

**Desarrollo de un sistema automatizado de asistencia vía WhatsApp mediante
Inteligencia artificial generativa local para la optimización del soporte técnico en
UNIDROGAS S.A.S.**

Juan Pablo Acuña Martínez

Trabajo de grado para optar al título de Ingeniero Mecatrónico

Director

Carlos Daniel Pimentel Díaz

Magíster en inteligencia artificial

Universidad Santo Tomás, Bucaramanga

División de Ingenierías y Arquitectura

Ingeniería Mecatrónica

2026

Contenido

Introducción.....	10
1.1 Planteamiento del problema.....	11
1.2 Justificación.....	12
1.3 Objetivos.....	13
1.3.1 Objetivo general.....	13
1.3.2 Objetivos específicos.....	13
2. Marco referencial.....	14
2.1 Marco teórico.....	14
2.2 Marco conceptual.....	17
2.3 Marco legal.....	18
3. Método.....	19
3.1 Configuración de la infraestructura y selección del modelo base.....	19
3.2 Generación de <i>embeddings</i> e indexación de documentos en motor de base de datos vectorial.....	27
3.3 Integración del modelo de lenguaje y WhatsApp.....	33
3.4 Alcance y limitaciones.....	37
4. Resultados.....	38
4.1 Resultados de recuperación semántica e identificación de requerimientos.....	41
4.2 Análisis de errores y matriz de confusión.....	41
4.3 Limitaciones y oportunidades de mejora.....	42
4.4 Resultados de integración con WhatsApp.....	42
5. Conclusiones.....	45

6. Recomendaciones.....	47
Referencias.....	48

Lista de tablas

Tabla 1. <i>Especificaciones técnicas de la infraestructura de hardware del servidor local</i>	18
Tabla 2. <i>Métricas de rendimiento y consumo según el nivel de cuantización del modelo</i>	20
Tabla 3. <i>Matriz de evaluación y selección preliminar de modelos de lenguaje</i>	22
Tabla 4. <i>Estructura de respuesta de la API de ChromaDB tras la recuperación semántica</i>	32
Tabla 5. <i>Métricas de rendimiento de la búsqueda semántica por categoría</i>	38

Lista de figuras

Figura 1. <i>Flujo de trabajo y arquitectura general de un sistema de Generación Aumentada por Recuperación (RAG).....</i>	13
Figura 2. <i>Resultados de la prueba de inferencia comparativa en la interfaz de Promptfoo.....</i>	24
Figura 3. <i>Comparativa del promedio de velocidad de inferencia entre los modelos finalistas... </i>	24
Figura 4. <i>Comparativa del promedio de latencia de respuesta entre los modelos finalistas.....</i>	25
Figura 5. <i>Estructura del archivo JSON para la normalización de la información.....</i>	28
Figura 6. <i>Diagrama de flujo del programa de validación e ingesta de datos.....</i>	29
Figura 7. <i>Resultados de las pruebas de consulta y recuperación de información semántica desde la API de ChromaDB.....</i>	30
Figura 8. <i>Diagrama de arquitectura y flujo de procesamiento para la generación de respuestas</i>	33
Figura 9. <i>Diagrama de estados del flujo conversacional para la gestión de incidencias.....</i>	34
Figura 10. <i>Matriz de confusión de la tasa de aciertos en la identificación de requerimientos....</i>	36
Figura 11. <i>Capturas de pantalla de conversación en WhatsApp.....</i>	39

Resumen

En este documento se aborda el problema de saturación del soporte técnico en UNIDROGAS S.A.S., con el objetivo de implementar un asistente virtual conversacional. Para su desarrollo, se integró el modelo de lenguaje local Gemma 3, una arquitectura RAG alimentada con manuales de la empresa y la API de WhatsApp. El sistema alcanzó una exactitud del 93% en la identificación del documento o procedimiento correspondiente a la consulta del usuario, lo que evidencia la viabilidad de automatizar procesos mediante inteligencia artificial y optimizar los tiempos de respuesta. Asimismo, su arquitectura permite resguardar la privacidad en la gestión interna del conocimiento institucional, si bien el canal de comunicación final depende de un proveedor externo.

Palabras clave: Inteligencia artificial, RAG, LLM, búsqueda semántica

Abstract

This document addresses the issue of technical support overload at UNIDROGAS S.A.S., with the goal of implementing a conversational virtual assistant. To develop the system, the local language model Gemma 3 was integrated with a RAG architecture indexed over the company's manuals and the WhatsApp API. The system achieved a 93% accuracy rate in identifying the document or procedure corresponding to the user's query, demonstrating the feasibility of automating processes using artificial intelligence and optimizing response times. Furthermore, its architecture ensures privacy in the internal management of institutional knowledge, although the final communication channel relies on an external provider.

Keywords: Artificial intelligence, RAG, LLM, semantic search

Glosario

API (Application Programming Interface): Mecanismo de integración basado en protocolos estandarizados que permite la conexión y el intercambio ordenado de datos entre distintas plataformas de software [1].

Base de datos vectorial: Repositorios de información diseñados específicamente para alojar, organizar y consultar vectores de alta dimensionalidad. Su principal ventaja frente a las bases de datos relacionales convencionales es el uso de métodos de búsqueda por aproximación, lo que les permite encontrar y devolver los registros que mejor coinciden con la intención semántica del usuario [2].

Chunking: Procedimiento propio del procesamiento de lenguaje natural que consiste en dividir archivos de texto extensos en bloques más manejables, manteniendo su sentido original [3].

Cuantización: Método de compresión para modelos de inteligencia artificial que recorta la precisión matemática de sus componentes internos (reduciendo formatos pesados de 16 o 32 bits a arquitecturas más ligeras de 8 o 4 bits). Esta adaptación disminuye drásticamente el consumo de memoria gráfica necesaria para ejecutar el sistema, conservando un alto nivel de coherencia en la generación de texto [4].

