Universidad Zaragoza
1474

Trabajo Fin de Grado

Inteligencia Artificial *On-Premise* para el análisis de IoCs con datos privados

On-Premise Artificial Intelligence model for IoC analysis based on private data

Autor

Nicolás Pueyo Soria

Directores

Héctor Francia Molinero

Ricardo Julio Rodríguez Fernández

ESCUELA DE INGENIERÍA Y ARQUITECTURA

2026

Agradecimientos

A mi familia y amigos, por ser y por estar.

Al equipo de Inetum por su buena acogida y su buen ambiente.

Y a Ricardo por su supervisión y sus acertadas revisiones.

Resumen

El presente Trabajo Fin de Grado aborda el diseño, desarrollo e implementación de una plataforma privada y local (*on-premise*) de asistencia conversacional adaptada a la ingeniería de ciberseguridad para el análisis seguro de indicadores de compromiso. Ante el riesgo de fuga de información corporativa sensible a través de servicios comerciales de terceros, se propone una arquitectura modular de código abierto que garantiza la soberanía absoluta de los datos, operando bajo restricciones estrictas de hardware (procesamiento exclusivo por CPU y memoria RAM). La metodología de desarrollo implementada se basa en un sistema de generación aumentada por recuperación con un enfoque híbrido de extracción: por un lado, consume de forma automatizada grafos de conocimiento relacionales bajo el estándar STIX 2.1 desde la plataforma de ciberinteligencia OpenCTI y, por otro, recupera contexto analítico adyacente mediante búsquedas por similitud semántica en la base de datos vectorial ChromaDB. La interfaz interactiva del analista se gestiona mediante OpenWebUI, orquestando los flujos a través de un servidor interconectado con un *backend* de servicios en FastAPI.

Los resultados cualitativos y cuantitativos, obtenidos mediante una rúbrica formal de evaluación sobre siete tipologías de consulta, demuestran la viabilidad del sistema y una alta estabilidad en los tiempos de inferencia local administrados por *Ollama*. El análisis comparativo concluye que existe un balance crítico entre coste de cómputo y madurez cognitiva: mientras que el modelo ligero *Lily-Cybersecurity-7B-v0.2* destaca por su velocidad, muestra debilidades ante la ambigüedad conceptual; por su parte, *CyberOSS-2.0-20B* exhibe un razonamiento lógico superior para la resolución de grafos de atribución complejos. También se comprueba que, aunque los tiempos de respuesta tienden a estabilizarse en torno a unos valores medios, factores como la complejidad de las tareas o la densidad contextual pueden influir en torno a un 30 % en el tiempo invertido por un modelo en generar una respuesta. Finalmente, se establece que el rendimiento de los modelos pequeños se incrementa notablemente al guiar sus tareas mediante comandos concretos en lugar de peticiones generales, validando la utilidad operativa de la plataforma como asistente local en la respuesta automatizada ante incidentes.

Palabras clave: Inteligencia Artificial, *On-Premise*, RAG, OpenCTI, ChromaDB, OpenWebUI, Indicadores de Compromiso, Ciberseguridad.

Abstract

This project addresses the design, development, and implementation of a private, local (on-premise) conversational assistance platform adapted to cybersecurity for the secure analysis of indicators of compromise. In response to the risk of sensitive corporate information leakage through third-party commercial services, this work proposes a modular open-source architecture that guarantees full data sovereignty, operating under strict hardware constraints, with processing performed exclusively on CPU and RAM. The implemented development methodology is based on a retrieval-augmented generation system with a hybrid extraction approach: on the one hand, it automatically consumes relational knowledge graphs under the STIX 2.1 standard from the OpenCTI cyber threat intelligence platform and, on the other hand, it retrieves adjacent analytical context through semantic similarity searches in the ChromaDB vector database. The analyst's interactive interface is managed through OpenWebUI, orchestrating the workflows through a server interconnected with a FastAPI service backend.

The qualitative and quantitative results, obtained through a formal evaluation rubric across seven types of queries, demonstrate the feasibility of the system and high stability in local inference times managed by *Ollama*. The comparative analysis concludes that there is a critical trade-off between computational cost and cognitive maturity: while the lightweight model *Lily-Cybersecurity-7B-v0.2* stands out for its speed, it shows weaknesses when facing conceptual ambiguity; in contrast, *CyberOSS-2.0-20B* exhibits superior logical reasoning for resolving complex attribution graphs. It is also shown that, although response times tend to stabilize around average values, factors such as task complexity or contextual density can influence the time required by a model to generate a response by around 30%. Finally, this work establishes that the performance of smaller models increases notably when their tasks are guided through concrete commands rather than general requests, validating the operational usefulness of the platform as a local assistant for automated incident response.

Keywords: Artificial Intelligence, On-Premise, RAG, OpenCTI, ChromaDB, OpenWebUI, Indicators of Compromise, Cybersecurity.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

1.1. Contexto y motivación

En los últimos años, la adopción de tecnologías basadas en Inteligencia Artificial (IA) se ha consolidado como una práctica habitual en el entorno corporativo para optimizar procesos y mejorar la productividad de los empleados [1, 2]. En particular, los Modelos de Lenguaje de Gran Escala (LLM, por sus siglas en inglés) han demostrado ser herramientas de utilidad para tareas como la síntesis de información y la redacción de informes técnicos. No obstante, este rápido despliegue tecnológico también introduce retos de seguridad significativos. La falta de formación específica en ciberseguridad de muchos usuarios puede dar lugar a malas prácticas en el manejo de la IA, como la exposición accidental de datos corporativos sensibles al enviarlos a servicios comerciales de terceros [1].

De forma paralela, la defensa frente a incidentes digitales se apoya cada vez más en la Ciberinteligencia de Amenazas (CTI, por sus siglas en inglés) [3], la cual permite estructurar y correlacionar datos sobre potenciales ataques mediante herramientas de código abierto como OpenCTI [4]. Sin embargo, la gestión diaria de estos repositorios de conocimiento sigue requiriendo una considerable inversión de tiempo por parte de los analistas para procesar e interpretar la información técnica de los nuevos Indicadores de Compromiso (IoC, por sus siglas en inglés).

En este escenario, surge la oportunidad de integrar la IA generativa como un asistente para facilitar la labor de los analistas en la interpretación y análisis de CTI. Para llevar a cabo esta integración en un contexto empresarial de manera segura, es necesario descartar las soluciones basadas en la nube y optar por modelos locales y privados. De este modo, este proyecto propone el desarrollo de una herramienta desplegada en un servidor privado que, mediante modelos de lenguaje especializados ejecutados localmente, permita a los ingenieros de ciberseguridad consultar, comparar y contrastar la ciberinteligencia de la organización con nuevos indicadores en un tiempo razonable, garantizando la soberanía de los datos sensibles de la empresa.

El presente proyecto se ha desarrollado en el marco de colaboración con la división de Zaragoza de Inetum España S.A., multinacional especializada en consultoría tecnológica y provisión de servicios de TI. En Aragón, Inetum desempeña un papel relevante al gestionar las infraestructuras y los servicios informáticos de entidades públicas de

gran envergadura, como la Diputación General de Aragón y el Servicio Aragonés de Salud. Además, con el fin de garantizar la soberanía, la transparencia y la viabilidad de la arquitectura propuesta, el sistema se implementa de manera íntegra utilizando herramientas y soluciones de software de código abierto.

1.2. Objetivos del proyecto

El objetivo general de este proyecto es diseñar, implementar y evaluar una plataforma privada y local de asistencia conversacional orientada a la ingeniería de ciberseguridad, integrada con una plataforma de gestión de ciberinteligencia de amenazas. El sistema propuesto debe permitir la consulta interactiva y el análisis contextualizado de IoCs garantizando la soberanía de los datos sensibles y optimizando el rendimiento de modelos de lenguaje bajo restricciones estrictas de hardware basadas exclusivamente en procesamiento por CPU y memoria RAM.

Para alcanzar este fin, se proponen los siguientes objetivos específicos:

- **Despliegue y configuración de un entorno local aislado:** Implementar una infraestructura privada que albergue un entorno de ejecución para modelos de lenguaje y una interfaz gráfica de usuario accesible para los analistas, operando de manera local y sin dependencias de servicios externos en la nube.
- **Desarrollo de un *pipeline* de recuperación semántica:** Diseñar un mecanismo de indexación, almacenamiento vectorial y recuperación semántica de datos que permita alimentar las consultas del asistente conversacional con la ciberinteligencia y el contexto histórico de la organización de forma precisa.
- **Integración y consulta bidireccional con sistemas de ciberinteligencia:** Establecer una conexión con la API de la plataforma de ciberinteligencia OpenCTI para permitir que el asistente interactúe con el grafo de conocimiento existente, traduciendo consultas en lenguaje natural a búsquedas estructuradas de observables y amenazas.
- **Evaluación comparativa de modelos de lenguaje especializados:** Evaluar el rendimiento, el nivel de detalle técnico, la tasa de alucinaciones y la precisión de múltiples modelos de lenguaje de diferentes escalas de parámetros en tareas complejas de correlación y análisis de indicadores.

1.3. Estructura del documento

El presente documento se estructura de la siguiente forma. El Capítulo 2 explica los conocimientos teóricos previos a tener en cuenta sobre diferentes conceptos clave para entender el desarrollo del proyecto. El Capítulo 3 detalla la arquitectura del proyecto, qué componentes se han utilizado, qué tecnologías se han usado para cada uno de estos componentes, así como qué organización tienen y cómo se despliegan físicamente, y finalmente explica el flujo general del sistema. En el Capítulo 4 se detallan las pruebas que se han realizado al sistema, se recogen y presentan los resultados obtenidos y se realiza un análisis de los mismos. Finalmente, en el Capítulo 5 se analiza todo el

contexto y trabajo desarrollado para extraer unas conclusiones, y se reflexiona sobre qué otras funcionalidades se podrían añadir al proyecto.

Adicionalmente, se incluye el Anexo A, con el registro de tareas del proyecto y las horas dedicadas a las mismas.

Capítulo 2

Conceptos previos

En este capítulo se introducen los conceptos fundamentales de ciberseguridad e IA que es necesario conocer para entender el desarrollo del proyecto. Primero se introducen los conceptos de respuesta a incidentes e indicadores de compromiso. A continuación, se explican la ciberinteligencia de amenazas y la plataforma OpenCTI. Después se introducen los modelos de lenguaje a gran escala, seguido de los motores de inferencia local y finalmente la generación aumentada por recuperación.

