir|introduzco|almacenar|almaceno)\b"
2112     MOV_VERBS_OUT =
2113 r"\b(saca|saco|sacar|extraer|extraigo|retirar|retiro|quitar|quito|remove|remuevo)
2114 \b"
2115     if re.search(MOV_VERBS_ADD, user_lower) or re.search(MOV_VERBS_OUT,
2116 user_lower):
2117         pass
2118     else:
2119         if
2120 re.search(r'\b(polvor[íi]n|polvorines|almac[eé]n|almacenes|dep[oó]sit)',
2121 user_lower):
2122
2123             # 2A: Capacidad/espacio disponible de un polvorín específico
2124             if
2125 re.search(r'\b(capacidad|espacio|disponible|queda|libre|ocupaci[oó]n|cu[aá]nto.*qu
2126 eda)\b', user_lower):
2127                 # Detectar polvorín

```

```

2128         polvorin_mencionado = None
2129         for palabra in user_input.split():
2130             if palabra.upper() in ["ALPHA", "BETA", "GAMMA", "DELTA"]:
2131                 polvorin_mencionado = palabra.upper()
2132                 break
2133
2134         if polvorin_mencionado:
2135             # Obtener NEQ máximo del PDF
2136             vs = obtener_vectorstore()
2137             if vs:
2138                 info_pdf = consultar_documentacion_tecnica.invoke({
2139                     "pregunta": f"¿Cuál es el NEQ máximo del polvorín
2140 {polvorin_mencionado}? Responde solo el número."
2141                 })
2142
2143                 try:
2144                     neq_maximo = float(re.search(r'(\d+(?:\.\d+)?)',
2145 info_pdf).group(1))
2146                 except:
2147                     neq_maximo = 0
2148             else:
2149                 neq_maximo = 0
2150
2151             # Obtener NEQ actual del SQL
2152             conn = sqlite3.connect("arsenal_legacy.db")
2153             cursor = conn.cursor()
2154             res = cursor.execute(
2155                 "SELECT SUM(neq_total_lote) FROM inventario WHERE
2156 polvorin_id = ?",
2157                 (polvorin_mencionado,)
2158             ).fetchone()
2159
2160             df = pd.read_sql_query(
2161                 "SELECT nombre_municion, cantidad, neq_total_lote FROM
2162 inventario WHERE polvorin_id = ?",
2163                 conn,
2164                 params=(polvorin_mencionado,)
2165             )
2166             conn.close()
2167
2168             neq_actual = res[0] if res and res[0] else 0.0
2169             neq_disponible = neq_maximo - neq_actual if neq_maximo > 0
2170         else 0
2171
2172         porcentaje = (neq_actual / neq_maximo * 100) if neq_maximo > 0
2173         else 0
2174
2175         resultado = f"""📊 CAPACIDAD DEL POLVORÍN
2176 {polvorin_mencionado}

```

```

2177     ◇ NEQ Máximo: {neq_maximo} kg
2178     ◇ NEQ Actual: {neq_actual:.2f} kg
2179     ◇ NEQ Disponible: **{neq_disponible:.2f} kg**
2180     ◇ Ocupación: {porcentaje:.1f}%
2181
2182     """
2183
2184         if not df.empty:
2185             resultado += "📦 Contenido actual:\n"
2186             for _, row in df.iterrows():
2187                 resultado += f"    • {row['nombre_municion']}:
2188 {row['cantidad']} unidades ({row['neq_total_lote']:.2f} kg)\n"
2189             else:
2190                 resultado += "📦 Polvorín vacío\n"
2191
2192             return resultado
2193         else:
2194             return "❌ ¿De qué polvorín quieres consultar la capacidad?
2195 (ALPHA, BETA, GAMMA, DELTA)"
2196
2197         # 2B: Lista completa de polvorines
2198         elif
2199 re.search(r'\b(lista|cu[aá]ntos|cu[aá]les|disponibles|existentes|hay|todos)\b',
2200 user_lower):
2201             return consultar_documentacion_tecnica.invoke({
2202                 "pregunta": "Lista completa y detallada de todos los
2203 polvorines del arsenal. Para cada polvorín indica: código/nombre, NEQ máximo
2204 permitido, capacidad en m³, grupos de riesgo compatibles, y ubicación. Responde en
2205 formato de lista estructurada y clara."
2206             })
2207
2208         # 2C: Características específicas
2209         else:
2210             return consultar_documentacion_tecnica.invoke({"pregunta":
2211 user_input})
2212
2213         # =====
2214         # PRIORIDAD 3: Consultas de inventario SQL (stock, existencias)
2215         # =====
2216         # Nota: Ahora solo activa si NO menciona polvorines/almacenes
2217         if re.search(r'\b(inventario|existencias|stock|cu[aá]nta munición|mostrar)\b',
2218 user_lower):
2219             filtro = ""
2220             for palabra in user_input.split():
2221                 if palabra.upper() in ["ALPHA", "BETA", "GAMMA", "DELTA"]:
2222                     filtro = palabra.upper()
2223                     break
2224
2225             return consultar_inventario.invoke({"filtro": filtro})
2226

```

```

2227 # =====
2228 # PRIORIDAD 4: Extracciones de munición
2229 # =====
2230 if
2231 re.search(r'\b(saca|saco|sacar|extraer|extraigo|retirar|retiro|quitar|quito|remove
2232 r|remuevo)\b', user_lower):
2233
2234     st.sidebar.info("🔍 Detectado: Extracción de munición")
2235
2236     prompt = f"""Analiza esta solicitud de EXTRACCIÓN de munición:
2237 "{user_input}"
2238
2239 Extrae EXACTAMENTE estos parámetros y responde SOLO en formato JSON (sin texto
2240 adicional, sin markdown):
2241
2242 {{{"polvorin": "NOMBRE_POLVORIN_O_AUTO", "municion": "NOMBRE_COMPLETO", "cantidad":
2243 "NUMERO"}}}
2244
2245 Ejemplos válidos:
2246 - Entrada: "saca 3 proyectiles de 155mm" → {{{"polvorin": "AUTO", "municion":
2247 "Proyectil 155mm", "cantidad": "3"}}}
2248 - Entrada: "extrae 5 balas del ALPHA" → {{{"polvorin": "ALPHA", "municion":
2249 "balas", "cantidad": "5"}}}
2250
2251 Si NO se menciona el polvorín, pon "AUTO" en polvorin.
2252 NO agregues texto antes o después del JSON."""
2253
2254     try:
2255         respuesta_texto = invocar_llm_seguro(prompt, reintentos=2)
2256         params = extraer_json_robusto(respuesta_texto)
2257
2258         if not params:
2259             return "❌ Error: No pude entender los datos de la solicitud.
2260 Intenta reformular."
2261
2262         st.sidebar.write("Datos detectados:", params) # Debug opcional
2263
2264         if not params.get("municion") or params.get("municion") ==
2265 "DESCONOCIDO":
2266             return "❌ ¿Qué tipo de munición quieres extraer?"
2267
2268         if not params.get("cantidad") or params.get("cantidad") ==
2269 "DESCONOCIDO":
2270             return "❌ ¿Cuántas unidades quieres extraer?"
2271
2272         if params.get("polvorin") == "AUTO" or not params.get("polvorin"):
2273             conn = sqlite3.connect("arsenal_legacy.db")
2274             cursor = conn.cursor()
2275             rows = cursor.execute(

```