Inferencia: Fase de ejecución computacional en la que un modelo de inteligencia artificial (como un LLM) procesa la entrada del usuario y el contexto recuperado para predecir y generar una respuesta [3].

LLM (Large Language Model): Sistema de inteligencia artificial entrenado para procesar, comprender y generar texto natural [4].

On-premise: Estrategia de alojamiento tecnológico en la que todo el sistema opera directamente dentro del hardware y las redes internas de una empresa [5].

Prompt: Instrucción, consulta o texto de entrada inicial proporcionado por el usuario para dirigir y condicionar la respuesta de un LLM [3].

Similitud de coseno: Ecuación matemática empleada para medir la cercanía temática de dos vectores basándose exclusivamente en su alineación direccional (el ángulo entre ellos), ignorando por completo su magnitud escalar. Es eficaz para sistemas de recuperación de información, ya que permite relacionar de manera precisa consultas muy cortas de los usuarios con fragmentos documentales extensos [6].

Token: Unidad fundamental de procesamiento (que puede representar una palabra, porción de una palabra o carácter) en la que se fragmenta el texto de entrada y de salida, permitiendo que el modelo de lenguaje pueda cuantificar y procesar matemáticamente la información durante las etapas de indexación y generación [3].

Introducción

El presente trabajo se desarrolla en el área de soporte técnico de UNIDROGAS S.A.S., un departamento que registra un alto volumen de consultas repetitivas de baja complejidad a través de sus canales de comunicación a nivel nacional. Frente a esta condición operativa, el proyecto documenta el desarrollo y la validación de un asistente virtual conversacional automatizado, accesible a través de la interfaz de WhatsApp.

La solución técnica construida se basa en la integración de un Modelo de Lenguaje Grande (LLM) de código abierto operando en conjunto con una arquitectura de Generación Aumentada por Recuperación (RAG). Para dar cumplimiento a los lineamientos de manejo de datos de la empresa, el procesamiento, la vectorización de los manuales corporativos y la inferencia se ejecutan en un despliegue local (on-premise) alojado en los servidores físicos de la organización. Esta arquitectura resguarda la privacidad en la gestión interna del conocimiento institucional, si bien el canal de comunicación final depende de un proveedor externo.

Para establecer la viabilidad técnica de esta arquitectura, el informe detalla el proceso metodológico llevado a cabo para su implementación. El documento abarca el diagnóstico de la infraestructura de hardware, la selección del modelo base según las restricciones de memoria, la preparación de la base de conocimiento vectorial y la orquestación del flujo de comunicación con la plataforma de mensajería. Asimismo, se reporta la evaluación cuantitativa del desempeño del sistema mediante una prueba controlada enfocada en la etapa de recuperación semántica. A partir del análisis de los resultados y la identificación del solapamiento de categorías, se exponen las limitaciones operativas del prototipo y se estructuran las recomendaciones requeridas para escalar la herramienta hacia un entorno de producción masiva.

1. Desarrollo de un sistema automatizado de asistencia vía WhatsApp mediante Inteligencia artificial generativa local para la optimización del soporte técnico en UNIDROGAS S.A.S.

1.1 Planteamiento del problema

En el entorno operativo de UNIDROGAS S.A.S., el área encargada de la gestión de soporte técnico y operativo se enfrenta actualmente a una saturación crítica en sus canales de comunicación. Esta problemática es provocada por un alto flujo de consultas repetitivas y de baja complejidad, provenientes tanto de los empleados en las oficinas centrales como del personal ubicado en las distintas sucursales a nivel nacional.

A pesar de que la organización cuenta con manuales de procesos y documentación técnica preexistente para la resolución de estas incidencias comunes, el acceso a esta información no es inmediato ni intuitivo para el usuario final. En consecuencia, el personal de soporte se ve en la obligación de intervenir manualmente en una gran cantidad de tareas rutinarias, lo que genera cuellos de botella que retrasan la atención de problemáticas críticas y aumentan los tiempos de respuesta a nivel general. Adicionalmente, cualquier iniciativa para automatizar este flujo de información se encuentra limitada por políticas de la empresa de mantener la confidencialidad de los datos institucionales, lo que descarta el uso de herramientas de terceros alojadas en nubes públicas.

Frente a esta situación que afecta la eficiencia de la cadena de valor de la empresa, se formula la siguiente pregunta de investigación:

¿De qué manera el desarrollo de un sistema automatizado de asistencia vía WhatsApp, impulsado por Inteligencia Artificial Generativa y arquitectura RAG en un entorno local permite optimizar los tiempos de respuesta del área de soporte técnico en UNIDROGAS S.A.S.?

1.2 Justificación

El desarrollo de este sistema de asistencia automatizada surge de la necesidad de evolucionar la gestión de soporte técnico de UNIDROGAS S.A.S. hacia una herramienta de interacción inteligente y dinámica. Actualmente, la recuperación de la documentación técnica demanda tiempo y esfuerzo. Mediante la integración de modelos de lenguaje grande (LLM) de código abierto con una arquitectura de Generación Aumentada por Recuperación (RAG), se supera la limitación de las búsquedas tradicionales por palabras clave y de la intervención manual. Esta integración permite una recuperación semántica estructurada de la información institucional mediante la segmentación y vectorización de los datos (*embeddings*), permitiendo que el sistema comprenda la intención del usuario y proporcione el contexto necesario para resolver incidencias.

Desde el punto de vista de recursos humanos, el proyecto tiene el potencial para incrementar la productividad dentro de la organización. El propósito de esta automatización es optimizar el uso del tiempo de los colaboradores del área de sistemas. Al delegar la atención de solicitudes rutinarias a la inteligencia artificial, el personal técnico puede redirigir sus capacidades a mantenimiento de infraestructura crítica y resolución de fallas complejas.

En el campo científico y tecnológico, la investigación busca aportar evidencia al evaluar la viabilidad técnica de ejecutar inteligencia artificial generativa en un entorno *on-premise*. El desarrollo evidencia cómo, mediante el uso de modelos cuantizados, es factible mantener un desempeño computacional estable sin depender de infraestructuras en la nube. Este enfoque

permite fortalecer la protección y confidencialidad de la información institucional, sirviendo como un modelo de referencia sobre la adopción segura y financieramente sustentable de herramientas de IA en organizaciones.