2.1. Respuesta a incidentes e indicadores de compromiso

En el ámbito de la ciberseguridad corporativa, la Respuesta a Incidentes (IR, por sus siglas en inglés de *Incident Response*) constituye la metodología estructurada y sistemática que adopta una organización para la detección, contención, mitigación y recuperación frente a incidentes de seguridad y ciberataques. De acuerdo con los estándares de la industria, el objetivo primordial de la IR abarca tanto la prevención proactiva de ciberataques, como la minimización del impacto en el negocio en caso de que una intrusión se complete de manera exitosa.

Para operativizar esta estrategia, las organizaciones definen sus tecnologías, responsabilidades y procedimientos dentro de un Plan de Respuesta a Incidentes. Este plan actúa como el componente técnico de la gestión integral de incidentes, especificando cómo deben identificarse, aislarse y resolverse los diferentes vectores de ataque de forma coordinada. En la actualidad, la integración de soluciones de ciberseguridad asistidas por inteligencia artificial permite acelerar este proceso, ayudando a los analistas a predecir posibles ataques, automatizar alertas y correlacionar comportamientos sospechosos antes de que escalen a incidentes críticos.

Para el estudio del estado de la ciberinteligencia en el marco de la respuesta a incidentes, los analistas dependen de los IoCs. En el campo de la informática forense y el análisis de malware, un IoC se define como cualquier artefacto, registro o evidencia digital identificado en un sistema operativo, un dispositivo de red o un flujo de tráfico que permite constatar, con un determinado nivel de confianza, la presencia de una actividad maliciosa o el compromiso de un activo [5].

En el marco del estándar internacional STIX 2.1 [5], la arquitectura de datos sobre la que se fundamenta la plataforma OpenCTI utilizada en este proyecto, conviene establecer una distinción técnica entre un observable (SCO, *STIX Cyber Observable*) y un *indicador* (SDO, *STIX Domain Object*). Un observable representa un dato técnico en bruto desprovisto de intencionalidad. Una dirección IP, un dominio o el hash de un binario son observables que pueden pertenecer tanto a servicios legítimos de la infraestructura como a servidores del atacante. Un indicador es la entidad que aporta contexto semántico y de seguridad al observable. Un indicador asocia un patrón de búsqueda (definido bajo estándares como STIX Pattern, reglas YARA o firmas Sigma) a un observable específico, determinando un período de validez temporal, relaciones con campañas de actores de amenazas y un nivel de certeza técnica de maliciosidad.

Para el alcance del sistema desarrollado en este proyecto, se consideran y analizan los siguientes tipos de observables e indicadores de compromiso técnicos:

- **URLs** identificadas como maliciosas, habitualmente vinculadas a servidores de descarga de cargas útiles, portales de suplantación de identidad o paneles de control de botnets.
- **Hashes** de ficheros identificados como códigos dañinos. El uso de funciones de dispersión criptográfica permite identificar unívocamente muestras de malware. Para este proyecto se contemplan SHA-256, SHA-1 y MD5.
- **Direcciones IPv4** identificadas como maliciosas, asociadas comúnmente a infraestructura de Comando y Control, nodos de salida Tor o proxies utilizados para el escaneo de vulnerabilidades.
- **Dominios** web identificados como maliciosos, utilizados para enmascarar direcciones IP de infraestructura atacante mediante técnicas de resolución dinámica de nombres.

A nivel estratégico, la utilidad de estos IoCs se rige bajo la *Pirámide del Dolor* de David Bianco [6, 7]. Este modelo conceptual organiza los indicadores de amenazas según el grado de dificultad (o "dolor") que causa al atacante el hecho de que un defensor los detecte y bloquee:

1. **Base (hashes, IPs y dominios):** Constituyen el nivel más bajo. Son extremadamente sencillos y baratos de modificar para el adversario. Un atacante puede alterar un solo byte de su código para cambiar el hash criptográfico del malware, o rotar de dirección IP en segundos mediante servicios de red dinámica, eludiendo bloqueos automatizados estáticos.
2. **Medio (artefactos de red y de host):** Requiere que el atacante modifique los patrones de comunicación de sus herramientas o las trazas que estas dejan en los registros de los sistemas comprometidos.
3. **Cúspide (herramientas y TTPs):** Las Tácticas, Técnicas y Procedimientos (TTPs) [8] definen el comportamiento operativo real y los hábitos del atacante. Obligar a un atacante a cambiar sus TTPs es el logro de mayor valor defensivo, ya que le exige reestructurar por completo su metodología y reaprender tácticas de infiltración.

La integración de la IA propuesta en este trabajo encuentra su motivación en esta

jerarquía. Mientras que los sistemas automáticos de seguridad tradicionales (como cortafuegos o antivirus) son excelentes procesando de manera rígida los observables de la base (hashes e IPs), carecen de la capacidad cognitiva para relacionar estos datos con comportamientos complejos. La incorporación de un LLM permite a los ingenieros de ciberseguridad procesar el texto no estructurado y los grafos de conocimiento de OpenCTI para correlacionar observables simples con el comportamiento general de la amenaza en la cúspide de la pirámide, forzando un impacto real sobre el coste operativo del atacante.

2.2. Ciberinteligencia de amenazas y OpenCTI

CTI consiste en la recopilación, análisis y estructuración de información sobre amenazas activas o potenciales con el fin de anticipar, detectar y mitigar ataques de forma proactiva [3]. Esta disciplina permite a los equipos de seguridad comprender el contexto, las motivaciones y las metodologías de los adversarios, optimizando la toma de decisiones estratégicas y los tiempos de respuesta ante incidentes.

Para centralizar y procesar este conocimiento, las organizaciones emplean plataformas especializadas como OpenCTI, una herramienta de código abierto diseñada para estructurar, almacenar, organizar y visualizar información técnica y no técnica sobre ciberamenazas. El núcleo de OpenCTI se fundamenta en un grafo de conocimiento basado en el estándar internacional STIX 2.1. En esta arquitectura, la inteligencia se modela mediante dos elementos [9]: los nodos, que representan SDOs, y aristas, que representan conexiones entre entidades y las relacionan mediante SCOs y sirven como evidencia de actividad maliciosa.

2.3. Modelos de lenguaje de gran escala

Los LLM son sistemas basados en redes neuronales profundas diseñados para comprender, procesar y generar lenguaje natural. Para que el modelo pueda procesar el texto en lenguaje humano, este debe someterse a un proceso de tokenización, en el cual se divide el flujo de caracteres en unidades discretas menores llamadas *tokens* (que pueden ser palabras, subpalabras o caracteres individuales) [2].

En el contexto corporativo, la adopción de estos modelos se divide principalmente entre servicios de infraestructura gestionada en la nube y despliegues locales. Si bien el uso de servicios comerciales en plataformas en la nube proporciona una gran capacidad de cómputo y escalabilidad, la transferencia de datos hacia servidores externos introduce riesgos severos de exposición, fugas de información y fallas de cumplimiento normativo. Por esta razón, el estado del arte en ciberseguridad se orienta firmemente hacia el despliegue local de modelos de menor tamaño, garantizando la soberanía absoluta de los datos sensibles de la organización [1, 10].

2.4. Motores de inferencia local

Mientras que un LLM representa la estructura estática de la red neuronal, el motor de inferencia es el software de ejecución activo que realiza las operaciones algebraicas

necesarias para responder a las consultas del usuario. En un servidor privado que carece de aceleradores gráficos (GPU), el motor de inferencia debe ejecutar el modelo utilizando exclusivamente el procesador (vCPU en contextos de centros de datos) y la memoria RAM del sistema.

En este escenario de inferencia puramente basado en CPU, la velocidad del sistema se ve fuertemente limitada. El cuello de botella principal no radica en la capacidad de cómputo de la vCPU, sino en el ancho de banda físico de la memoria RAM, puesto que el procesador debe leer la totalidad de los parámetros del modelo para generar cada *token* individual. Mientras que las GPUs utilizan memoria especializada de gran velocidad con un ancho de banda de entre 600 y 1500 GB/s, la memoria RAM convencional de un servidor host suele transferir datos a velocidades de entre 50 y 200 GB/s [11].

Esta limitación física provoca que la asignación de hilos de ejecución en el motor de inferencia no escale de manera lineal con el número de núcleos virtuales asignados. Si se configuran demasiados hilos de ejecución concurrentes en el motor de inferencia para procesar un modelo pesado, los hilos terminarán compitiendo por los mismos canales físicos de acceso a la memoria RAM. Esta contención de memoria genera un cuello de botella técnico que degrada el rendimiento del sistema [12] y reduce la tasa de generación de texto (*tokens* por segundo), por lo que la optimización de estos sistemas requiere limitar de forma explícita el número de hilos de ejecución a un valor cercano al número de núcleos físicos del procesador.

2.5. Generación aumentada por recuperación

La Generación Aumentada por Recuperación (RAG, por sus siglas en inglés de *Retrieval-Augmented Generation*) es una técnica que optimiza el comportamiento de un modelo de lenguaje al complementar sus instrucciones (*prompts*) de entrada con información externa relevante sin necesidad de modificar físicamente los parámetros del modelo [13]. El flujo operativo de RAG consta de tres etapas lógicas:

1. **Recuperación:** Ante una consulta del analista, el sistema busca de manera automática los fragmentos de documentos o informes técnicos más relevantes dentro de una base de datos.
2. **Aumentación:** El sistema añade estos fragmentos recuperados al *prompt* original del usuario en forma de contexto informativo verídico.
3. **Generación:** El LLM procesa la consulta enriquecida y genera una respuesta redactada en lenguaje natural que se fundamenta estrictamente en la evidencia proporcionada, reduciendo las alucinaciones del sistema [13].

Para dar soporte y agilizar la etapa de recuperación en tiempo real, se emplean las bases de datos vectoriales (como ChromaDB o Qdrant), las cuales almacenan los textos de ciberinteligencia previamente convertidos en vectores numéricos (llamados *embeddings*) que capturan su significado semántico. De este modo, la base de datos es capaz de localizar de forma matemática los fragmentos de información más pertinentes según la intención de la consulta del analista, inyectándolos instantáneamente en el contexto de la IA [2].

Capítulo 3

Análisis, diseño e implementación del sistema

En este capítulo se explica el proceso de diseño y construcción del sistema desarrollado en este TFG. Primero se describe la arquitectura del sistema. A continuación, se detallan las tecnologías que se han utilizado en cada componente. Por último, se desarrolla el flujo que siguen los datos a través del sistema.