```

2276         "SELECT polvorin_id, cantidad, nombre_municion FROM
2277 inventario"
2278     ).fetchall()
2279     conn.close()
2280
2281     busc = normalizar_nombre_municion(params["municion"])
2282     resultados = []
2283     for pol_id, cant, nom in rows:
2284         if busc in normalizar_nombre_municion(nom):
2285             resultados.append((pol_id, cant, nom))
2286
2287     if not resultados:
2288         return f"⊖ No se encontró munición similar a
2289 '{params['municion']}' en ningún polvorín."
2290     elif len({r[0] for r in resultados}) == 1:
2291         params["polvorin"] = resultados[0][0]
2292         st.sidebar.success(f"☑ Polvorín detectado automáticamente:
2293 {params['polvorin']}")
2294     else:
2295         opciones = "\n".join([f" • {r[0]}: {r[1]} unidades ({r[2]}]"
2296 for r in resultados])
2297         return f"⚠ '{params['municion']}' está en múltiples
2298 polvorines:\n\n{opciones}\n\nEspecifica el polvorín de origen."
2299
2300
2301         resultado = extraer_municion.invoke({
2302             "polvorin_origen": params["polvorin"],
2303             "nombre_municion": params["municion"],
2304             "cantidad": params["cantidad"]
2305         })
2306
2307         return resultado
2308
2309     except json.JSONDecodeError as e:
2310         st.sidebar.error(f"✗ Error JSON: {e}")
2311         st.sidebar.code(params_json)
2312         return f"✗ Error al procesar la solicitud.\nEspecifica: tipo de
2313 munición, cantidad y (opcional) polvorín.\nEjemplo: 'saca 3 proyectiles 155mm del
2314 ALPHA'"
2315     except Exception as e:
2316         st.sidebar.error(f"✗ Error general: {e}")
2317         return f"✗ Error al procesar extracción: {str(e)}"
2318
2319     # =====
2320     # PRIORIDAD 5: Adiciones/Movimientos de munición
2321     # =====
2322     if
2323 re.search(r'\b(mete|meto|meter|a[ñn]adir|a[ñn]ado|cargar|cargo|transportar|transpo
2324 rto|agregar|agrego|introducir|introduzco|almacenar|almaceno)\b', user_lower):
2325

```

```

2326     st.sidebar.info("🔍 Detectado: Adición de munición")
2327
2328     prompt = f"""Analiza esta solicitud de AÑADIR munición: "{user_input}"
2329
2330 Extrae estos parámetros y responde SOLO en JSON (sin texto extra, sin markdown):
2331
2332 {{"polvorin": "ID_O_FALTA", "municion": "NOMBRE_COMPLETO", "cantidad": "NUM",
2333 "neq": "NUMERO_O_AUTO", "grupo": "LETRA_O_AUTO"}}
2334
2335 IMPORTANTE: Si NO se menciona explícitamente el polvorín (ALPHA, BETA, GAMMA,
2336 DELTA), pon "FALTA".
2337
2338 Ejemplos:
2339 - "mete 100 balas de 9mm en ALPHA" → {{"polvorin": "ALPHA", "municion": "Bala
2340 9mm", "cantidad": "100", "neq": "AUTO", "grupo": "AUTO"}}
2341 - "añadir 100 balas" → {{"polvorin": "FALTA", "municion": "balas", "cantidad":
2342 "100", "neq": "AUTO", "grupo": "AUTO"}}
2343
2344 Si NO se menciona NEQ o grupo, pon "AUTO"."""
2345
2346     try:
2347         respuesta_texto = invocar_llm_seguro(prompt, reintentos=2)
2348         params = extraer_json_robusto(respuesta_texto)
2349
2350         if not params:
2351             return "❌ Error: No pude entender los datos. Asegúrate de
2352 indicar munición y cantidad."
2353
2354         st.sidebar.write("Datos detectados:", params) # Debug opcional
2355
2356         if not params.get("polvorin") or params.get("polvorin") in ["FALTA",
2357 "DESCONOCIDO", "ID_NO_PROVIDED", "ID"]:
2358             return "❌ FALTA INFORMACIÓN: Polvorín de destino
2359
2360 Debes especificar a qué polvorín quieres añadir la munición.
2361
2362 Ejemplo: "mete 100 balas de 9mm en el polvorín ALPHA"
2363
2364 Para ver la lista de polvorines disponibles, pregunta: "lista de polvorines" ""
2365
2366         if not params.get("municion") or params.get("municion") ==
2367 "DESCONOCIDO":
2368             return "❌ ¿Qué tipo de munición quieres añadir?"
2369
2370         if not params.get("cantidad") or params.get("cantidad") ==
2371 "DESCONOCIDO":
2372             return "❌ ¿Cuántas unidades?"
2373
2374         if params.get("neq") == "AUTO" or params.get("grupo") == "AUTO":

```

```

2375         st.sidebar.info("🔍 Buscando datos técnicos en PDFs...")
2376         datos_pdf = buscar_datos_municion_en_pdf(params["municion"])
2377
2378         if datos_pdf:
2379             if params.get("neq") == "AUTO":
2380                 params["neq"] = datos_pdf["neq"]
2381             if params.get("grupo") == "AUTO":
2382                 params["grupo"] = datos_pdf["grupo"]
2383
2384             st.sidebar.success(f"✅ Encontrado: NEQ={params['neq']},
2385 Grupo={params['grupo']}")
2386             st.session_state.pending_manual_add = {
2387                 "polvorin": params["polvorin"],
2388                 "municion": params["municion"],
2389                 "cantidad": str(params["cantidad"]),
2390                 "neq": str(params["neq"]),
2391                 "grupo": str(params["grupo"])}
2392
2393             return f"""📄 DATOS ENCONTRADOS EN DOCUMENTACIÓN
2394
2395 Munición identificada: **{params['municion']}**
2396 - NEQ unitario: {params['neq']} kg
2397 - Grupo de riesgo: {params['grupo']}
2398
2399 ¿Proceder?
2400 ➡️ Añadir {params['cantidad']} unidades al polvorín {params['polvorin']}
2401
2402 Responde "Sí" para confirmar."""
2403         else:
2404             st.sidebar.warning("⚠️ No se encontró en PDFs")
2405             return f"""⚠️ No encontré datos técnicos de
2406 '{params['municion']}' en la documentación.
2407
2408 Opciones:
2409 1. Proporciona los datos manualmente:
2410     "añadir {params['cantidad']} {params['municion']} al {params['polvorin']} con
2411     NEQ X.XX y grupo Y"
2412
2413 2. O sube el PDF con las especificaciones técnicas de municiones."""
2414
2415         try:
2416             qty = int(float(params["cantidad"]))
2417             neq = float(params["neq"])
2418         except:
2419             return "❌ Error: Valores numéricos no válidos."
2420
2421         conn = sqlite3.connect("arsenal_legacy.db")
2422         cursor = conn.cursor()
2423         res = cursor.execute(
2424

```