Finalmente, los campos de aplicación de esta solución trascienden el área de sistemas. La arquitectura diseñada sienta las bases para un ecosistema tecnológico escalable que puede ser replicado e integrado en otras dependencias definidas en la estructura orgánica de la empresa. Áreas como Talento Humano, Administración de Depósito o la Dirección Financiera poseen el potencial de adoptar este mismo modelo para gestionar sus propios repositorios documentales y automatizar la asistencia interna.

1.3 Objetivos

1.3.1 Objetivo general

Analizar la viabilidad y el desempeño de un sistema de asistencia automatizada para incidencias de baja complejidad en UNIDROGAS S.A.S., implementando inteligencia artificial generativa (arquitectura RAG) en un entorno local integrado a WhatsApp, con el fin de optimizar la eficiencia operativa y resguardar la privacidad de la información institucional.

1.3.2 Objetivos específicos

- Emplear un entorno de ejecución y un modelo de lenguaje (LLM) de código abierto operativos en hardware local, facilitando el procesamiento de datos sin dependencia de la nube.

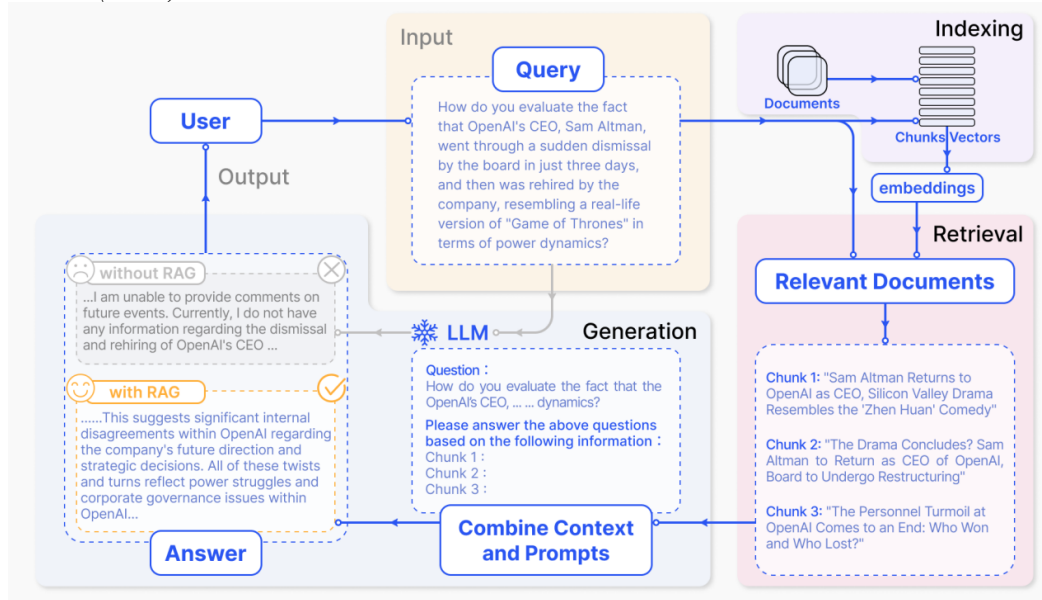
- Clasificar la información de los manuales de la empresa para conformar un repositorio de conocimiento estructurado y vectorizado, habilitando al sistema para realizar búsquedas eficientes y recuperar los datos almacenados.
- Describir el flujo de comunicación y evaluar el desempeño del prototipo funcional que integra la base de datos vectorial ChromaDB, el modelo de lenguaje y el canal de mensajería externa de WhatsApp. Esta validación se realizará midiendo la exactitud de recuperación de información y de identificación de requerimientos.

2. Marco referencial

2.1 Marco teórico

El desarrollo de sistemas modernos de inteligencia artificial generativa ha requerido la implementación de arquitecturas cuyas respuestas sean adecuadas y contextualmente relevantes. En el núcleo de estas soluciones se encuentra la arquitectura RAG (Generación Aumentada por Recuperación), un marco de trabajo tecnológico cuyo funcionamiento se instrumenta a través de un flujo de datos síncrono dividido en tres etapas secuenciales: recuperación, aumento y generación. El ciclo inicia transformando la consulta del usuario en un vector para que, en la fase de recuperación, el motor extraiga los fragmentos matemáticamente más cercanos a la solicitud. Posteriormente, el sistema integra estos fragmentos en una plantilla estructurada y los inyecta en el modelo de lenguaje (LLM), el cual ejecuta una inferencia restringida a los datos provistos [7]. En la se observa el flujo de la información de una arquitectura RAG típica.

Figura 1. Flujo de trabajo y arquitectura general de un sistema de Generación Aumentada por Recuperación (RAG)



Tomado de [3].

Para habilitar este proceso de recuperación, es necesario estructurar los datos de origen. Aquí interviene la segmentación textual estructurada (*chunking*), un proceso que divide documentos extensos en fragmentos más pequeños y semánticamente coherentes. Esta técnica previene la pérdida de contexto local y evita exceder las ventanas de memoria de los modelos de lenguaje, permitiendo que el sistema indexe y recupere únicamente la información almacenada en la base de datos [3].

Una vez segmentado, el texto debe ser traducido a un formato procesable por la máquina mediante *embeddings*. Estas representaciones matemáticas de texto en forma de vectores continuos dentro de un espacio de alta dimensionalidad permiten que los sistemas informáticos procesen y cuantifiquen la similitud semántica entre palabras o manuales completos, identificando relaciones conceptuales subyacentes más allá de la simple coincidencia de caracteres [8].