3.1. Arquitectura del sistema

La arquitectura de la solución propuesta se ha diseñado siguiendo un paradigma modular y desacoplado. Por restricciones corporativas, solo se dispone de una máquina virtual, por lo que todo está desplegado en la misma. No obstante, el sistema está diseñado para ser fácilmente distribuible. Esta topología, detallada en el diagrama de componentes y conectores (en notación UML) de la Figura 3.1, permite separar de forma estricta el plano de interacción con el usuario e inferencia local del plano de ingesta, normalización y ciberinteligencia de amenazas.

A nivel de infraestructura, la comunicación entre los componentes se realiza a través de puertos y *endpoints* específicos dentro de una red virtual interna, articulándose en dos bloques tecnológicos principales:

- **Dominio de interacción e inferencia local (plano izquierdo):** Este dominio gestiona la interfaz del analista y la ejecución de la IA. El componente OpenWebUI actúa como el *frontend* del sistema, comunicándose directamente con el puerto `9099` del *Pipeline Server*. Este último actúa como un *middleware* de intercepción y enrutamiento, derivando las consultas de enriquecimiento hacia el *Backend RAG* en el puerto `9000`, concretamente al *endpoint* `/query-context` y transmitiendo el *prompt* final aumentado hacia el *endpoint* `/api/chat` de *Ollama* en el puerto `11434`, el cual expone localmente los modelos *Lily-Cybersecurity-7B-v0.2* y *CyberOSS-2.0-20B*.
- **Dominio de ciberinteligencia, ingesta y vectorización (plano derecho):** Este dominio administra la adquisición de conocimiento de seguridad. El componente *OpenCTI Client* encapsula la conexión única hacia el puerto `8080` de la instancia externa de OpenCTI. De este cliente dependen de manera exclusiva

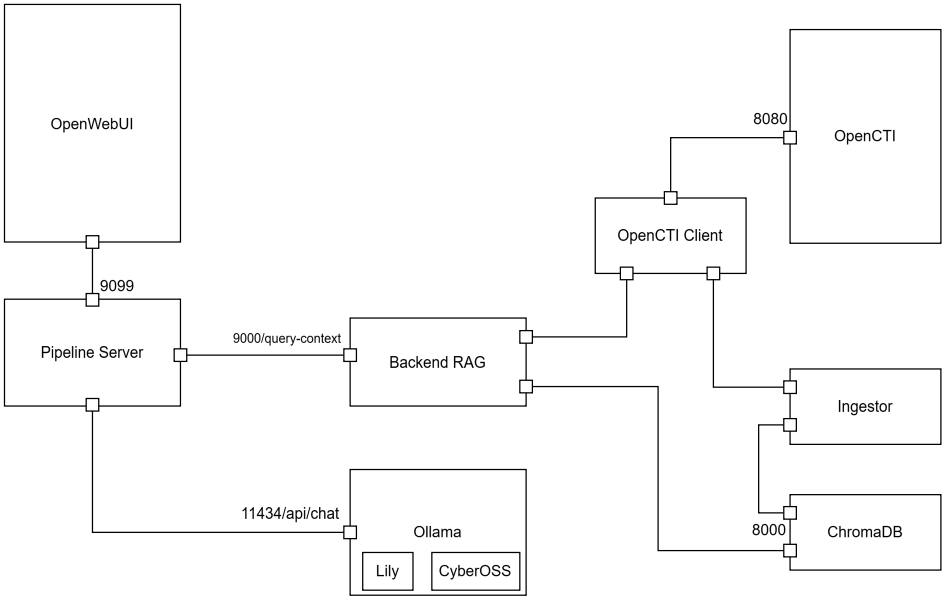

Figura 3.1: Diagrama de componentes y conectores (en notación UML)

el *Ingestor* (responsable del aprovisionamiento masivo de SDOs y SCOs) y el *Backend RAG* (responsable de las consultas semánticas y relacionales en caliente). Ambos módulos escriben y consultan, respectivamente, la base de datos vectorial ChromaDB a través de su API REST en el puerto `8000`.

3.2. Pila tecnológica

Con el propósito de estructurar de forma modular el asistente inteligente, la arquitectura del sistema propuesto se articula en torno a los siguientes componentes:

- **Motor de inferencia local y selección de modelos:** Para la ejecución de los modelos de lenguaje de gran escala bajo un entorno local, se evaluaron diversas alternativas de motores de inferencia, como *Ollama* y vLLM. Si bien vLLM ofrece un rendimiento excepcional en la gestión de solicitudes concurrentes gracias a su algoritmo de optimización de memoria [12], su arquitectura está diseñada de manera prioritaria para operar bajo aceleración por GPU. Debido a la restricción de infraestructura impuesta en este proyecto, se seleccionó *Ollama* como el motor de inferencia local principal. *Ollama*, desarrollado sobre C/C++, proporciona una capa de abstracción eficiente para entornos virtualizados, facilidad de configuración multimodelo mediante archivos de instrucción y una API local compatible con los estándares de la industria.

  Para posibilitar la interacción con el motor, los modelos de lenguaje se cargan en su versión cuantizada. En cuanto a los modelos de lenguaje seleccionados, se descartaron las opciones generalistas en favor de modelos especializados en ciberseguridad mediante *fine-tuning*. Esta especialización garantiza que la IA sea capaz de interpretar terminología técnica de ciberinteligencia, correlacionar observables y procesar de manera segura datos potencialmente ofensivos (como volcados de registros o firmas de malware) sin sufrir de bloqueos por políticas de seguridad corporativas comerciales. Con el fin de realizar una evaluación

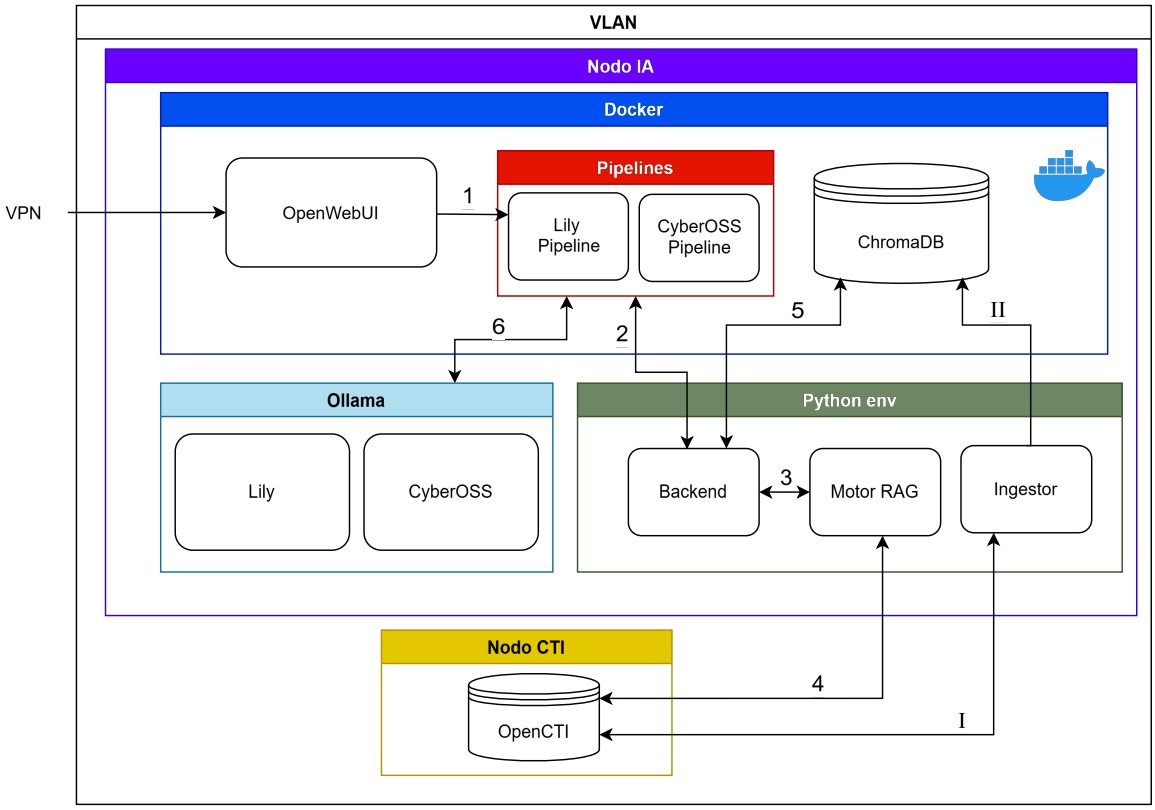

Figura 3.2: Diagrama de despliegue con comunicaciones

comparativa bajo los recursos asignados al servidor, se seleccionaron dos modelos especializados de escalas de parámetros diferenciadas:

- *Lily-Cybersecurity-7B-v0.2:* Modelo ligero desarrollado por Sego Lily Labs, ajustado sobre la arquitectura de Mistral-7B-Instruct-v0.2 a partir de un conjunto de 22.000 instrucciones de seguridad manuales [14]. En su cuantización a 4 bits, presenta un peso de archivo de aproximadamente 4,37 GB, lo que permite una inferencia ágil y fluida con un consumo mínimo de memoria RAM.
- *CyberOSS-2.0-20B (CyberPal-2.0-20B):* Modelo avanzado de la *suite* de seguridad de código abierto CyberPal 2.0, ajustado sobre la arquitectura gpt-oss-20b mediante el *pipeline* de enriquecimiento y trazado de razonamiento técnico *SecKnowledge 2.0* [10]. En su cuantización de 4 bits (Q4_K_M), el tamaño del archivo asciende a 15,8 GB, demandando un mayor esfuerzo de lectura por parte de la memoria RAM del servidor, pero aportando un análisis conceptual de nivel experto.

La elección de esta pareja de modelos permite someter la herramienta a una validación práctica frente a los *benchmarks* de referencia de la literatura científica [10].

- **Interfaz gráfica de usuario y orquestación de flujos:** Para proporcionar a los analistas de ciberseguridad un entorno visual, accesible e interactivo, se evaluaron diversas plataformas autohospedadas capaces de desplegarse mediante contenedores Docker y configurarse de manera sencilla a través de parámetros de

entorno. En una primera fase de diseño, se contempló la implementación de la interfaz ChatUI desarrollada por Hugging Face. Sin embargo, este entorno fue descartado debido a incompatibilidades severas en su integración nativa con el motor de inferencia local *Ollama*. Durante las pruebas de concepto, la ambigüedad en la documentación oficial respecto a su interconexión local propició fallos críticos en el control del estado, tales como la imposibilidad de mapear y listar dinámicamente los modelos de *Ollama*, y respuestas de los modelos no mostradas en la interfaz.