```

2425         "SELECT SUM(neq_total_lote) FROM inventario WHERE polvorin_id =
2426         ?",
2427         (params["polvorin"],)
2428     ).fetchone()
2429     conn.close()
2430
2431     neq_actual = res[0] if res and res[0] else 0.0
2432     neq_nuevo = qty * neq
2433     neq_total_resultante = neq_actual + neq_nuevo
2434
2435     st.sidebar.info(f"🔍 Validando polvorín {params['polvorin']}...")
2436     validacion = validar_polvorin(params["polvorin"],
2437     neq_total_resultante)
2438
2439     if not validacion["valido"]:
2440         st.sidebar.error("❌ Validación falló")
2441         return validacion["mensaje"]
2442
2443     st.sidebar.success(validacion["mensaje"])
2444
2445     resultado = ejecutar_movimiento_municion.invoke({
2446         "polvorin_destino": params["polvorin"],
2447         "nombre_municion": params["municion"],
2448         "cantidad": params["cantidad"],
2449         "neq_unitario": params["neq"],
2450         "grupo_riesgo": params["grupo"]
2451     })
2452
2453     except json.JSONDecodeError as e:
2454         st.sidebar.error(f"❌ Error JSON: {e}")
2455         return f"❌ Error al procesar. Especifica: munición, cantidad,
2456         polvorín.\nEjemplo: 'mete 100 balas 9mm en ALPHA'"
2457     except Exception as e:
2458         st.sidebar.error(f"❌ Error: {e}")
2459         return f"❌ Error: {str(e)}"
2460
2461     # =====
2462     # PRIORIDAD 6: Consultas técnicas generales (NEQ, grupos de riesgo, normativa)
2463     # =====
2464     # Patrones que indican consulta de documentación técnica:
2465     # - Palabras técnicas: neq, grupo, riesgo, especificación, norma, nsn, etc.
2466     # - Tipos de munición: proyectil, munición, cartucho, granada
2467     # - Consultas de datos: información, componentes, características, descripción
2468     patron_rag =
2469     r'\b(neq|grupo|riesgo|especificaci[ó]n|dato|t[eé]cnico|norma|compatible|' \
2470         r'nsn|proyectil|municion|munición|cartucho|granada|' \
2471         r'informaci[ó]n|componente|caracter[ií]stica|descripci[ó]n|' \

```

```

2470         r'melara|parabellum|otan|nato|calibre|fragmentaci[oó]n|' \
2471         r'explosivo|espoleta|propulsor|vida\s+[uú]til|almacenamiento)\b'
2472
2473     if re.search(patron_rag, user_lower):
2474         return consultar_documentacion_tecnica.invoke({"pregunta": user_input})
2475
2476     # =====
2477     # PRIORIDAD 7: Conversación general
2478     # =====
2479     return invocar_llm_seguro(f""Eres el asistente SIGLM (Sistema Integrado de
2480 Gestión de Logística de Munición) de la Armada.
2481
2482 Funciones disponibles:
2483 - Consultar inventario: "dime el inventario" / "stock del ALPHA"
2484 - Añadir munición: "mete X unidades de Y en polvorín Z"
2485 - Extraer munición: "saca X unidades de Y"
2486 - Consultas técnicas: "cuál es el NEQ de..." / "lista de polvorines" /
2487 "características del ALPHA"
2488
2489 Responde brevemente y profesionalmente a: {user_input}""")
2490
2491 # =====
2492 # INTERFAZ DE USUARIO (STREAMLIT)
2493 # =====
2494 def tesseract_info():
2495     if pytesseract is None:
2496         return "pytesseract no está importado"
2497     try:
2498         v = pytesseract.get_tesseract_version()
2499         langs = pytesseract.get_languages(config="")
2500         return f"Tesseract OK: {v}\nIdiomas: {'', '.join(langs)}"
2501     except Exception as e:
2502         return f"ERROR Tesseract: {e}"
2503
2504 st.title("📄 SIGLM - Sistema de Gestión de Logística de Munición")
2505
2506 # Sidebar: Gestión de documentos e inventario
2507 with st.sidebar:
2508     st.subheader("📄 Gestión de Documentos")
2509     st.caption("Sube PDFs con: normativa, especificaciones, polvorines,
2510 albaranes")
2511
2512     uploaded = st.file_uploader("Subir PDFs", accept_multiple_files=True,
2513 type="pdf", key="pdf_uploader")
2514
2515     if uploaded and st.button("📄 Procesar PDFs"):
2516         with st.spinner("Analizando documentos..."):
2517             success, movimientos = procesar_nuevos_pdfs(uploaded)
2518
2519             if success:

```

```

2520         st.success(f"✅ {len(uploaded)} documento(s) procesado(s)")
2521     else:
2522         st.error("❌ Error al procesar documentos")
2523
2524     if "messages" not in st.session_state:
2525         st.session_state.messages = []
2526
2527     if success:
2528         nombres = ", ".join([f.name for f in uploaded])
2529         st.session_state.messages.append({
2530             "role": "assistant",
2531             "content": f"✅ Documentación cargada y vectorizada:
2532 {len(uploaded)} PDF(s).\n\n📁 Archivos: {nombres}"
2533         })
2534     else:
2535         st.session_state.messages.append({
2536             "role": "assistant",
2537             "content": "❌ No se pudo procesar la documentación. Revisa
2538 que los PDFs sean válidos y vuelve a intentarlo."
2539         })
2540
2541     # Si se detectaron movimientos, añadir al chat
2542     if movimientos:
2543         st.success(f"📄 {len(movimientos)} albarán(es) detectado(s)")
2544
2545         # Guardar en session_state
2546         st.session_state.movimientos_pendientes = movimientos
2547
2548         # Crear mensaje para el chat
2549         mensaje_movimientos = "📄 **ALBARANES DE MOVIMIENTO
2550 DETECTADOS**\n\n"
2551
2552         for idx, mov in enumerate(movimientos, 1):
2553             datos = mov['info']['datos']
2554             mensaje_movimientos += f"***#{idx} - {mov['archivo']}**\n"
2555             mensaje_movimientos += f"• Tipo:
2556 {datos.get('tipo_operacion')}\n"
2557             mensaje_movimientos += f"• Munición:
2558 {datos.get('municion')}\n"
2559             mensaje_movimientos += f"• Cantidad: {datos.get('cantidad')}"
2560             unidades\n"
2561
2562             if datos.get('tipo_operacion') == 'ENTRADA':
2563                 mensaje_movimientos += f"• Destino: {datos.get('destino',
2564 'No especificado')}\n"
2565             else:
2566                 mensaje_movimientos += f"• Origen: {datos.get('origen',
2567 'No especificado')}\n"
2568

```