Para gestionar este vasto volumen de representaciones espaciales, se utilizan las bases de datos vectoriales. Estos sistemas de almacenamiento están especializados y optimizados para indexar, gestionar y consultar eficientemente los *embeddings* en alta dimensionalidad. A diferencia de las bases de datos relacionales tradicionales, utilizan algoritmos de búsqueda aproximada para recuperar la información matemáticamente más afín a la intención espacial de la solicitud del usuario [2].

La evaluación de esta afinidad dentro del espacio continuo requiere métricas de cálculo de distancia y similitud vectorial. Las tres métricas algorítmicas estandarizadas evidenciadas en [6] cumplen funciones específicas según el caso de uso:

- Distancia L2 al cuadrado: Calcula la distancia geométrica espacial absoluta entre los extremos de dos vectores omitiendo el cálculo de la raíz cuadrada para optimizar el rendimiento computacional. Resulta útil cuando la correspondencia estricta de coordenadas es vital, pero presenta deficiencias en el Procesamiento de Lenguaje Natural (PLN) al penalizar injustamente la comparación entre textos de distinta extensión.
- Producto punto: Evalúa simultáneamente la alineación direccional y la magnitud de los vectores sin normalizar su longitud. Esta propiedad hace que los vectores con magnitudes mayores influyan desproporcionadamente, convirtiéndolo en el estándar para sistemas de recomendación donde la magnitud representa la "intensidad" de preferencia de un usuario.
- Similitud de coseno: Evalúa de forma exclusiva la orientación direccional calculando el coseno del ángulo entre los vectores, aislando por completo la magnitud escalar. Para las arquitecturas RAG, constituye el enfoque más robusto, ya que ignora las disparidades volumétricas y permite un emparejamiento semántico de gran relevancia entre consultas breves y documentos extensos, basando la correlación en la intención temática.

El despliegue de estos sistemas impone retos computacionales que se mitigan mediante la cuantización. Esta técnica de compresión y optimización reduce la precisión numérica de los parámetros internos del modelo (por ejemplo, de 16 o 32 Bits a 8 o 4 Bits), disminuyendo drásticamente la memoria de video requerida sin comprometer la calidad semántica de las respuestas generadas [4].

Sumado a la optimización, el entorno donde operan estos modelos define su seguridad y rendimiento. La ejecución on-premise es un modelo de despliegue donde el software y los motores de inferencia operan de forma exclusiva en los servidores físicos y redes locales de la organización. Esta topología mitiga la latencia de red, reduce la dependencia de nubes públicas y fortalece la confidencialidad y soberanía sobre los datos institucionales [5].

Finalmente, la orquestación de estos componentes se logra a través de una API (Interfaz de programación de aplicaciones). Este conjunto estandarizado de protocolos funciona como la capa de integración que dirige el flujo de información bidireccional, conectando de manera estructurada canales de mensajería externos (como WhatsApp) con el motor de base de datos vectorial y el modelo de inteligencia artificial generativa [1].

2.2 Marco conceptual

Para la correcta comprensión y desarrollo del presente proyecto, resulta necesario delimitar los paradigmas y disciplinas que estructuran la solución tecnológica.

Inteligencia artificial y modelos de lenguaje grande (LLM) . Este proyecto se enmarca en los sistemas de inteligencia artificial generativa, específicamente en los Modelos de Lenguaje Grande (LLM). Estos modelos computacionales están diseñados para comprender y generar

respuestas en texto natural, funcionando como el núcleo de procesamiento encargado de interpretar las consultas de los usuarios dentro de marcos de generación aumentada [9].

Búsqueda semántica. A diferencia de los sistemas de recuperación de información tradicionales que dependen de la coincidencia exacta de caracteres o palabras clave (búsqueda léxica), la búsqueda semántica constituye un paradigma avanzado enfocado en interpretar la intención y el significado contextual detrás de una consulta. Este enfoque aborda el lenguaje natural como un problema matemático, transformando palabras, oraciones o manuales completos en coordenadas numéricas dentro de un espacio vectorial continuo [8].

Bajo este modelo, la relevancia de la información no se determina por la repetición de un término, sino por la proximidad geométrica de los vectores.

Generación aumentada por recuperación (RAG). La convergencia entre los algoritmos de búsqueda semántica y los modelos generativos da origen al marco de trabajo conocido como Generación Aumentada por Recuperación (RAG). Este paradigma sostiene que, para formular respuestas contextualizadas, la inteligencia artificial no debe limitarse a los parámetros memorizados durante su entrenamiento inicial. Por el contrario, su capacidad de predicción se enriquece en tiempo real al inyectar información externa [9].

2.3 Marco legal

El proyecto se fundamenta normativamente en la Ley General de Protección de Datos Personales [10]. Esta legislación colombiana establece los lineamientos obligatorios para la recolección, almacenamiento y procesamiento de información personal, exigiendo la seguridad de los datos y la autorización de los titulares. En el contexto del proyecto, la interacción con los usuarios se realiza a través del API de WhatsApp, un canal donde se procesan números telefónicos

y posibles datos sensibles ingresados en las descripciones de las incidencias. Para dar cumplimiento a esta normativa, la arquitectura del sistema adopta un enfoque de privacidad desde el diseño al ejecutar el modelo de lenguaje y la base de datos en hardware de la compañía, además el proveedor de la API de WhatsApp tiene su propia política de privacidad [11] alineada con la normativa colombiana.

Además de la protección de datos, la operación del asistente virtual se guía por el Marco Ético para la Inteligencia Artificial en Colombia [12]. Este documento oficial define principios básicos como la transparencia y el control humano. En la práctica del proyecto, esto se cumple al darle la opción de escalar su caso a un técnico humano si la herramienta no logra resolver su requerimiento.

3. Método

El método se divide en tres fases principales: la configuración de la infraestructura local, la vectorización de los documentos de la empresa y la integración del modelo de lenguaje con WhatsApp.