Para solucionar estos inconvenientes, se seleccionó la plataforma OpenWebUI. Esta herramienta destaca por su compatibilidad nativa e inmediata con la API de *Ollama*, configuración de parámetros tanto de forma declarativa (en la configuración de Docker) como en caliente desde su panel de administración web [15]. Un factor tecnológico determinante para su elección fue la disponibilidad de su arquitectura desacoplada de flujos de trabajo, denominada *Pipelines*. Este servidor satélite programado en Python actúa como un *middleware* capaz de interceptar, filtrar y modificar las peticiones dirigidas al modelo de lenguaje [16]. Esta capacidad proporciona el punto de anclaje necesario para inyectar lógica personalizada, realizar llamadas a servicios externos y canalizar las búsquedas de ciberinteligencia dentro de la base de datos vectorial de forma transparente para el analista, facilitando así la integración del sistema RAG.

- ***Backend*, motor de RAG y *pipelines*:** Este bloque representa el núcleo diferencial y el valor de desarrollo propio de este proyecto. Mientras que los componentes analizados hasta ahora se basan en el despliegue, parametrización y organización arquitectónica de imágenes de Docker (como OpenWebUI o la base de datos vectorial), es en esta capa donde se introduce la lógica personalizada que gobierna el flujo, tratamiento y enriquecimiento de los datos de seguridad. Aunque se describen como componentes diferenciados, el servidor de *Pipelines*, el *backend* de servicios y el motor de RAG operan como un único subsistema lógicamente acoplado debido a su constante cooperación en tiempo real.

  El flujo de procesamiento de la información se inicia cuando el analista realiza una consulta en la interfaz gráfica que menciona un IoC, momento en el cual la solicitud es interceptada de manera inmediata por el servidor de *middleware Pipelines*. En lugar de transmitir la petición de forma directa al motor de inferencia local, el *pipeline* extrae la petición y desvía la solicitud hacia un *endpoint* específico del *backend*. Al recibir este paquete, el *backend* analiza la petición y llama a las funciones del motor de RAG, el cual recupera contexto técnico sobre el indicador consultando de manera estructurada tanto la API de OpenCTI como la base de datos vectorial local (ChromaDB). Una vez obtenido este contexto histórico y relacional, se devuelve en forma de respuesta al *pipeline*, encargado de reformular el *prompt* del usuario inyectando la información recuperada y de transmitir la instrucción final enriquecida al modelo correspondiente para que este realice su razonamiento.

  Para la construcción del *backend* de servicios se seleccionó el marco de trabajo FastAPI. Esta elección se fundamenta principalmente en su sencillez y en que permite un desarrollo de *endpoints* de manera rápida y eficiente, facilitando un acoplamiento ágil con el resto de componentes escritos en Python. Asimismo,

FastAPI destaca por su capacidad para gestionar de manera óptima la asincronía de forma nativa.

- **Base de datos vectorial:** La incorporación de este componente responde a la necesidad de complementar la información relacional y estructurada obtenida de la API de OpenCTI con capacidades de recuperación semántica profunda. Mediante el uso de un modelo local de *embeddings*, el sistema procesa, fragmenta e indexa tanto los informes técnicos de amenazas como los metadatos de los observables extraídos de la plataforma de ciberinteligencia. De este modo, ante una consulta del analista, la base de datos permite ejecutar búsquedas de similitud para deducir inferencias latentes y proporcionar un contexto complementario de alto valor que no se encuentra explícitamente enlazado en el grafo de conocimiento original.

  Para la implementación de este almacén de vectores se seleccionó ChromaDB. En primer lugar, se define como una base de datos vectorial de código abierto altamente ligera y optimizada para despliegues locales (*on-premise*), lo que evita introducir una sobrecarga de infraestructura compleja o costosa en el servidor. En segundo lugar, su arquitectura modular permite que la base de datos se ejecute de forma embebida directamente dentro del proceso de Python o de manera independiente mediante un contenedor, lo que optimiza drásticamente el uso de recursos y memoria RAM en entornos limitados a CPU. Finalmente, destaca su excelente compatibilidad con el ecosistema de desarrollo de Python y su amplia adopción en la comunidad, lo que garantiza una API sencilla para operaciones de inserción, consulta, actualización y borrado, así como un soporte nativo para el filtrado avanzado de metadatos.

- **Módulo de comunicación, ingesta y normalización de ciberinteligencia:** Este componente representa la capa de software desarrollada a medida para interactuar con la plataforma OpenCTI, resolviendo la adquisición de datos de seguridad y su transformación hacia la base de datos vectorial. La conectividad lógica con el servidor se implementa bajo el patrón de diseño *Singleton* con soporte multihilo mediante un mecanismo de exclusión mutua, garantizando que exista una única instancia activa del cliente de API oficial durante la ejecución del sistema, lo que evita la saturación de conexiones en el nodo de OpenCTI.

  Sobre esta infraestructura de conexión se articulan tres subcomponentes de desarrollo propio en Python:

  - **Cliente de consultas semánticas:** Componente que se ejecuta en tiempo real dentro del flujo de la conversación. Permite realizar búsquedas exactas e intermedias sobre observables e indicadores bajo la estructura del estándar STIX 2.1, rastreando de forma automática las relaciones lógicas creadas por los analistas y localizando reportes técnicos relevantes que sirvan como evidencia contextualizada para el modelo.

  - **Ingestor de entidades:** Módulo encargado del aprovisionamiento masivo de datos históricos. Implementa el protocolo de paginación oficial basado en cursores de OpenCTI para descargar de forma secuencial y ordenada colecciones completas de actores de amenazas, malware, tácticas, reportes e indicadores técnicos, almacenando la información de ciberinteligencia en bruto localmente en formato JSON.

- **Normalizador semántico e indexador:** extrae las relaciones complejas y las unifica en plantillas textuales estructuradas y metadatos legibles para la IA. Posteriormente, un *script* de vectorización automatizado codifica estos textos normalizados en representaciones densas de 384 dimensiones empleando el modelo local `all-MiniLM-L6-v2`, insertándolos finalmente mediante un algoritmo por lotes en ChromaDB bajo una métrica de similitud de distancia del coseno para dar soporte a las búsquedas de proximidad semántica.

3.3. Flujo del sistema

En la Figura 3.2 se detalla la organización del despliegue de todos los artefactos mencionados, así como las comunicaciones que tienen entre ellos. Las comunicaciones se marcan con números para indicar el orden en el que se ejecutan. Las comunicaciones están etiquetadas de la siguiente manera: con números arábigos del 1 al 6 se indica el flujo de consulta de un analista, y con números romanos I y II el proceso de ingesta de datos de OpenCTI y su inserción en ChromaDB. Una flecha bidireccional (por ejemplo, la flecha etiquetada con 1 entre OpenWebUI y *Pipelines*) significa que se manda una solicitud y se espera una respuesta, mientras que una flecha en una sola dirección (por ejemplo, entre el *ingestor* y ChromaDB, etiquetada con II) significa que simplemente se envían datos en una sola dirección.

En este contexto empresarial, la infraestructura asignada para el despliegue de la solución consta de entornos de virtualización Linux (Red Hat 10), sin entorno de escritorio, sobre hardware de centro de datos basado únicamente en CPU y memoria RAM, prescindiendo por completo de aceleración gráfica por GPU. Además, se asigna una sola máquina al proyecto, por lo que todo se despliega en el mismo servidor, a excepción de OpenCTI, que ya se encontraba desplegado antes de iniciar el proyecto en un servidor dedicado.

Una vez definida la distribución lógica y física de la infraestructura, se define la dinámica temporal que sigue el sistema. Como se ha mencionado previamente, el sistema sigue dos flujos de datos diferentes, por una parte está la ingesta, vectorización y almacenamiento de datos de OpenCTI (módulo *Ingestor* que interactúa con OpenCTI y ChromaDB), cuyas comunicaciones están etiquetadas con números romanos en la Figura 3.2 (I y II).

3.3.1. Consumo y vectorización de OpenCTI

El cometido del módulo *Ingestor* es conectarse a OpenCTI, configurando las credenciales adecuadas. Para cada tipo de entidad CTI que se cargará (concretamente, *threat actors, malware, attack patterns, indicators y reports*), el módulo define qué atributos necesita recuperar. Para todos los tipos de objeto se cargan identificador, nombre, descripción y fecha de creación. Para *threat actors* y *reports*, estos son los únicos campos recuperados. Se recuperan campos adicionales para *malware* (se añade la familia de *malware* a la que pertenece), *attack patterns*, según el identificador MITRE [8] e *indicators* (se carga el patrón que hay que buscar para asociar el indicador a comportamiento malicioso y la sintaxis que sigue dicho patrón, por ejemplo, STIX). Así, no se descarga todo el objeto CTI, ya que esto sería un proceso muy largo y pesado, sino solo los aspectos relevantes para las fases posteriores del procesado.

Los datos se recuperan mediante ingesta paginada, consumiendo todas las páginas devueltas por OpenCTI hasta que no quedan más disponibles. Esto se debe a que OpenCTI puede tener cientos de miles o incluso millones de entidades, por lo que una única llamada a la API no sería suficiente para recuperar todos los datos o, según la configuración de *timeouts*, podría no devolver nada.

Una vez finalizada la extracción, guarda los datos obtenidos en ficheros JSON independientes. Como referencia, una ejecución de este programa carga 55 *threat actors*, 355 *malwares*, 410 *attack patterns*, 515.569 *indicators* y 5.924 *reports* (análisis realizado el 8 de junio de 2026).

El módulo `Normalizer` se encarga de la normalización de los datos brutos extraídos de OpenCTI. Transforma los objetos recuperados en documentos textuales homogéneos adecuados para su posterior vectorización. Los datos extraídos de OpenCTI se almacenan como JSON bruto, por lo que cada objeto recuperado mantiene su estructura original. Dado que los modelos de lenguaje trabajan mejor sobre texto descriptivo que sobre JSON complejos, se procesa entonces en una capa intermedia entre la ingesta bruta y la generación de *embeddings* para contener, para cada objeto CTI, la información relevante en un formato legible e igual para todos los objetos de un mismo tipo para facilitar la búsqueda por parte del LLM al vectorizar el texto. Aunque indicadores y reportes se dejan tal y como están, se incluye en esta parte la búsqueda de relaciones de ciertos tipos de objetos en OpenCTI para mejorar la semántica asociada al mismo, como relaciones de tipo `uses` en *threat actors* (para recuperar *attack patterns* usados y *malware* asociado), *malware* (para recuperar *attack patterns* usados) y *attack patterns* (para recuperar *threat actors* que usen estos patrones y técnicas).