```

2569         mensaje_movimientos += f"• Fecha: {datos.get('fecha',
2570 'N/A')}\n"
2571         mensaje_movimientos += f"• Autorizado por:
2572 {datos.get('autorizado_por', 'N/A')}\n\n"
2573
2574         mensaje_movimientos += "Para ejecutar un movimiento, escribe:
2575 **ejecutar movimiento #N**\n"
2576         mensaje_movimientos += "Para ver detalles: **detalles movimiento
2577 #N**"
2578
2579         # Añadir mensaje del sistema al chat
2580         st.session_state.messages.append({
2581             "role": "assistant",
2582             "content": mensaje_movimientos
2583         })
2584
2585         st.rerun()
2586
2587     if st.button("🔄 Reset Sistema"):
2588         import gc
2589
2590         # CRÍTICO: Limpiar la instancia cacheada de vectorstore
2591         if 'vectorstore' in st.session_state:
2592             try:
2593                 # Intentar cerrar la conexión si tiene método close
2594                 if hasattr(st.session_state.vectorstore, '_client'):
2595                     st.session_state.vectorstore._client = None
2596             except:
2597                 pass
2598             st.session_state.vectorstore = None
2599
2600         # Limpiar otras referencias por si acaso
2601         if 'vector_store' in st.session_state:
2602             st.session_state.vector_store = None
2603         if 'retriever' in st.session_state:
2604             st.session_state.retriever = None
2605
2606         # Forzar garbage collection para liberar recursos
2607         gc.collect()
2608         time.sleep(1) # Dar más tiempo a Windows para cerrar handles
2609
2610         # Intentar eliminar con reintentos
2611         if os.path.exists(PERSIST_DIRECTORY):
2612             max_intentos = 5 # Más intentos
2613             for intento in range(max_intentos):
2614                 try:
2615                     shutil.rmtree(PERSIST_DIRECTORY)
2616                     st.success("🗑️ Base de datos vectorial eliminada")
2617                     break
2618                 except (PermissionError, OSError) as e:

```

```

2619         if intento < max_intentos - 1:
2620             time.sleep(1.5)
2621             gc.collect()
2622         else:
2623             st.error(f"⚠ No se pudo eliminar completamente:
2624 {str(e)}")
2625             st.warning("💡 Solución: Cierra y reinicia la aplicación,
2626 o elimina manualmente la carpeta 'db_vectores'")
2627         else:
2628             st.info("❗ No hay base de datos vectorial que eliminar")
2629             st.rerun()
2630
2631     if st.button("🔍 Comprobar Tesseract"):
2632         st.code(tesseract_info())
2633
2634     st.divider()
2635
2636     st.subheader("📊 Inventario Actual")
2637     conn = sqlite3.connect("arsenal_legacy.db")
2638     df = pd.read_sql_query("SELECT * FROM inventario", conn)
2639     conn.close()
2640     st.dataframe(df, width='stretch', hide_index=True)
2641
2642     # Chat principal
2643     st.subheader("💬 Asistente Inteligente")
2644
2645     # Inicializar historial de mensajes
2646     if "messages" not in st.session_state:
2647         st.session_state.messages = []
2648
2649     # Mostrar historial de conversación
2650     chat_container = st.container()
2651
2652     with chat_container:
2653         for msg in st.session_state.messages:
2654             with st.chat_message(msg["role"]):
2655                 st.write(msg["content"])
2656
2657     # Input del usuario
2658     if prompt_user := st.chat_input("Escribe tu comando o consulta..."):
2659         # Añadir mensaje del usuario al historial
2660         st.session_state.messages.append({"role": "user", "content": prompt_user})
2661
2662     # Procesar comando
2663     with st.spinner("⚙ Procesando..."):
2664         try:
2665             out = ejecutar_con_herramientas(prompt_user)
2666         except Exception as e:
2667             out = f"❌ Error crítico: {str(e)}"

```