3.1 Configuración de la infraestructura y selección del modelo base

El propósito de esta etapa es desplegar el entorno de ejecución del software sobre el hardware disponible en la empresa, habilitando el procesamiento de la inferencia de la información sin conexión con proveedores de inteligencia artificial de terceros. Para ello una de las tareas a realizar es una auditoría de las especificaciones técnicas del servidor destinado, con el fin de identificar las restricciones físicas del sistema, específicamente la capacidad de la memoria de video (VRAM). Este diagnóstico permite establecer un límite de uso seguro para la tarjeta gráfica

(GPU) antes de instalar el motor de inferencia local. Con base en esta restricción, se puede determinar qué modelos de lenguaje pueden operar de forma estable sin desbordar la cantidad de memoria disponible. En la Tabla 1 se dispone la información recopilada de la auditoría de hardware.

Tabla 1. *Especificaciones técnicas de la infraestructura de hardware del servidor local*

Componente	Especificación	Detalle
CPU	Modelo	Intel® Xeon® CPU E5-4650 v2 @ 2.40GHz
	Cantidad de núcleos	10
	Cantidad de hilos	20
	Frecuencia base	2.40Ghz
	TDP	95W
	Año de lanzamiento	2014
Memoria RAM	Capacidad	256 GB
	Tipo de memoria	DDR3
Almacenamiento	Capacidad	3.5 TB HDD
GPU	Modelo	NVIDIA GeForce RTX 3050
	Frecuencia base	1.51GHz
	VRAM	6 GB
	Tipo de memoria	GDDR6
	TDP	80W
	Versión de CUDA	13.0
	Cantidad de núcleos CUDA	2304
	Año de lanzamiento	2022

Nota: Datos recolectados mediante la auditoría del equipo asignado por la empresa. TDP = Potencia de Diseño Térmico; VRAM = Memoria de Acceso Aleatorio de Video; HDD = Unidad de Disco Duro.

El hardware analizado presenta un contraste marcado entre su gran volumen de memoria RAM y sus limitaciones en velocidad de transferencia. Si bien el sistema cuenta con una capacidad física amplia, el estándar de la memoria combinado con un procesador antiguo generaría un cuello

de botella durante el proceso de inferencia. En la práctica, la velocidad de respuesta sería demasiado lenta para el usuario final. En consecuencia, se hace necesario procesar la información directamente en la GPU; esto significa que la ocupación en memoria no puede exceder la capacidad de la VRAM disponible (6 GB) para aprovechar sus altas velocidades.

Para el despliegue de los modelos locales, se evaluaron los motores de inferencia *Ollama* y *LM Studio*. Si bien ambas herramientas permiten la ejecución en segundo plano mediante línea de comandos, se seleccionó *Ollama* por su arquitectura nativa orientada a entornos de servidor Linux. *Ollama* se despliega y gestiona como un servicio del sistema, lo cual permite alta disponibilidad y un uso eficiente de la memoria [13]. Adicionalmente cuenta con compatibilidad nativa inmediata con librerías para el desarrollo de arquitectura RAG y herramientas de evaluación de desempeño como *Promptfoo* al exponer una API para la interacción con el modelo.

En la documentación oficial de *llama.cpp* (motor de inferencia de *Ollama*) se detalla según el nivel de cuantización del modelo cuánto ocupa cada uno de sus pesos o parámetros en memoria, como se observa en la Tabla 2.

Tabla 2. Métricas de rendimiento y consumo según el nivel de cuantización del modelo

Métrica	Q4_K_S	Q4_K_M	Q5_K_S	Q5_K_M	Q6_K	Q8_0
Bits/peso	4.6672	4.8944	5.5704	5.7036	6.5633	8.5008
Tamaño (GiB)	4.36	4.58	5.21	5.33	6.14	7.95
Procesamiento del prompt (t/s)	818.55	821.81	752.52	758.69	812.01	865.09
@ 512	±9.58	±21.44	±0.99	±7.43	±10.82	±8.30
Generación de texto (t/s)	76.71	71.93	69.53	67.23	58.67	50.93
@ 128	±0.20	±1.52	±0.18	±1.08	±3.13	±0.08

Nota: t/s = Tokens por segundo. Datos de rendimiento extraídos y adaptados de la documentación oficial de la herramienta de cuantización llama.cpp [14].

Por defecto, al descargar un modelo desde los repositorios oficiales de Ollama, el gestor asigna automáticamente la cuantización Q4_K_M, un formato de 4 bits configurado para reducir drásticamente el peso del archivo y facilitar la inferencia en hardware con recursos limitados.

A partir de lo anterior, es posible estimar la cantidad de parámetros que se adecúan al hardware que va a ejecutar el LLM mediante la ecuación 1.

$$VRAM(GB) \approx (P \times \text{Bytes por parámetro}) + C \#(1)$$

Donde:

P: miles de millones de parámetros del modelo.

Bytes por parámetro: cantidad de bytes que ocupa un parámetro.

C: tamaño del contexto.

Dado que la GPU tiene un límite físico de 6 GB de VRAM y considerando el almacenamiento para la ventana de contexto, se define por seguridad que el entorno de ejecución no debe superar el 80 % de la capacidad disponible. Es decir, se establece un límite de ocupación máxima de 4.8 GB, de los cuales 0.5 GB corresponden a la variable C. El margen de reserva restante del 20 % (1,2 GB) contempla la memoria requerida para la ejecución concurrente del modelo de representaciones vectoriales embeddinggemma.

$$P \approx \frac{VRAM(GB) - C}{\text{Bytes por parametro}}$$

$$P \approx \frac{4.8 - 0.5}{\frac{4.8944}{8}} \approx 7.02$$

Con el resultado anterior es posible definir como uno de los criterios de selección del modelo que su cantidad de parámetros sea igual o inferior a 7 mil millones de parámetros (7 B) y

adicionalmente, bajo el criterio de vigencia tecnológica, se descartan iteraciones previas de una misma familia de modelos. Por consiguiente, ante la disponibilidad de versiones recientes con arquitecturas optimizadas, se omiten modelos predecesores (como Gemma 2 o Qwen 2.5) para priorizar sus versiones más actuales (como Gemma 3 o Qwen 3). En la Tabla 3 se realiza el filtrado y selección preliminar de los modelos de lenguaje analizados.