El resultado final del proceso de normalización son documentos estructurados, uniformes para cada tipo de objeto recuperado de OpenCTI. Por razones de procesado y eficiencia, se guardan en formato `JSONL`, que almacena un objeto JSON por línea, aunque extrayendo el contenido y dejándolo en un formato humanamente legible.

Tras la fase de normalización, los objetos CTI extraídos desde OpenCTI se encuentran representados como documentos textuales semiestructurados. Aunque estos documentos ya son legibles y contienen información relevante sobre actores, malware, técnicas, indicadores e informes, todavía no pueden ser utilizados directamente para realizar búsquedas semánticas. Para ello es necesario transformarlos en *embeddings*, que permiten comparar documentos y consultas en función de su similitud semántica.

El proceso de vectorización consiste en cargar los ficheros JSON generados durante la normalización, extraer el campo relevante de cada documento y convertirlo en un vector numérico mediante el modelo `sentence-transformers/all-MiniLM-L6-v2`. Este modelo genera una representación densa del significado del texto, de forma que documentos conceptualmente similares quedan próximos dentro del espacio vectorial. En paralelo, también se cargan los ficheros de metadatos asociados a cada colección, comprobando que el número de textos y metadatos coincide antes de generar los *embeddings*. Esta comprobación garantiza que cada vector queda asociado al documento y metadatos correctos.

Los *embeddings* generados se almacenan en ficheros separados por tipo de entidad. Así, se generan ficheros independientes para *threat_actors*, *malware*, *attack_patterns*, *reports* e *indicators*. Esta separación permite mantener una organización coherente del corpus

CTI y facilita la carga posterior en colecciones diferenciadas dentro de ChromaDB. Además, guardar los *embeddings* como ficheros permite diferenciar las partes del proceso de ingesta, facilitando la ejecución del proceso en caso de reconstrucción de la base de datos.

Una vez generados los *embeddings*, el siguiente paso consiste en cargarlos en ChromaDB. Para ello, el *script* de inserción se conecta al servidor de ChromaDB. Para cada tipo de entidad, el *script* crea o recupera una colección con el mismo nombre de la entidad. Las colecciones se configuran utilizando similitud coseno.

Antes de insertar los documentos en ChromaDB, el *script* valida que los textos normalizados, los metadatos y los embeddings tengan la misma longitud. Esta validación evita desalineaciones entre documentos, vectores y metadatos. Posteriormente, la inserción se realiza por lotes, asociando a cada documento su identificador, su texto, su *embedding* y sus metadatos. De este modo, cada entrada almacenada en ChromaDB conserva tanto la representación vectorial necesaria para la búsqueda semántica como la información textual y de trazabilidad necesaria para interpretar el resultado recuperado.

El resultado final de este proceso es una base vectorial organizada en colecciones CTI, donde cada documento procedente de OpenCTI puede ser recuperado por similitud semántica. Durante una consulta, el *backend* RAG genera un *embedding* de la pregunta o del indicador detectado y consulta las colecciones de ChromaDB para obtener los documentos más cercanos. Estos documentos se incorporan al contexto final como información contextual o inferida, diferenciándola de la inteligencia explícita recuperada directamente desde OpenCTI.

La vectorización y la carga en ChromaDB permiten complementar la información confirmada de OpenCTI con contexto semántico adicional, como actores, malware, técnicas, informes o indicadores relacionados conceptualmente con la consulta. Sin embargo, la similitud vectorial no implica una relación confirmada entre entidades CTI. Por este motivo, los documentos recuperados desde ChromaDB se tratan como contexto inferido y se separan explícitamente de la información considerada confirmada.

3.3.2. *Frontend*, *backend* y motor de RAG

Una vez poblada la base de datos vectorial mediante el proceso de ingesta descrito anteriormente, el uso habitual del sistema consiste en que un analista formule una pregunta desde la interfaz web y reciba una respuesta generada por un modelo local, enriquecida con contexto de OpenCTI y ChromaDB. Este flujo de consulta se representa mediante las comunicaciones etiquetadas con números arábigos del 1 al 6 en la Figura 3.2, y se detalla en forma de diagrama de secuencia en la Figura 3.3.

Consulta desde OpenWebUI y selección de la *pipeline*

El analista accede a OpenWebUI, selecciona una de las opciones disponibles y escribe una pregunta sobre un indicador de compromiso. Desde el punto de vista del usuario, las *pipelines* aparecen listadas en la lista de modelos, aunque no son directamente los modelos de *Ollama*, de forma que el uso de las mismas es transparente.

La comunicación 1 de la Figura 3.2 corresponde al envío de la conversación de OpenWebUI al servidor de *Pipelines*. Este servidor forma parte del ecosistema de OpenWebUI

y permite añadir lógica personalizada antes de llamar al modelo. En este trabajo se han definido dos *pipelines*: `Lily RAG` y `CyberOSS RAG`. Ambas reciben la pregunta del usuario, preparan la recuperación de contexto y, finalmente, llaman al modelo local correspondiente.

Antes de continuar, la *pipeline* comprueba si la petición recibida corresponde realmente a una consulta del analista o si se trata de una tarea interna de OpenWebUI. OpenWebUI puede lanzar peticiones automáticas al modelo para generar, por ejemplo, el título del chat, etiquetas, resúmenes o sugerencias de continuación. Estas tareas no requieren consultar OpenCTI ni ChromaDB, pero consumen los recursos dedicados a la inferencia en actividades que no resultan de interés para el analista. Por ello, si la *pipeline* detecta una tarea interna, devuelve una respuesta mínima y evita ejecutar todo el flujo RAG. Este caso aparece en la Figura 3.3 como una rama alternativa. Si, por el contrario, la petición sí corresponde a una consulta sobre CTI, la *pipeline* inicia la recuperación de contexto.

Solicitud de contexto al *backend* RAG

Una vez identificada la pregunta del analista, la *pipeline* no llama todavía al modelo. Antes necesita obtener información relevante. Para ello, realiza la comunicación 2 de la Figura 3.2: envía una petición al *backend* RAG.

Este *backend* FastAPI expone el *endpoint* `/query-context`, que recibe peticiones que contienen dos datos: la pregunta del usuario y el perfil de contexto que se desea utilizar. El perfil de contexto permite ajustar cuánta información se devolverá. Esta decisión se debe a que durante las pruebas ejecutadas sobre los modelos se observó que *Lily-Cybersecurity-7B-v0.2*, al ser un modelo pequeño con una ventana de contexto menor, se saturaba al recibir contextos muy grandes, por lo que se procesa el contexto obtenido para mantener la información relevante para el análisis sin sobrecargar al modelo.

El *backend* actúa únicamente como servicio de recuperación y preparación de contexto. Es decir, no genera la respuesta final ni se comunica con los modelos de lenguaje. Su responsabilidad es reunir la información necesaria para que la *pipeline* pueda construir después un *prompt* enriquecido. Cuando el *backend* recibe la petición, valida que el cuerpo de la solicitud tenga formato JSON y que contenga una pregunta válida. Si la petición no cumple estos requisitos, devuelve un error controlado. Si la petición es correcta, comienza el proceso de recuperación mostrado en la parte central de la Figura 3.3.

Detección del indicador de compromiso

El primer trabajo del *backend* consiste en detectar qué indicador de compromiso aparece en la pregunta. Para ello, aplica expresiones regulares sobre el texto recibido. Esta aproximación se ha elegido porque muchos indicadores tienen formatos fácilmente reconocibles. Por ejemplo, una dirección IPv4, una URL, un dominio o un hash presentan estructuras sintácticas muy concretas. Usar expresiones regulares permite detectar estos casos de manera rápida, simple y controlada, sin necesidad de emplear soluciones más costosas o complejas.

El orden de detección también es relevante. Algunos indicadores pueden solaparse sintácticamente. Por ejemplo, una URL contiene un dominio, por lo que si se buscara

primero un dominio podría extraerse solo una parte de la URL e ignorar parte del indicador. Para evitar este problema, el sistema aplica una precedencia fija: primero se detectan URLs, después hashes SHA-256, SHA-1 y MD5, y finalmente direcciones IPv4 y dominios. Si no se encuentra ningún IoC válido en la pregunta, el *backend* devuelve un error indicándolo. Si se detecta un IoC, el flujo continúa con la resolución explícita en OpenCTI.

Consulta de información explícita en OpenCTI

Una vez detectado el indicador, el *backend* consulta OpenCTI para buscar información confirmada sobre él. Esta consulta corresponde a la comunicación 4 de la Figura 3.2. En la Figura 3.3, esta parte aparece como la consulta desde el *backend* RAG hacia OpenCTI y la posterior devolución de contexto explícito.

Esta información se considera la parte fuerte del contexto, ya que procede directamente de OpenCTI. El sistema busca el IoC de varias formas: en primer lugar, intenta encontrar indicadores cuyo valor coincida directamente con el indicador consultado. En segundo lugar, revisa patrones STIX que puedan contener dicho valor. Por último, recupera informes relacionados que mencionen el indicador.

El resultado de esta fase no se entrega todavía al modelo. Antes, el *backend* transforma los datos recuperados en un texto estructurado y legible seleccionando los campos más relevantes para el análisis, como el valor del indicador, el tipo, el estado, la confianza, el patrón STIX, la descripción, las fechas de validez o el origen. Además, se aplica una deduplicación de indicadores. Durante las pruebas se observó que la aparición repetida de la misma información podía confundir a modelos pequeños y reducir la calidad de sus respuestas. Por ello, los indicadores repetidos se agrupan utilizando una clave compuesta por el valor del IoC, su estado, la confianza y el patrón STIX.

Búsqueda semántica en ChromaDB

Después de recuperar la información explícita desde OpenCTI, el *backend* también busca contexto adicional en ChromaDB. Esta parte corresponde a la comunicación 5 de la Figura 3.2. En la Figura 3.3, aparece como la búsqueda semántica contra ChromaDB y la devolución de contexto inferido.

A diferencia de la consulta a OpenCTI, esta recuperación no busca únicamente coincidencias exactas. En este caso, la pregunta del usuario se transforma en un *embedding* mediante el modelo `all-MiniLM-L6-v2` de `sentence-transformers`. Este vector se compara con los documentos almacenados previamente en ChromaDB durante el proceso de ingesta.