```
2668
2669     # Añadir respuesta al historial
2670     st.session_state.messages.append({"role": "assistant", "content": out})
2671
2672     # Recargar interfaz para mostrar actualización
2673     st.rerun()
```

ANEXO III: GLOSARIO

A

AASTP-1 / AASTP-5

Publicaciones OTAN para seguridad de municiones y explosivos.

ABAC (Attribute-Based Access Control)

Control de acceso basado en atributos (nivel, unidad, rol, sensibilidad, clasificación). Modelo más flexible que RBAC, permite decisiones granulares considerando múltiples dimensiones de contexto y seguridad.

ACID (Atomicity, Consistency, Isolation, Durability)

Propiedades fundamentales de transacciones en bases de datos: Atomicidad (todo o nada), Consistencia (estado válido garantizado), Aislamiento (transacciones independientes) y Durabilidad (cambios persistentes). SQLite del prototipo implementa ACID para garantizar integridad del inventario de municiones.

Adapter (patrón)

Componente arquitectónico que encapsula un servicio externo (Vector DB, SQL, IA, ERP) para estandarizar integración, seguridad y operación reproducible, desacoplando el núcleo del SIGLM de dependencias externas.

Agente

Módulo especializado dentro del sistema (clasificación, extracción, recuperación, generación de respuestas). Coordinado por un orquestador central que planifica, ejecuta y gestiona fallos.

Alucinación

Generación de contenido incorrecto con alta confianza en modelos LLM; riesgo crítico en contexto de defensa mitigado con *grounding*, verificación y módulos deterministas.

Análisis predictivo

Uso de modelos de ML para anticipar consumos y necesidades futuras de municiones, apoyando planificación, preposicionamiento y asignación de recursos logísticos.

Anclaje / Grounding

Regla fundamental: toda afirmación del modelo debe estar respaldada por evidencia verificable (contexto recuperado, datos certificados, cálculos auditables) con trazabilidad completa.

Angular

Framework de desarrollo *frontend* basado en TypeScript para aplicaciones web complejas; arquitectura objetivo para la interfaz de usuario del SIGLM en producción, con gestión de estado, *routing* y componentes modulares.

Auditoría / Log de auditoría

Registro inmutable de operaciones (quién, cuándo, qué, resultado, origen del dato). Esencial para trazabilidad, acreditación, cumplimiento normativo y análisis forense.

Autenticación multifactor (MFA)

Mecanismo de seguridad que requiere múltiples pruebas de identidad (usuario/contraseña + token hardware/software).

API (Application Programming Interface)

Interfaz de programación que define métodos y contratos para comunicación entre servicios.

Aprendizaje Supervisado

Técnica de ML donde se entrenan modelos con datos etiquetados; usado para clasificación de municiones y validación de documentos.

Aprendizaje No Supervisado

Técnica donde el modelo identifica patrones sin etiquetas previas; aplicable a *clustering* de familias de municiones por características.

B

Backend

Componente servidor de la arquitectura que implementa la lógica de negocio, gestión de bases de datos, autenticación, autorización y procesamiento de solicitudes.

Backoff Exponencial

Estrategia de reintentos donde el tiempo de espera aumenta exponencialmente (2s, 4s, 8s...) ante fallos transitorios; implementada en `invocar_llm_seguro()` del prototipo para manejar errores temporales del modelo.

Base de Datos Relacional

Sistema SQL para datos maestros y transaccionales (inventarios, movimientos, metadatos). Integrada con componentes vectoriales y almacenes de documentos.

Base de Datos Vectorial

Sistema especializado para almacenar y recuperar *embeddings* de alta dimensión; optimizado para búsqueda semántica en documentación técnica y normativa.

Baseline

Línea base o método de referencia inicial en desarrollo; en el prototipo, la extracción con PyPDFLoader actúa como *baseline* antes de recurrir a OCR selectivo.

BERT

Modelo de lenguaje preentrenado bidireccional; base para tareas de clasificación y extracción de entidades en textos técnicos militares.

BM25

Algoritmo probabilístico de recuperación de información basado en frecuencia de términos; alternativa a búsqueda vectorial para documentos muy estructurados.

Búsqueda Semántica

Búsqueda que comprende significado y contexto mediante *embeddings*; superior a coincidencia exacta de palabras para documentación heterogénea.

Búsqueda Vectorial

Método que representa documentos como vectores en espacio multidimensional, permitiendo recuperación por similitud conceptual usando FAISS/HNSW.

C

Cadena de Pensamiento (Chain-of-Thought)

Técnica de *prompting* que desglosa razonamiento en pasos intermedios; mejora explicabilidad en validaciones de compatibilidad y trazabilidad.

Callback

Función invocada automáticamente en respuesta a eventos o cambios de estado; en arquitecturas reactivas como Streamlit, se evitan *callbacks* explícitos mediante re-ejecución del script completo.

ChromaDB

Base de datos vectorial embebida de código abierto para almacenamiento y recuperación de embeddings; alternativa ligera a Milvus utilizada en el prototipo SIGLM para validar búsqueda semántica sin infraestructura distribuida.

Chunking (Fragmentación)

Proceso de dividir documentos largos en fragmentos manejables (*chunks*) con solapamiento; en SIGLM se usa RecursiveCharacterTextSplitter con tamaño 2000 caracteres y solapamiento de 250 para optimizar recuperación RAG.

Cifrado en Reposo / en Tránsito

Protección criptográfica de datos almacenados (AES-256) y en comunicaciones (TLS). Requisito crítico para soberanía del dato.

Clasificación de Seguridad / Etiquetado

Metadatos de seguridad (Secreto, Confidencial, Interno) que controlan acceso, filtrado y segregación de información.

CommonMark

Especificación estándar de Markdown que define sintaxis consistente; utilizada en el prototipo para formatear respuestas del chatbot con énfasis, listas y bloques de código.

Compatibilidad (de Municiones)

Reglas OTAN para co-almacenamiento/co-transporte según grupo de riesgo y grupo de compatibilidad; algunas combinaciones están prohibidas.

Compensación (Transacción Compensatoria)

En patrón Saga, operación que deshace semánticamente efectos previos cuando un paso falla; no es rollback técnico sino lógica de negocio.

Contextualized Word Representations

Representaciones de palabras que varían según contexto (ELMo, BERT); mejoran extracción de entidades en documentación técnica.

Control de Acceso

Mecanismo de seguridad (RBAC, ABAC) que regula quién accede a qué recursos con qué permisos, basado en roles/atributos y logs.

Cuantización (Quantization)

Reducción de precisión de parámetros (32→8/4 bits) para bajar coste computacional sin degradar rendimiento significativamente.

D

Data Lake (Lago de datos)

Repositorio centralizado para documentos en bruto (PDF, imágenes OCR, mensajes operativos) y derivados procesados; fuente documental del sistema.

Data Lineage (Trazabilidad de datos)

Trazabilidad completa del dato: documento origen → extracción → transformación → estado en BD → uso en respuestas. Clave para auditoría.

Dataset

Conjunto de datos para entrenar, validar y probar modelos de ML; en SIGLM incluye documentación histórica y casos de uso validados.

Deep Learning (DL)

Rama de ML con redes neuronales profundas (múltiples capas) para aprender representaciones complejas de texto y documentos.

Desmilitarización

Proceso seguro de baja/neutralización de material explosivo; operación crítica que requiere documentación técnica y de trazabilidad completa en el ciclo de vida.

Desacoplamiento

Propiedad arquitectónica de microservicios: independencia y ausencia de dependencias directas; facilita evolución y escalabilidad.

Distancias de Seguridad (QD, Quantity Distance)

Separaciones mínimas respecto a instalaciones, vías y otros almacenes, función de NEQ, tipo de construcción y grupos de riesgo; reguladas por AASTP.