Tabla 3. *Matriz de evaluación y selección preliminar de modelos de lenguaje*

Familia de modelos	Versiones y tamaños evaluados	Estado de viabilidad	Justificación
Gemma (Google)	Gemma 3: 270M, 1B, 4B, 12B, 27B	Viable (Gemma 3:4B)	Gemma 2 se descarta por obsolescencia. De Gemma 3, se descartan las versiones con más de 7B de parámetros. La versión de 4B es la mayor permitida.
	Gemma 2: 2B, 9B, 27B		
Qwen (Alibaba)	Qwen 3: 4B, 8B, 32B, 72B, 110B, 235B	Viable (Qwen 3:4B)	Qwen 2.5 se descarta por obsolescencia. De Qwen 3, la versión de 8B excede la memoria. Se mantiene la versión de 4B.
	Qwen 2.5: 0.5B, 1.5B, 3B, 7B, 14B, 32B, 72B		
Llama (Meta)	Llama 4: 16x17B, 128x17B	Viable (Llama 3.2:3B)	Las versiones 4, 3.3 y 3.1 se descartan por superar el límite de parámetros. Llama 3.2 cumple con el tamaño y vigencia con su modelo de 3B.
	Llama 3.3: 70B Llama 3.2: 1B, 3B Llama 3.1: 8B, 70B, 405B		
DeepSeek	R1: 1.5B, 7B, 8B, 14B, 32B, 70B, 671B	Viable (DeepSeek-R1:7B)	V3 se descarta por tamaño. La versión R1 de 7B cumple el límite máximo exacto.
	V3: 671B		
Mistral	Large 3: 675B	Descartado	Todas las versiones evaluadas exceden el límite máximo de 7B permitido por el hardware.
	Mixtral: 8x7B, 8x22B		
Phi (Microsoft)	Phi 4: 14B Phi 3: 3.8B, 14B	Descartado	Phi 4 y Phi 3 (14B) exceden la VRAM. Phi 3 (3.8B) se descarta por obsolescencia frente a

Familia de modelos	Versiones y tamaños evaluados	Estado de viabilidad	Justificación
			su predecesor directo.
Grok	Grok-1: 314B	Descartado	Excede la capacidad de VRAM.
Orca (Microsoft)	Orca 2: 7B, 13B	Descartado	Descartados por obsolescencia frente a familias de modelos más recientes.
	Orca mini: 3B, 7B, 13B, 70B		
Dolphin	Dolphin3: 7B	Descartado	Descartado por obsolescencia frente a familias de modelos más recientes.

Nota: La nomenclatura “B” hace referencia a miles de millones de parámetros (*billions*), y “M” a millones de parámetros. La selección preliminar asume como restricción principal el límite calculado de 7 B de parámetros de acuerdo con las especificaciones del servidor local.

A partir de la selección preliminar consolidada en la Tabla 3, los modelos viables que avanzan a la evaluación de rendimiento son: Gemma 3:4B, Llama 3.2:3B, DeepSeek-R1:7B y Qwen 3:4B. Para determinar cuál de estas arquitecturas ofrece el mejor desempeño, se diseñó una prueba de inferencia utilizando la herramienta *Promptfoo*, basada en la ejecución de cinco *prompts* orientados a la generación de textos extensos. Esta evaluación tiene como objetivo contrastar dos métricas clave: la velocidad de inferencia y la latencia de las respuestas. Las condiciones iniciales de la prueba fueron las siguientes:

- Instrucción del sistema (*System prompt*): "Eres un asistente de IA experto, analítico y muy detallista. Tu objetivo en esta prueba es proporcionar respuestas exhaustivas, largas y profundamente explicadas al usuario. Utiliza múltiples párrafos y estructura bien tu texto."
- Entrada del usuario: Estructurada bajo la variable $\{\{pregunta\}\}$, la cual fue sustituida iterativamente por los cinco *prompts* definidos para la prueba.
- Temperatura de generación: Valor por defecto de cada uno de los modelos.
- Tamaño de la ventana de contexto: Capacidad por defecto establecida en la arquitectura de cada modelo.

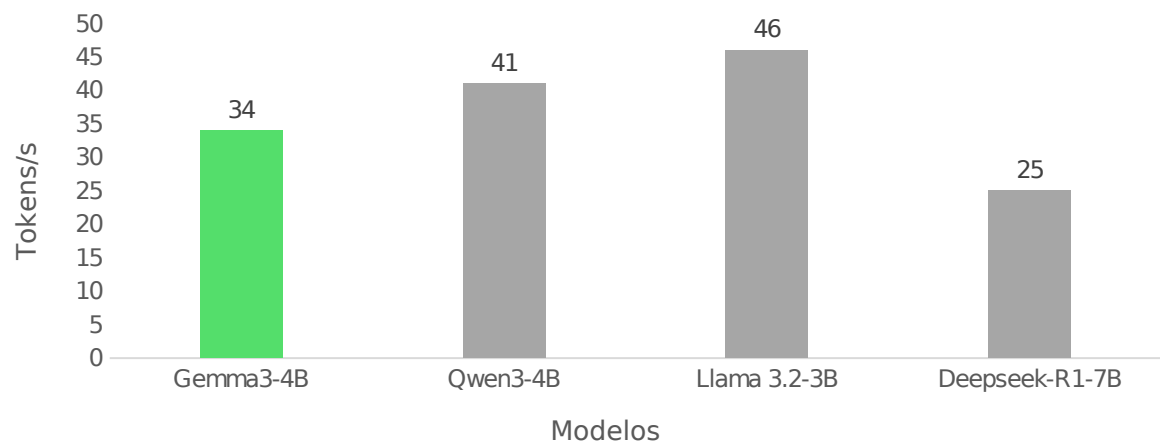
Es importante destacar que las evaluaciones no se diseñaron para someter a los modelos a condiciones idénticas de ejecución paramétrica. El propósito fundamental de esta prueba es analizar el comportamiento y rendimiento de cada modelo de lenguaje operando en su estado base original. En consecuencia, esta aproximación metodológica permite identificar cuál de las arquitecturas ofrece el desempeño más eficiente sobre las restricciones del hardware, antes de aplicar cualquier técnica de ajuste. En la se observa una matriz con los resultados para tres de los *prompts* de prueba en los modelos.