El objetivo de esta fase es encontrar documentos semánticamente cercanos a la consulta. Estos documentos pueden aportar contexto útil, como informes, campañas, amenazas o entidades relacionadas. No obstante, esta información debe interpretarse con cautela: que un documento sea similar desde el punto de vista vectorial no significa que exista una relación confirmada con el IoC. Por este motivo, el sistema distingue explícitamente entre la información fuerte procedente de OpenCTI y la información inferida procedente de ChromaDB.

La cantidad de documentos recuperados y el nivel de detalle dependen del perfil de

contexto seleccionado. El perfil `small` limita el número de documentos y resume la información más importante, por lo que resulta adecuado para modelos con menor capacidad de contexto, como *Lily-Cybersecurity-7B-v0.2*. El perfil `large`, en cambio, incluye más información y conserva documentos más completos, por lo que está orientado a modelos capaces de procesar entradas más largas, como *CyberOSS-2.0-20B*.

Construcción del contexto final y solicitud al modelo

Cuando el *backend* ya ha obtenido la información explícita de OpenCTI y el contexto semántico de ChromaDB, se unen ambos bloques en un único contexto final. Este paso aparece en la Figura 3.3 como la fase de construcción de contexto.

El contexto final conserva la separación entre ambas fuentes. Por un lado, se incluye la información explícita recuperada directamente de OpenCTI. Por otro, se añade el contexto inferido obtenido mediante similitud vectorial en ChromaDB. Esta separación es importante porque ayuda al modelo a no presentar como cierto aquello que solo se ha recuperado por similitud semántica.

Una vez construido este bloque, el *backend* lo devuelve a la *pipeline*. Esta respuesta cierra la comunicación 2 de la Figura 3.2: la *pipeline* había solicitado contexto y el *backend* responde con el contexto consolidado.

Con el contexto ya recuperado, la *pipeline* construye el *prompt* final que se envía al modelo. En este punto comienza la fase de generación de respuesta, representada en la parte inferior de la Figura 3.3. La *pipeline* crea el *prompt* combinando el contexto recuperado con la pregunta del usuario y unas instrucciones concretas de análisis para el modelo, como la enumeración de objetos del contexto o que comente la separación entre contexto fuerte y contexto inferido.

La llamada al modelo local corresponde a la comunicación 6 de la Figura 3.2. La *pipeline* envía el *prompt* a Ollama, que ejecuta el modelo seleccionado. Existe un matiz en la implementación que es transparente al usuario pero relevante en el desarrollo del proyecto: *CyberOSS-2.0-20B* utiliza una plantilla de chat diferente al estándar que espera *Ollama*, empleando *tokens* específicos como `<|start|>user`, `<|end|>` y `<|start|>assistant` para indicar cuándo termina la pregunta del usuario y empieza la respuesta. Estos mensajes tienen que ser incluidos en el mensaje que se envía a *Ollama*, y es relevante, por ejemplo, si en un futuro se quiere añadir un nuevo modelo, dado que habría que revisar qué plantilla de chat utiliza.

Finalmente, *Ollama* ejecuta el modelo local y devuelve la respuesta generada a la *pipeline*. Después, la *pipeline* reenvía esa respuesta a OpenWebUI, que finalmente la muestra al analista. Este último tramo completa el flujo de consulta mostrado con los números 1 a 6 en la Figura 3.2.

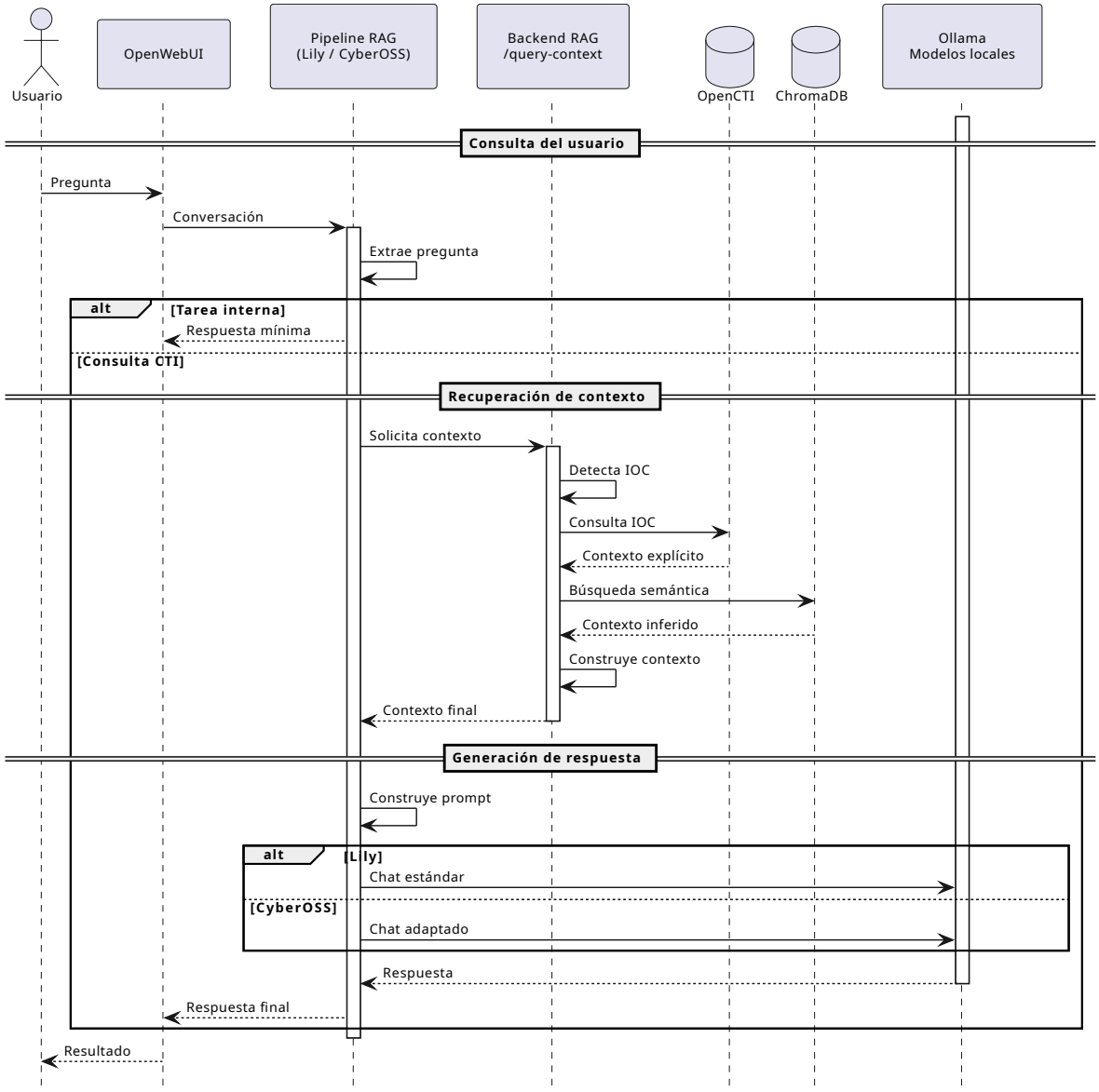

Figura 3.3: Flujo completo de consulta

Capítulo 4

Evaluación y discusión de resultados

En este capítulo se comenta la experimentación realizada para probar el sistema, comentando el entorno en el que se han desarrollado los experimentos, los experimentos realizados y los resultados obtenidos. Finalmente, se desarrolla una discusión de dichos resultados.

4.1. Entorno de experimentación

Las pruebas han sido desarrolladas dentro de la máquina virtual en la que se ha llevado a cabo el desarrollo. Cuenta con 32 GB de memoria RAM, 16 vCPUs y Red Hat Enterprise Linux 10 como sistema operativo. En dicha máquina están alojados los servicios desarrollados, con OpenWebUI en la versión 0.8.11. El *backend* está funcionando sobre Python 3.12.12, y se utilizan la versión 1.0.0 de ChromaDB y la versión 0.18.2 de Ollama. El servidor *pipelines* funciona sobre Python 3.11.13 dentro de un contenedor Docker.

4.2. Descripción de experimentos

Para evaluar la viabilidad operativa de los modelos Lily y CyberOSS[1] en el flujo de trabajo de un analista de ciberseguridad, se diseñó una batería de 7 consultas específicas basadas en indicadores reales inyectados en la base de datos vectorial mediante OpenCTI (Tabla 4.1).

El propósito de este conjunto de preguntas es medir las capacidades críticas del sistema en un entorno CTI:

- **Precisión en consultas directas (dominio, IP, hash conocidos):** Comprueban si el motor RAG recupera y procesa correctamente la información estructurada.
- **Capacidad de síntesis y abstracción (pregunta abierta):** Mide la aptitud del modelo para generar un resumen a partir del contexto.

[1]Para este capítulo se utilizarán estos nombres acortados para referirse a los modelos debido a la cantidad de menciones a los mismos.

- **Gestión de la incertidumbre (contexto adicional y separación fuerte/débil):** Evalúa si el modelo distingue de forma nativa entre un dato factual confirmado (p. ej., un indicador de compromiso directo) y una inferencia o sospecha.
- **Prevención de enlaces erróneos (relación con actores):** Reto de proximidad conceptual diseñado para comprobar si el LLM asume falsas atribuciones por el mero hecho de que un actor y un observable coexistan en el mismo documento de contexto.

Tipo de pregunta	**Pregunta**
Dominio conocido	*Dame detalles sobre resisterma.com.br*
IP conocida	*Dame detalles sobre 85.209.195.160*
Hash conocido	*Dame detalles sobre 684634f51cb07d40e58c88b51c62ff1c83a18b69*
Pregunta abierta	*Resume la información disponible acerca de 62.211.188.248*
Contexto adicional	*¿Qué contexto adicional CTI existe para officeonlaine.com?*
Relación con actores	*¿Está 62.211.188.248 relacionada con algún actor de amenazas?*
Separación fuerte/débil	*¿Qué información acerca de officeonlaine.com está confirmada en OpenCTI y cuál es solo inferida?*

Tabla 4.1: Preguntas utilizadas en la evaluación

Cada una de las respuestas se puntuó utilizando una rúbrica (Tabla 4.2), con puntuaciones comprendidas entre 0 y 2 para seis criterios técnicos distintos, lo que resulta en un máximo de 12 puntos por pregunta. Para asignar valores, se observa el contexto que el *backend* devuelve para cada consulta para ver qué datos aparecen mencionados en cada parte del contexto, y después se comparan esos datos con la respuesta obtenida.