Docker

Plataforma de *containerización* para empaquetar código y dependencias; facilita *deployment* consistente en infraestructura *on-premise*.

Datos Clasificados

Información de defensa con niveles de confidencialidad (según normativas OTAN) que requiere segregación, cifrado y control de acceso estricto.

E

Elasticsearch

Motor de búsqueda y análisis distribuido, alternativa a índices vectoriales para búsqueda full-text en documentación técnica.

ELMo (Embeddings from Language Models)

Método para generar representaciones contextualizadas usando LSTM bidireccionales; precursor de BERT.

Embedding

Representación vectorial numérica de palabras, documentos o conceptos en espacio de baja dimensión; base de búsqueda semántica y recuperación.

Enriquecimiento con Metadatos

Adición de atributos (vigencia, tipo documental, clasificación, sistema de armas, etiquetas temáticas) para mejorar recuperación y control.

ERP (Enterprise Resource Planning)

Sistema corporativo de gestión (SAP, Oracle); en SIGLM se plantea integración bidireccional para inventario, logística y reportería.

Escalabilidad

Capacidad de un sistema para manejar crecimiento en volumen de datos, usuarios o carga sin degradar rendimiento; crítica en defensa.

Estándar

Especificación técnica/normativa (ISO/IEC, AASTP, OTAN) que define requisitos y procedimientos, referencia para cumplimiento.

Extracción de Información

Tarea de PLN que identifica y extrae datos relevantes estructurados (NEQ, compatibilidad, grupos de riesgo) de texto no estructurado.

F

FAISS (Facebook AI Similarity Search)

Librería optimizada para búsqueda de similitud vectorial y vecinos más cercanos; usado como motor de índices en *retriever*.

Feature

Característica o atributo de datos usado como entrada para modelos de ML; en SIGLM: grupos de riesgo, NEQ, vigencia.

Faithfulness

Métrica que evalúa si respuestas del modelo RAG están fundamentadas en contexto recuperado; medida de confiabilidad en dominio.

Fallback

Estrategia de respaldo que se activa cuando el método principal falla; en ingesta híbrida, si OCR no produce texto válido, el sistema conserva extracción original como *fallback*.

Few-Shot Prompting

Prompting con pocos ejemplos (2-3) para guiar modelo sin entrenamiento adicional; reduce alucinaciones mediante ejemplos de referencia.

Fine-Tuning (Ajuste Fino)

Adaptación de modelo preentrenado a dominio específico (documentación militar, vocabulario logístico) con dataset reducido.

Flujo de Ingesta (Ingestion Pipeline)

Cadena de pasos para transformar documentos: OCR, fragmentación (*chunking*), *embeddings*, indexación, extracción de metadatos y validación.

Framework

Estructura conceptual y de código que proporciona componentes reutilizables y patrones para desarrollo de software escalable.⁰

Frontend

Componente cliente de la arquitectura que implementa la interfaz de usuario (UI) con la que interactúan los operadores

Function Calling

Mecanismo de LangChain donde el LLM genera llamadas estructuradas en JSON a funciones predefinidas decoradas con `@tool`; permite al modelo invocar dinámicamente herramientas especializadas según contexto conversacional.

G

Generación de Lenguaje Natural

Tarea de PLN que produce texto coherente; en RAG se usa para generar respuestas sobre compatibilidades, movimientos, etc.

Generación Aumentada por Recuperación (RAG)

Arquitectura que recupera evidencia de fuentes externas antes de generar, reduciendo alucinaciones y mejorando precisión en dominio especializado.

Gestión del Ciclo de Vida

Conjunto de controles desde recepción, almacenamiento, movimientos, inspecciones, baja con trazabilidad completa y reglas de seguridad.

Gestión Documental

Área de TIC que se ocupa del almacenamiento, organización, recuperación y control de acceso a documentos; núcleo del SIGLM.

GGUF

Formato de archivo para modelos de lenguaje cuantizados optimizado para inferencia eficiente; usado por Ollama para ejecutar Mistral con requisitos reducidos de memoria.

GPT (Generative Pre-Trained Transformer)

Familia de LLM generativos; referencia para capacidades de comprensión y generación adaptables a dominio de defensa.

Grounding

Práctica de fundamentar respuestas en fragmentos reales de documentos recuperados; reduce alucinaciones, mejora trazabilidad y confianza.

gRPC

Framework de RPC de alto rendimiento usando HTTP/2; alternativa a REST para comunicación inter-servicios de baja latencia.

Grupo de Compatibilidad (A–S)

Clasificación OTAN en 19 grupos de letras (A, B, C, ..., S) que determina qué municiones pueden co-almacenarse; crítica para segregación.

Grupo de Riesgo (1.1–1.6)

Clasificación OTAN de explosividad; 1.1 es detonación en masa, 1.6 de menor peligro. Condiciona almacenamiento, QD y transporte.

GPU

Hardware para cómputo paralelo; reduce latencia en búsqueda vectorial y procesamiento de *embeddings* a escala.

H

Hardware Security Module (HSM)

Dispositivo físico criptográfico que gestiona claves y operaciones sensibles de forma aislada; para cumplimiento de soberanía del dato.

Heurística

Método de decisión basado en reglas prácticas derivadas de experiencia; `_page_needs_ocr()` implementa heurística con criterios de longitud, densidad alfanumérica y proporción de símbolos para determinar necesidad de OCR.

Hierarchical Navigable Small World (HNSW)

Estructura de índice aproximado de vecinos para espacios de alta dimensión; equilibra latencia, memoria y *recall*.

Human-in-the-Loop

Supervisión humana en puntos críticos: confirmación de transacciones de municiones, validación de extracción, autorización de excepciones.

I

ICL (In-Context Learning)

Capacidad del LLM de aprender una tarea con ejemplos en el *prompt* (*few-shot*) sin reentrenamiento adicional.

Indexación (Vectorial)

Proceso de almacenar *embeddings* y construir índices (FAISS, HNSW, Milvus) para recuperación eficiente por similitud.

Inference

Uso de un modelo entrenado para hacer predicciones en nuevos datos; también llamado ejecución en producción.

Infraestructura

Componentes base de sistemas: servidores, redes, almacenamiento; en SIGLM se plantea *on-premise* para soberanía del dato.

Ingeniería de Prompts

Disciplina de diseñar y refinar instrucciones (*prompts*) para obtener respuestas óptimas de LLM; incluye rol, tarea, ejemplos, *grounding*.

Ingesta

Entrada y procesamiento de documentación heterogénea (PDF, escaneados OCR, mensajes operativos); etapa inicial del pipeline.

Instruction Prompting

Estilo de *prompt* orientado a "seguir instrucciones explícitas" para tareas nuevas sin dataset etiquetado específico.

Inteligencia Artificial

Campo de informática que desarrolla sistemas capaces de realizar tareas que requieren inteligencia humana; base conceptual del SIGLM.

Interfaz

Punto de contacto entre componentes que define métodos de interacción; en SIGLM: APIs seguras, CLI, portales autenticados.

J

JSON

Formato de intercambio de datos ligero, usado en APIs REST y documentación estructurada.

K

Kafka

Plataforma de *streaming* distribuida para eventos; gestiona flujos de consumos/movimientos de municiones en tiempo real.

Kubernetes

Orquestador de contenedores que automatiza el despliegue, escalado y gestión de aplicaciones containerizadas en *on-premise*.

L

Label

Etiqueta o clase asignada a instancia de datos; esencial en aprendizaje supervisado para clasificación de municiones.

LangChain

Framework de orquestación Python para desarrollo de aplicaciones con LLMs; proporciona abstracciones para cadenas RAG, gestión de *prompts*, *binding* de herramientas y composición de componentes de IA en el prototipo SIGLM.

LDAP (Lightweight Directory Access Protocol)

Servicio de directorio para autenticación y gestión de roles; en SIGLM se integra con directorio institucional de defensa.