Figura 2. Resultados de la prueba de inferencia comparativa en la interfaz de Promptfoo

Variables		Outputs			
		ollama/gemma3	ollama/qwen3.0b	ollama/deepseek-r1	ollama/llama3.2
Description		pregunta			
Redacción Creativa (Storytelling)	Escribe una historia épica de ciencia ficción y muy detallada (de al menos 5 párrafos) sobre una tripulación de astronautas que descubre ruinas antiguas en una luna de Júpiter. Describe los paisajes, las emociones y el misterio del hallazgo.	PASS	PASS	PASS	PASS
		El polvo rojo de Gliese VI, la luna de Júpiter en la que ahora se adentraban, se arremolinaba alrededor de las botas de la Capitana Eva Rostova con una tenacidad silenciosa. La luz de Júpiter, una monstruosa esfera amarilla que dominaba el cielo, pro...	Historia Épica: Las Ruinas de Eos	En el corazón de Júpiter, allí en la Luna Europa, una teams of astronautas al viajar was dispuesta a descubrir maravillas scientificallly y misterios. Después de años de investigación y viaje, se encontró en las playas frío de esta satélite satelital.	La Excursión a la Luna de Júpiter
		...	La nave Odiseo aterrizó con delicadeza en la superficie de Eos, una luna de Júpiter que había sido descubierta recientemente por sensores de exploración interestelar. El paisaje era un manto de hielo azul-pl...	...	En el año 2234, la tripulación del spacecraft "Aurora" estaba en misión para explorar las lunas de Júpiter. El capitán Jameson había seleccionado esta misión como la más importante de su carrera, ya que la luna
Explicación Científica (Factual)	Explica detalladamente y paso a paso cómo funciona el sistema inmunológico humano desde el momento exacto en que un virus desconocido entra al cuerpo hasta que es neutralizado. Usa analogías claras y extiende tu explicación para que sea muy completa.	PASS	PASS	PASS	PASS
		(Absolutamente! Con gusto te proporcionaré una explicación exhaustiva y detallada de cómo funciona el sistema inmunológico humano, desde la invasión de un virus desconocido hasta su neutralización. La complejidad de este sistema es asombrosa, y utili...	Paso a Paso: Cómo Funciona el Sistema Inmunológico Humano al Encontrarse con un Virus Desconocido	El sistema inmunológico humano es una defensa avanzada contra los invasores, incluyendo bacterias, virus, plasmódi y otros entes peligrosos. Cuando un virus desconocido llega al cuerpo, el sistema inmunológico desarrolla de manera cuidadosa para com...	El Sistema Inmunológico Humano: Un Ejército Defensor contra los Virus
		...	El sistema inmunológico humano es una red compleja y dinámica de defensas que actúa como un "sistema de seguridad" para proteger el cuerpo contra p...	Tokens: 1.850 (132+1.727) Latency: 62.890 ms Tokens/Sec: 27	El sistema inmunológico humano es un complejo sistema de defensa que nos protege del daño causado por patógenos como virus, bacterias y otros microorganismos. Cuando un virus
Ensayo Analítico (Razonamiento lógico)	Redacta un ensayo argumentativo extenso y profundo sobre el impacto de la Inteligencia Artificial en el futuro del trabajo y la educación universitaria. Debe contener una introducción clara, un desarrollo con argumentos a favor y en contra, y una con...	PASS	PASS	PASS	PASS
		El filo de la Navaja Digital: Inteligencia Artificial, el Futuro del Trabajo y la Educación Universitaria	The Impact of Artificial Intelligence on the Future of Work and Higher Education: A Balanced Analysis	El Impacto de la Inteligencia Artificial en el futuro del trabajo y la educación universitaria	El Impacto de la Inteligencia Artificial en el Futuro del Trabajo y la Educación Universitaria: Un Análisis Profundo
		La Inteligencia Artificial (IA) ha emergido como una fuerza transformadora de una magnitud sin precedentes en nuestra era, permeando cada ve...	Introduction	La revolución tecnológica en las últimas décadas ha generado avances significativos que han transformado nuestras vidas. La Inteligencia	La Inteligencia Artificial (IA) ha sido un tema de debate intenso en las últimas décadas, especialmente en relación con su impac

Nota: Captura de pantalla de la interfaz web de Promptfoo durante la ejecución de las pruebas. Las columnas representan los modelos desplegados localmente (Gemma 3, Qwen 3, DeepSeek-R1 y Llama 3.2), y las filas exponen los *prompts* utilizados junto con las salidas generadas.

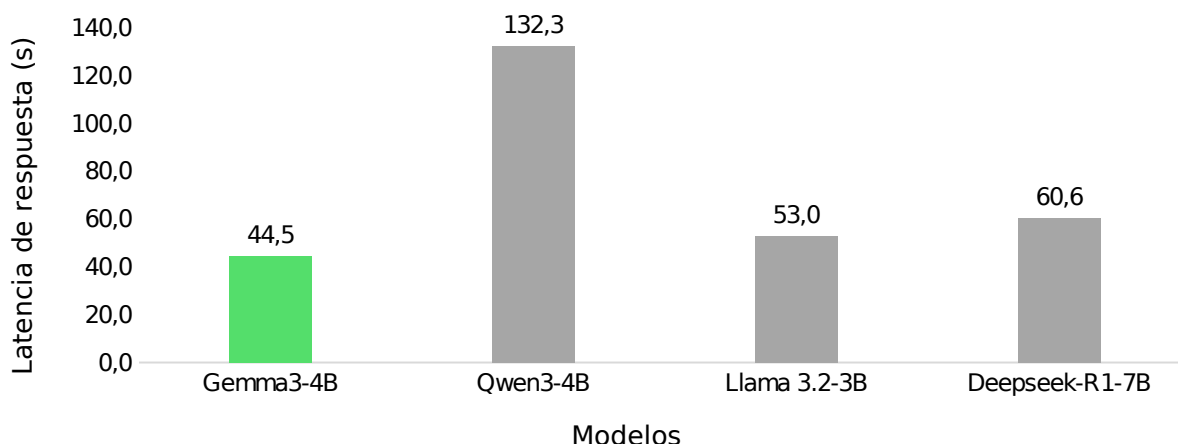
Como resultado de la extracción de métricas de la plataforma, la presenta el promedio de la velocidad de inferencia (medida en *tokens* por segundo) alcanzada por cada arquitectura durante la ejecución de los ensayos. Por su parte, los resultados correspondientes a la prueba de latencia se detallarán más adelante en la .