Criterio	**0**	**1**	**2**
¿Menciona indicadores explícitos?	No	Parcialmente	Completamente
¿Muestra bien el estado y la confianza?	No	Incompleto	Correcto
¿Menciona y separa el contexto inferido?	Ignora	Menciona pero mal	Menciona y separa correctamente
¿Separa relaciones fuertes y débiles?	Mezcla/confunde	Separa vagamente	Separa correctamente
¿Alucina?	Sí, gravemente	Sí, levemente	No
¿Es útil para análisis?	Genérico	Superficial	Útil para analista

Tabla 4.2: Rúbrica de evaluación utilizada

4.3. Discusión de resultados

A nivel cuantitativo, los tiempos de inferencia se resumen en la Tabla 4.3 (expresados en formato *mm:ss,cs*). Como era de esperar, el modelo Lily resulta mucho más rápido, con una media de 94,5 segundos por respuesta, frente a los 187,2 segundos de CyberOSS. Al

ser el archivo de CyberOSS más de tres veces más pesado que el de Lily (concretamente una relación aproximada de $15,8$ GB$/4,37$ GB $\approx 3,6\times$), la transferencia de datos ralentiza la velocidad de generación, un coste que el analista debe valorar a cambio de una mayor capacidad de análisis. Este resultado responde a un hecho físico evidente en entornos *on-premise*: al no contar con aceleración por GPU, el factor limitante es la velocidad de la memoria RAM para leer los pesos del modelo.

Sin embargo, al observar las desviaciones típicas registradas en la Tabla 4.3 (31,2 segundos para Lily y 57,6 segundos para CyberOSS), se constata que la dispersión de los tiempos de respuesta guarda una relación directa con la longitud de los textos generados. Frente a las medias de 94,5 y 187,2 segundos, estas desviaciones representan un coeficiente de variación de alrededor del 33 % para Lily y del 30 % para CyberOSS. Estos porcentajes evidencian que el comportamiento de la inferencia local sobre CPU no es plano ni totalmente inmóvil, sino que varía de forma notable según la complejidad de la consulta y el volumen de *tokens* de salida. En el caso de CyberOSS, se entiende también una inversión de tiempo extra, dado que tiende a dar las respuestas largas formateadas.

Pregunta	**Lily**	**CyberOSS**
Dominio conocido	01:49,2	03:17,3
IP conocida	01:16,6	02:20,5
Hash conocido	01:29,7	02:45,0
Abierta	02:05,5	04:21,4
Contexto inferido	01:43,8	02:37,0
Separación fuerte/débil explícita	00:35,3	02:01,6
Relación con actor	02:01,5	04:27,4
Mejor tiempo	00:35,3	02:01,6
Peor tiempo	02:05,5	04:27,4
Rango	01:30,2	02:25,7
Media	01:34,5	03:07,2
Mediana	01:43,8	02:45,0
Desviación típica	00:31,2	00:57,6

Tabla 4.3: Comparativa de tiempos de respuesta (en formato *mm:ss,cs*)

El análisis cualitativo de las respuestas permite estudiar el comportamiento de ambos modelos frente a diferentes estructuras de consulta. A continuación, se desglosan los resultados obtenidos en función de la naturaleza de los indicadores y de los retos interpretativos planteados en cada escenario de prueba.

En las consultas directas sobre datos fijos, reflejadas en las tablas "Resultados para dominio conocido"(Tabla 4.4), "Resultados para IP conocida"(Tabla 4.5) y "Resultados para hash conocido"(Tabla 4.6), se aprecia una clara brecha técnica. CyberOSS demuestra regularidad gracias a su mayor ventana y arquitectura, reflejando en sus respuestas los grafos estructurados de OpenCTI con solvencia. Por contra, Lily sufre para procesar de manera estricta los elementos limítrofes, recurriendo en ocasiones a generalidades de ciberseguridad en lugar de ceñirse exclusivamente a los datos, lo que penaliza sus puntuaciones en las tablas de dominio (5/12), IP (7/12) y hash (8/12).

Criterio	Lily	CyberOSS
¿Menciona indicadores explícitos?	1	2
¿Muestra bien el estado y confianza?	1	2
Menciona y separa el contexto inferido	0	2
¿Separa relaciones fuertes y débiles?	1	2
¿Alucina?	1	1
¿Es útil para análisis?	1	2
Puntuación	**5**	**11**

Tabla 4.4: Resultados para dominio conocido

Criterio	Lily	CyberOSS
¿Menciona indicadores explícitos?	1	2
¿Muestra bien el estado y confianza?	1	2
Menciona y separa el contexto inferido	2	1
¿Separa relaciones fuertes y débiles?	1	1
¿Alucina?	1	2
¿Es útil para análisis?	1	1
Puntuación	**7**	**9**

Tabla 4.5: Resultados para dirección IP conocida

Criterio	Lily	CyberOSS
¿Menciona indicadores explícitos?	2	2
¿Muestra bien el estado y confianza?	2	2
Menciona y separa el contexto inferido	0	2
¿Separa relaciones fuertes y débiles?	1	2
¿Alucina?	2	1
¿Es útil para análisis?	1	2
Puntuación	**8**	**11**

Tabla 4.6: Resultados para hash conocido

Se observa al contrastar los "Resultados para IP conocida"(Tabla 4.5) con la "Pregunta abierta"(Tabla 4.7), ambas formuladas sobre el mismo indicador, que a pesar de contar con idéntico contexto inyectado, Lily eleva de forma drástica su desempeño, subiendo de los 7 hasta los 11 puntos (alcanzando la misma nota que CyberOSS).

Esto demuestra de forma empírica que los modelos de menor escala son altamente sensibles a la estructura de la instrucción: cuando se les guía explícitamente hacia una tarea acotada de síntesis ("Resume la información disponible acerca de..."), logran canalizar mejor la atención, separando las entidades reales del ruido ambiental con la misma eficacia que un modelo de mayor envergadura.

Criterio	Lily	CyberOSS
¿Menciona indicadores explícitos?	2	2
¿Muestra bien el estado y confianza?	2	2
Menciona y separa el contexto inferido	2	2
¿Separa relaciones fuertes y débiles?	2	2
¿Alucina?	2	1
¿Es útil para análisis?	2	2
Puntuación	**11**	**11**

Tabla 4.7: Resultados para pregunta abierta

Esta dependencia del guiado se constata por los resultados en la prueba de "Relación con actores"(Tabla 4.8), el escenario con mayor disparidad de notas (2/12 frente a 12/12). Al enfrentarse a una consulta compleja que requiere mucho análisis de contexto, Lily comete un error analítico: asume de forma automática un vínculo directo de atribución entre el observable analizado y un actor de amenazas, solo por compartir cercanía semántica dentro de los bloques de texto recuperados de la base de datos vectorial. Esto además propicia que ignore partes clave del contexto, como estado y confianza. CyberOSS, en cambio, demuestra mayor robustez, interpretando con precisión las separaciones de contexto para expresar explícitamente que no existe una prueba confirmada de vinculación directa, evitando la alucinación.

Criterio	Lily	CyberOSS
¿Menciona indicadores explícitos?	1	2
¿Muestra bien el estado y confianza?	0	2
Menciona y separa el contexto inferido	1	2
¿Separa relaciones fuertes y débiles?	0	2
¿Alucina?	0	2
¿Es útil para análisis?	0	2
Puntuación	**2**	**12**

Tabla 4.8: Resultados de relación con actores

Criterio	Lily	CyberOSS
¿Menciona indicadores explícitos?	2	2
¿Muestra bien el estado y confianza?	1	2
Menciona y separa el contexto inferido	2	2
¿Separa relaciones fuertes y débiles?	2	1
¿Alucina?	2	1
¿Es útil para análisis?	2	2
Puntuación	**12**	**10**

Tabla 4.9: Resultados de separación fuerte/débil explícita

Criterio	Lily	CyberOSS
¿Menciona indicadores explícitos?	1	2
¿Muestra bien el estado y confianza?	0	2
Menciona y separa el contexto inferido	1	2
¿Separa relaciones fuertes y débiles?	0	1
¿Alucina?	1	1
¿Es útil para análisis?	1	2
Puntuación	**4**	**10**

Tabla 4.10: Resultados para pregunta por contexto inferido

Por último, el impacto de la densidad conceptual y el volumen del contexto sobre el mecanismo de atención de los modelos se manifiesta de forma opuesta en la "Separación fuerte/débil explícita" (Tabla 4.9) y en la pregunta por "Contexto inferido" (Tabla 4.10).

En la Tabla 4.9, cuando la directriz de la consulta exige una tarea de discriminación muy directa y focalizada (*"¿qué está confirmado y qué es inferido?"*), el modelo ligero Lily alcanza su máximo rendimiento analítico con una puntuación perfecta (12/12), superando los 10/12 de CyberOSS, el cual penaliza levemente en el criterio de alucinación y separación de relaciones. Esto demuestra que, bajo un direccionamiento estricto y sin ruido ambiental masivo, la ventana de atención de un modelo menor es plenamente eficaz para la categorización CTI.

Sin embargo, esta situación se invierte drásticamente al pasar a la consulta por "Contexto inferido" (Tabla 4.10). En este escenario, el bajo rendimiento de Lily (4/12) no se debe a una saturación por volumen masivo, sino a la incapacidad del modelo ligero para procesar e interpretar metadatos técnicos estructurados (como los estados *Revoked*, los niveles de confianza o las referencias MISP) presentes en el contexto de OpenCTI. Ante la falta de madurez para desgranar estos atributos, Lily asume erróneamente una relación operacional directa, afirmando que actores con similitud semántica tienen relación con el indicador, sin mencionar que son inferidos y recurre a una conclusión vaga basada en recomendaciones genéricas de ciberseguridad. En contraposición, CyberOSS demuestra resiliencia (10/12) al discriminar con mayor rigor el alcance de la búsqueda por similitud, evitando lagunas de contenido y manteniendo una precisión analítica notablemente superior.

Como síntesis de este bloque cualitativo, la Tabla 4.11 consolida de forma numérica el rendimiento global observado a lo largo de las siete pruebas experimentales. Las calificaciones finales reflejan una brecha analítica directa provocada por la escala de la arquitectura. El modelo ligero Lily cierra la evaluación con un promedio global de 6,86 sobre 12 puntos posibles, arrastrado por sus dificultades puntuales en escenarios de inferencia o metadatos técnicos complejos, lo que equivale a una calificación corregida de 5,71 sobre 10. Por el contrario, la mayor densidad y entrenamiento previo de CyberOSS se traducen en una consistencia a nivel de análisis técnico superior, promediando un 10,57 sobre 12 (un 8,81 en escala sobre 10), confirmándose como una herramienta cualitativamente más fiable para el entorno operacional. Cabe destacar que los resultados obtenidos se alinean con los que se muestran en el estudio de [10].