Large Language Model (LLM)

Modelo neural profundo entrenado en textos masivos; capaz de comprensión, extracción y generación de lenguaje complejo adaptado a dominio.

Lemmatización

Proceso de PLN que reduce palabras a forma canónica (lema), facilitando análisis y búsqueda en documentación técnica.

Logística de Municiones

Procesos de recepción, almacenamiento, solicitudes, devoluciones, asignaciones, desmilitarización con restricciones de seguridad y trazabilidad completa.

LSTM (Long Short-Term Memory)

Tipo de red neuronal recurrente con capacidad de memoria a largo plazo; arquitectura base de Tesseract OCR v5 con modelos LSTM para reconocimiento de caracteres y de ELMo para *embeddings* contextualizados.

M

Machine Learning (ML)

Rama de IA que permite sistemas aprender patrones de datos; usado en SIGLM para predicción de consumos, clasificación.

Markdown

Lenguaje de marcado ligero para formatear texto con sintaxis simple; usado en respuestas del chatbot del prototipo para estructurar información con encabezados, listas, énfasis y bloques de código.

Mensajes Operativos

Mensajería no estructurada desde unidades (consumos, movimientos, solicitudes) que el sistema interpreta extrayendo entidades logísticas (NIIN, NEQ, etc.).

Metadatos

Datos que describen otros datos/documentos: vigencia, procedencia, clasificación, tipo documental, sistema de armas, ubicación, etc.

Middleware

Software que actúa como intermediario entre aplicaciones, facilitando comunicación segura y gestión de datos.

Milvus

Base de datos vectorial de código abierto citada como opción para persistencia de *embeddings* y búsqueda semántica eficiente.

Mistral

Modelo de lenguaje de gran tamaño de 7 mil millones de parámetros desarrollado por Mistral AI; utilizado en el prototipo SIGLM ejecutándose localmente vía Ollama para generación de respuestas y razonamiento con soporte de contextos hasta 32k tokens.

MLOps (Machine Learning Operations)

Prácticas para operar y mejorar modelos en producción: monitorización, reentrenos, métricas, ciclos de mejora continua.

Modelo

Representación matemática entrenada que mapea inputs a outputs; componente central de sistemas de IA.

Módulo Determinista

Componente algorítmico exacto para validación de restricciones críticas (compatibilidades OTAN, QD, NEQ máxima); complementa LLM.

MongoDB

Base de datos NoSQL orientada a documentos; flexible para almacenar documentos y metadatos heterogéneos.

Motor de Búsqueda

Sistema que indexa contenido y permite recuperar información relevante mediante consultas; puede ser *full-text* o vectorial.

mTLS (Mutual TLS)

Protocolo de seguridad donde cliente y servidor se autentican mutuamente; asegura comunicaciones inter-servicios.

Multi-turno

Conversaciones con múltiples intercambios consecutivos donde el sistema mantiene contexto entre turnos; en SIGLM permite flujos como: detección de albarán → presentación de movimientos → confirmación del usuario → ejecución de transacción.

mxbai-embed-large

Modelo de *embeddings* de 335 millones de parámetros optimizado para español e inglés; genera vectores de 1024 dimensiones utilizados en el prototipo para búsqueda semántica sobre documentación técnica militar con ejecución completamente offline.

N

Named Entity Recognition (NER)

Tarea de PLN que identifica y clasifica entidades nombradas (NIIN, localizaciones, unidades) en textos operativos.

NEQ (Nett Explosive Quantity)

Masa equivalente de explosivo en kg; usado para cálculos de QD, límites máximos acumulados y validación de compatibilidad.

NgRx

Librería de gestión de estado reactivo para aplicaciones Angular basada en arquitectura Redux; mencionada como componente de la arquitectura frontend objetivo para el SIGLM en producción, ausente en el prototipo Streamlit simplificado.

NIIN (National Item Identification Number) / NSN (NATO Stock Number)

Identificadores logísticos normalizados OTAN para artículos; el sistema los extrae y valida desde mensajes y documentos.

NIST

Instituto Nacional de Estándares y Tecnología; productor de normas de seguridad cibernética y arquitecturas *zero-trust*.

NLP (Natural Language Processing, PLN)

Técnicas para procesar lenguaje: extracción de entidades, clasificación, QA sobre manuales, análisis de mensajes operativos.

NoSQL

Familias de BD no relacionales; flexibles en esquema para documentos y datos heterogéneos.

Norma

Especificación de requisitos y procedimientos técnicos; fuente de autoridad en defensa (AASTP, ISO/IEC, OTAN).

O

OCR (Optical Character Recognition)

Reconocimiento óptico de caracteres, consiste en el reconocimiento de texto en imágenes o documentos escaneados; requiere benchmarking y motores robustos (Tesseract, PP-OCR).

Ollama

Runtime de código abierto para ejecutar modelos de lenguaje de gran tamaño localmente sin dependencias de servicios *cloud*; proporciona API compatible con OpenAI y permite al prototipo SIGLM cumplir requisitos de soberanía de datos ejecutando Mistral y *mx-bai-embed-large* on-premise.

On-Premise

Despliegue en infraestructura propia, no en nube; relevante para soberanía del dato y seguridad en defensa.

Orquestador (Orchestrator)

Componente que analiza consulta, planifica flujo, coordina agentes, gestiona fallos y registra trazabilidad completa.

Orquestación

Coordinación automatizada de múltiples servicios y contenedores en sistemas distribuidos; esencial para resiliencia.

OTAN

Organización del Tratado del Atlántico Norte; fuente de normas (AASTP, grupos de riesgo/compatibilidad) aplicadas en defensa española.

OWASP

Open Web Application Security Project; publica top 10 de vulnerabilidades en aplicaciones y LLM (prompt injection, information disclosure, etc.).

P

Pañol

Dependencia/almacén asociado a unidad militar o instalación; origen/destino de movimientos de municiones con trazabilidad.

Payload

Contenido útil de un mensaje o solicitud API; información sustancial transmitida.

PEFT (Parameter-Efficient Fine-Tuning)

Técnica de ajuste fino eficiente con menos parámetros/recursos (LoRA, Adapter); mencionada para mejora continua del modelo.

Permisos

Derechos de acceso asignados a roles/usuarios en sistema de control; granularidad según operación (lectura, escritura, autorización).

PII (Personally Identifiable Information)

Datos que identifican individuos; protegidos por leyes de privacidad; en defensa, incluye información de personal y ubicaciones.

PIL (Python Imaging Library) / Pillow

Biblioteca Python para procesamiento y manipulación de imágenes; utilizada en pipeline de OCR del prototipo para convertir pixmaps de PyMuPDF a formato compatible con Tesseract.

Pipeline

Flujo secuencial de procesamiento: ingesta, extracción, *embeddings*, indexación, validación, generación de respuestas.

Polvorín

Almacén de municiones/explosivos con requisitos estrictos de seguridad por grupo de riesgo, QD, NEQ máxima, compatibilidades.

PostgreSQL

BD relacional robusta, alternativa para datos maestros y transaccionales del SIGLM.

Precision

Métrica de evaluación que mide proporción de resultados recuperados que son realmente relevantes.

Preprocesamiento

Etapas inicial en ML que limpia, normaliza y transforma datos crudos para entrada al modelo.

Procesamiento de Lenguaje Natural

Campo de IA para máquinas entiendan, analicen y generen lenguaje humano; fundamental en SIGLM.

Privacidad

Principio de seguridad que protege confidencialidad de datos y derechos individuales en sistemas de información clasificada.

Prompt

Instrucción textual que guía comportamiento del LLM; en SIGLM se estructura con rol, tarea, contexto, *grounding* y CoT.

Prompt Injection

Ataque adversarial que intenta manipular instrucciones o extraer información; mitigación mediante sanitización y defensa en profundidad.

Protocolo

Conjunto de reglas estandarizadas que definen comunicación entre sistemas; en defensa: AASTP, OTAN.