	Lily	**CyberOSS**
Promedio	6,86	10,57
Sobre 10	5,71	8,81

Tabla 4.11: Resultados finales promediados

La evaluación comparativa revela un claro balance técnico entre coste de cómputo y calidad analítica: la gran fortaleza de Lily radica en su velocidad y agilidad bajo recursos estrictos de CPU, pero su debilidad es la tendencia a alucinar relaciones por proximidad semántica ante consultas abstractas o complejas; por el contrario, la fortaleza de CyberOSS es su madurez cognitiva, razonamiento lógico y resiliencia frente a la ambigüedad conceptual, siendo su única debilidad el alto tiempo de espera en inferencia debido al volumen de su arquitectura.

Desde una perspectiva operativa, conviene delegar en Lily tareas que requieran poco análisis, como la clasificación preliminar de alertas, el filtrado automatizado de observables y la generación de resúmenes bajo directrices muy específicas, mientras que el uso de CyberOSS debe reservarse para el análisis forense profundo, la resolución de grafos de atribución de amenazas complejas y la toma de decisiones estratégicas donde la precisión técnica prime sobre la inmediatez de la respuesta.

Capítulo 5

Conclusiones y trabajo futuro

El desarrollo de este proyecto ha permitido diseñar e implementar una arquitectura de asistencia conversacional privada y *on-premise* orientada al análisis de Indicadores de Compromiso. A través de la integración de OpenCTI, OpenWebUI y ChromaDB, se ha demostrado la viabilidad técnica de construir entornos de Inteligencia Artificial que garantizan la soberanía de los datos corporativos sensibles sin depender de servicios en la nube.

Desde la perspectiva del diseño, el flujo de trabajo modular propuesto ha probado su eficiencia al resolver el desacoplamiento de datos mediante un *backend* RAG integrado. En esta arquitectura, la base de datos vectorial ChromaDB actúa como un componente de apoyo complementario que trabaja de forma conjunta con la API de OpenCTI; mientras esta última suministra la ciberinteligencia relacional y estructurada, el almacén vectorial asiste al sistema recuperando contexto analítico adyacente bajo demanda. Esta cooperación mitiga las limitaciones de ventana de contexto de los modelos locales sin saturar estáticamente los *prompts*, confirmando la viabilidad de desplegar una plataforma de ciberinteligencia cohesiva utilizando herramientas de código abierto basadas en el estándar STIX.

En el plano cualitativo, las pruebas empíricas evidencian que el rendimiento del sistema está fuertemente ligado tanto a la escala de parámetros del modelo como a la precisión de las instrucciones suministradas. Mientras que *Lily-Cybersecurity-7B-v0.2* ofrece una agilidad de respuesta óptima para entornos limitados a CPU gracias a su ligereza, presenta debilidades al procesar consultas abstractas. Por el contrario, *CyberOSS-2.0-20B* demuestra una madurez cognitiva superior al discriminar el ruido informativo y contextualizar la incertidumbre, asumiendo un mayor coste de tiempo debido a la inferencia puramente basada en CPU. Asimismo, los resultados confirman que los modelos de menor escala incrementan notablemente su precisión cuando se les guía mediante tareas y comandos concretos frente a peticiones generales. A nivel cuantitativo, los coeficientes de variación de alrededor del 30 % para ambos modelos denotan que existe una variabilidad a tener en cuenta respecto a la media, dependiendo del tamaño del contexto inyectado al modelo y la complejidad de la tarea encomendada.

A partir de las conclusiones extraídas, se definen las siguientes líneas estratégicas de trabajo futuro para la evolución de la plataforma. En primer lugar, se plantea optimizar la búsqueda semántica en ChromaDB y moderar el contexto entregado a los modelos,

desarrollando una capa intermedia que filtre, reorganice la información recuperada, e incorpore técnicas de compresión de contexto, algoritmos de reclasificación y filtros condicionados por tipo de observable. En segundo lugar, se contempla refinar la lógica de recuperación expandiendo el sistema hacia un asistente CTI general que explote de forma nativa las relaciones de OpenCTI. Ello permitirá estructurar grafos relacionales profundos para formular análisis de campañas o perfiles de actores de amenazas basados en conocimiento formal.

Finalmente, en el plano del despliegue, se pretende avanzar hacia una infraestructura contenerizada completa mediante Docker, incorporando mecanismos automatizados para actualizar la base vectorial de forma incremental sin necesidad de regenerar el índice completo por lotes. Asimismo, se contempla añadir herramientas de registro estructurado para registrar la trazabilidad de los *prompts* e inyectar un sistema heurístico de citas en la respuesta final del asistente, forzando a los modelos a indicar si una afirmación procede de la ciberinteligencia factual de OpenCTI o del almacenamiento semántico de ChromaDB.

Bibliografía

[1] K. Chen, X. Zhou, Y. Lin, S. Feng, L. Shen y P. Wu, «A survey on privacy risks and protection in large language models», *Journal of King Saud University Computer and Information Sciences*, vol. 37, n.º 7, pág. 163, sep. de 2025, ISSN: 1319-1578, 2213-1248.

[2] D. Jurafsky y J. H. Martin, *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition, with Language Models*, 3rd. 2026.

[3] P. Santos, R. Abreu, M. J. C. S. Reis, C. Serôdio y F. Branco, «A Systematic Review of Cyber Threat Intelligence: The Effectiveness of Technologies, Strategies, and Collaborations in Combating Modern Threats», *Sensors*, vol. 25, n.º 14, pág. 4272, 9 de jul. de 2025, ISSN: 1424-8220.

[4] OpenCTI. «OpenCTI Documentation», visitado 17 de jun. de 2026. dirección: https://docs.opencti.io/latest/

[5] OASIS Open, *Structured threat information expression (STIX) version 2.1*, jun. de 2021. dirección: https://docs.oasis-open.org/cti/stix/v2.1/os/stix-v2.1-os.html

[6] D. Bianco. «Enterprise Detection & Response: The Pyramid of Pain», Enterprise Detection & Response, visitado 27 de mayo de 2026. dirección: https://detect-respond.blogspot.com/2013/03/the-pyramid-of-pain.html

[7] J. Castro, *Rethinking the Pyramid of Pain in 2025: Precision, Pressure, and the Power of Context*, ResearchGate, jul. de 2025. visitado 20 de jun. de 2026. dirección: https://www.researchgate.net/publication/394107570_Rethinking_the_Pyramid_of_Pain_in_2025_Precision_Pressure_and_the_Power_of_Context

[8] MITRE. «MITRE ATT&CK», visitado 8 de jun. de 2026. dirección: https://attack.mitre.org/

[9] OpenCTI. «OpenCTI Data Model», OpenCTI Data Model, visitado 27 de mayo de 2026. dirección: https://docs.opencti.io/latest/usage/data-model/

[10] M. Levi, D. Ohayon, A. Blobstein, R. Sagi, I. Molloy e Y. Allouche, *Toward Cybersecurity-Expert Small Language Models*, 2025. visitado 19 de jun. de 2026. dirección: https://arxiv.org/abs/2510.14113

[11] J. L. Hennessy y D. A. Patterson, *Computer Architecture: A Quantitative Approach*, 6.ª ed. Morgan Kaufmann, 2017, ISBN: 978-0-12-811905-1.

[12] W. Kwon, Z. Li, S. Zhuang, Y. Sheng, L. Zheng, C. H. Yu, J. Gonzalez, H. Zhang e I. Stoica, «Efficient Memory Management for Large Language Model Serving with PagedAttention», en *Proceedings of the 29th Symposium on Operating Systems Principles*, Koblenz Germany: ACM, 23 de oct. de 2023, págs. 611-626, ISBN: 979-8-4007-0229-7.

[13] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W.-t. Yih, T. Rocktäschel, S. Riedel y D. Kiela, *Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks*, 2021. visitado 19 de jun. de 2026. dirección: https://arxiv.org/abs/2005.11401

[14] «Segolilylabs/Lily-Cybersecurity-7B-v0.2 · Hugging Face», visitado 29 de mayo de 2026. dirección: https://huggingface.co/segolilylabs/Lily-Cybersecurity-7B-v0.2

[15] OpenWebUI. «Open WebUI», visitado 1 de jun. de 2026. dirección: https://docs.openwebui.com/

[16] OpenWebUI. «Pipelines / Open WebUI», visitado 1 de jun. de 2026. dirección: https://openwebui.com/features/extensibility/pipelines/

Anexos

Anexo A

Organización del trabajo

En este anexo se recogen el desglose de tareas planificadas, reflejadas en el diagrama de Gantt (Figura A.1) y las horas dedicadas a dichas tareas, recogidas en la Tabla A.1.

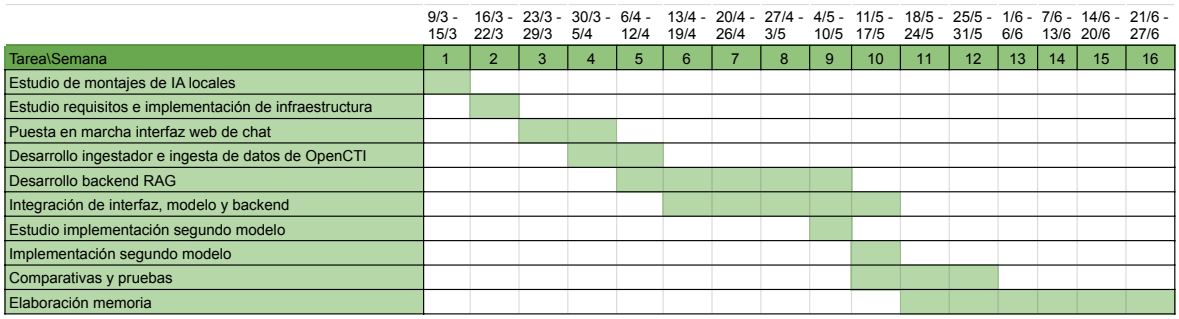

Figura A.1: Diagrama de Gantt

Tarea	**Horas**
Estudio de montajes de IA locales	10
Estudio requisitos e implementación de infraestructura	10
Puesta en marcha interfaz web de chat	20
Desarrollo ingestador e ingesta de datos	25
Desarrollo *backend* y motor RAG	55
Integración interfaz, modelo y backend	70
Estudio de segundo modelo	5
Implementación segundo modelo	5
Comparativas y pruebas	25
Desarrollo memoria	80
Total	**305**

Tabla A.1: Tabla de tareas y tiempos