PyMuPDF (fitz)

Biblioteca Python de alto rendimiento para procesamiento de documentos PDF; extrae texto nativo accediendo directamente a objetos del PDF, significativamente más rápida que alternativas como pypdf para documentación generada digitalmente.

Python

Lenguaje de programación interpretado de alto nivel; utilizado para implementar toda la lógica de negocio, orquestación, procesamiento documental y validaciones del prototipo SIGLM con más de 1800 líneas de código.

Q

QD (Quantity Distance)

Véase Distancias de Seguridad.

R

RAG

Véase Generación Aumentada por Recuperación.

Ranking

Ordenamiento de resultados de búsqueda por relevancia usando algoritmos (BM25, scoring semántico, reranking).

RBAC (Role-Based Access Control)

Control de acceso basado en roles (operador, supervisor, comandante); pilar de seguridad del SIGLM con integración LDAP.

Recall

Métrica que mide proporción de resultados relevantes recuperados del total existente; en busca de municiones crítico.

Red Neuronal

Estructura computacional inspirada en redes biológicas; base de deep learning y transformadores.

Regresión

Tarea de ML que predice valores continuos (p. ej., consumo futuro de municiones) basándose en características.

Regulación

Marco legal que establece requisitos y normativas; en defensa: AASTP, ISO/IEC 27001, IA Act UE.

Relevancia

Grado en que documento recuperado responde adecuadamente a consulta; métrica central en RAG.

Representación de Texto

Conversión de texto a formato numérico (bag-of-words, *embeddings*); base de recuperación semántica.

REST (Representational State Transfer)

Estilo arquitectónico para APIs web usando HTTP y operaciones CRUD estándar; usado en SIGLM.

Registro Inmutable / Firma Digital

Garantías de integridad de logs y acciones mediante timestamps y criptografía; especialmente para escrituras críticas.

Retrieval

Proceso de recuperación de información relevante de índices/BDs basándose en similitud o keywords.

Reranker

Componente que refina resultados iniciales ordenándolos por relevancia más precisa; mejora calidad de recuperación.

S

Saga (Patrón Saga)

Patrón para orquestar flujos multi-paso con estados y compensaciones ante fallos; muy útil en ingesta heterogénea.

Sanitización de Entradas

Validación/limpieza de *prompts* e inputs para reducir riesgos de manipulación (*prompt injection*, alucinaciones).

SC2N

Red de Mando y Control del sistema de defensa español.

Seguridad

En TIC, medidas y mecanismos para proteger sistemas, datos e infraestructura contra acceso no autorizado y amenazas.

Segregación

Medidas físicas/lógicas para separar municiones no compatibles: barreras, distancias mínimas, compartimentación.

Seq2Seq (Sequence-to-Sequence)

Arquitectura codificador-decodificador; base conceptual de transformadores, usada en tareas de traducción y generación.

Servicio Web

Componente de software que expone funcionalidades mediante APIs (REST, gRPC) en arquitectura distribuida.

SIGLM

Sistema Inteligente de Gestión Logística de Municiones (ficticio); propuesta del TFM para digitalizar optimizar y hacer inteligente la logística de municiones en Armada Española.

Similitud Semántica

Medida de cómo de similar es significado entre dos textos, calculada mediante distancia vectorial de *embeddings*.

Soberanía del Dato

Principio que datos de defensa deben residir en territorio nacional, no en servidores extranjeros; requisito crítico para SIGLM.

SQL

Lenguaje estándar para consultar y manipular bases de datos relacionales; usado para datos maestros y transaccionales.

SQLite

Motor de base de datos SQL embebido que implementa transacciones ACID en archivo único sin dependencias de servidor; utilizado en el prototipo para almacenar inventario de municiones con estructura equivalente a PostgreSQL de producción.

SSL/TLS

Protocolos de seguridad que cifran comunicaciones en red; TLS es versión moderna de SSL.

Streamlit

Framework Python para construcción rápida de aplicaciones web interactivas con renderizado reactivo; proporciona la interfaz de usuario del prototipo SIGLM incluyendo chatbot, file uploader y visualización de inventario con código declarativo mínimo.

T**TCP/UDP**

Protocolos de transporte: TCP garantiza entrega ordenada; UDP es más rápido pero sin garantías. Usados según requisitos de latencia/confiabilidad.

Tesseract

Motor de OCR de código abierto desarrollado originalmente por HP; versión 5 con modelos LSTM utilizada en el prototipo para extraer texto de páginas PDF escaneadas con configuración `--psm 6` optimizada para formularios.

Token (2FA / Segundo Factor)

Mecanismo adicional de autenticación (hardware o software) para operaciones de escritura críticas en municiones.

Tokenización

Proceso de PLN que divide texto en unidades discretas (palabras, caracteres, subpalabras); preprocesamiento para *embeddings*.

Tool (LangChain)

Función especializada decorada con `@tool` de LangChain que encapsula una capacidad específica invocable dinámicamente por el LLM; el prototipo define cinco tools principales: `consultar_inventario`, `consultar_capacidad_polvorin`, `consultar_documentacion_tecnica`, `ejecutar_movimiento_municion` y `extraer_municion`.

TLS

Protocolo de cifrado en tránsito para comunicaciones seguras inter-servicios; requerido en defensa.

Training

Proceso de ajuste de parámetros de un modelo ML usando datos de entrenamiento.

Trazabilidad

Capacidad de reconstruir qué documentos/datos sustentan una decisión o respuesta, con logs inmutables y metadatos auditable.

Transformer

Arquitectura neural basada en mecanismo de atención; revolucionaria para PLN, base de BERT, GPT y modelos avanzados.

Tree of Thoughts

Técnica avanzada de *prompt engineering* que explora múltiples caminos de razonamiento simultáneamente; mejora coherencia en decisiones críticas.

Tuning

Véase Fine-Tuning; también ajuste de hiperparámetros de modelos.

U

UX (User Experience)

Experiencia de usuario, conjunto de percepciones y respuestas resultantes del uso de un sistema; el prototipo implementa principios de UX como visibilidad del estado (tabla de inventario), prevención de errores (confirmaciones) y consistencia (patrón uniforme de interacción).

V

Vector

Representación matemática de datos en espacio multidimensional; base de búsqueda semántica, *embeddings* y recuperación.

Vector DB / Vector Store

Base de datos especializada (Milvus, FAISS, Pinecone) para almacenar y recuperar *embeddings* eficientemente.

Vigencia (Normativa/Documental)

Atributo de metadatos que indica período de validez; crítico para evitar usar normativa obsoleta en validaciones.

Validación

Evaluación de calidad y corrección de datos, procesos y decisiones; crítica en sistemas de defensa para garantizar integridad operativa.

W

WebSocket

Protocolo de comunicación bidireccional sobre TCP; permite conexiones persistentes para aplicaciones en tiempo real.

Word2Vec

Técnica que genera *embeddings* de palabras capturando semántica; revolucionaria en representación textual y precursora de modelos modernos.

Workflow

Secuencia de pasos y procesos para completar una tarea; especialmente importante en sistemas de gestión documental y logística.

X

XML

Formato de marcado extensible utilizado para estructurar y transportar datos; usado en documentación militar y intercambio inter-sistemas.

Z

Zero-Shot Prompting

Prompting sin ejemplos para guiar modelo; eficiente en tokens pero con mayor riesgo de alucinaciones comparado con *few-shot*.

Zero-Trust

Modelo de seguridad que verifica cada acceso de forma independiente, sin confianza implícita; alineado con estándares NIST y defensa.

Zero Trust Architecture

Arquitectura de seguridad que asume todas las conexiones son potencialmente hostiles; requiere autenticación continua y autorización granular.