Figura 3. Comparativa del promedio de velocidad de inferencia entre los modelos finalistas

Nota: La velocidad de inferencia se mide en tokens por segundo (tokens/s) e indica la cantidad de unidades de texto (palabras o fragmentos de palabras) que el modelo es capaz de generar en un segundo durante la redacción de la respuesta.

A partir de los datos recolectados, el modelo Llama 3.2:3B demostró el mejor rendimiento bruto en la generación de texto, alcanzando una velocidad máxima de 46 *tokens/s*, seguido de cerca por la arquitectura Qwen 3:4B con 41 *tokens/s*. Por el contrario, el modelo DeepSeek-R1:7B presentó el menor desempeño de la comparativa, registrando 25 *tokens/s*. Este comportamiento representa un resultado estadísticamente previsible; dado que DeepSeek posee una mayor cantidad de parámetros frente a sus competidores, exige una carga computacional mayor y un consumo de memoria de video (VRAM) más intensivo por parte de la GPU para lograr la inferencia.

Figura 4. Comparativa del promedio de latencia de respuesta entre los modelos finalistas



Nota: El tiempo de respuesta (latencia) se mide en segundos (s) e indica el lapso transcurrido desde el envío de la instrucción hasta la culminación de la respuesta generada.

Como se observa en la , el modelo Qwen 3:4B registró un tiempo de respuesta atípicamente alto, superior a los 130 segundos, una latencia que lo hace inviable para su implementación en un chat. En contraste, la arquitectura Gemma 3:4B demostró el mejor desempeño en esta métrica, registrando el menor retardo de todo el grupo evaluado.

Al contrastar ambos indicadores, se seleccionó el modelo Gemma 3:4B, ya que presenta el equilibrio operativo más adecuado para los requerimientos del proyecto. Si bien su velocidad de generación es moderada (34 *tokens/s*), su ventaja comparativa radica en su reducida latencia. Esta característica es crítica para la viabilidad del sistema, puesto que busca ofrecer tiempos de espera mínimos para el usuario final al momento de realizar consultas, optimizando la experiencia de uso sin exceder los límites del hardware disponible.

3.2 Generación de *embeddings* e indexación de documentos en motor de base de datos vectorial

Para el motor de base de datos vectorial se optó por ChromaDB debido a su compatibilidad nativa con el ecosistema de Ollama. Con el propósito de disponer de un entorno de ejecución

aislado, escalable y libre de conflictos de dependencias, el despliegue de este motor se realizó mediante un contenedor de Docker.

En cuanto al procesamiento del texto, la generación de las representaciones vectoriales se llevó a cabo utilizando el modelo embeddinggemma como función de *embeddings*. La selección de este modelo se fundamenta en que está basado en Gemma 3 y posee arquitectura multilingüe, la cual ofrece soporte para el idioma español. Esta última característica es importante para el proyecto, ya que permite que la intención, el contexto y el vocabulario técnico de los documentos se capturen con un nivel adecuado de correspondencia semántica.

Para organizar y almacenar esta información de manera eficiente, se empleó el concepto de "colección" (*collection*) propio de la arquitectura de ChromaDB. Operativamente, una colección funciona como el análogo a una tabla en una base de datos relacional tradicional; representa el contenedor lógico donde se agrupan los documentos originales, sus metadatos asociados (como los identificadores y las etiquetas) y los vectores numéricos resultantes.

Al momento de instanciar la colección para este sistema, su entorno dimensional se configuró estableciendo el parámetro “hnsw:space” con el valor “cosine”. Esta definición algorítmica indica que el motor utilizará la métrica de similitud del coseno para calcular la distancia entre la consulta del usuario y los vectores de la colección. Dicha configuración evalúa el ángulo y la orientación de los vectores en el espacio multidimensional, lo que permite procesar el lenguaje natural priorizando la relevancia del significado por encima de la longitud del texto.

Para la preparación de la base de conocimiento, el contenido de los manuales institucionales se extrajo y normalizó en un archivo JSON con la siguiente estructura:

- id: Identificador único del bloque de información.
- title: Nombre descriptivo de la incidencia o del procedimiento técnico.

- **content:** Instrucciones detalladas para la resolución del problema. En lugar de aplicar algoritmos de segmentación automática (*chunking*) sobre los manuales extensos, este campo se redactó de forma manual. Esta decisión obedece a que la partición algorítmica de documentos técnicos suele fracturar los pasos secuenciales de una solución o aislar el contexto. Al estructurar la información en respuestas atómicas, directas y completas desde el origen, se optimiza la ingesta de datos, permitiendo que el modelo de lenguaje recupere instrucciones.
- **images:** Arreglo que contiene las rutas de archivo local dirigidas a las imágenes de apoyo.
- **documents:** Arreglo de rutas de archivo local que apuntan a los manuales complementarios en formato PDF.
- **tags:** Palabras clave asociadas para optimizar la indexación y la búsqueda semántica del modelo.

En la Figura 5 se ejemplifica la estructura del archivo JSON.

Figura 5. *Estructura del archivo JSON para la normalización de la información*

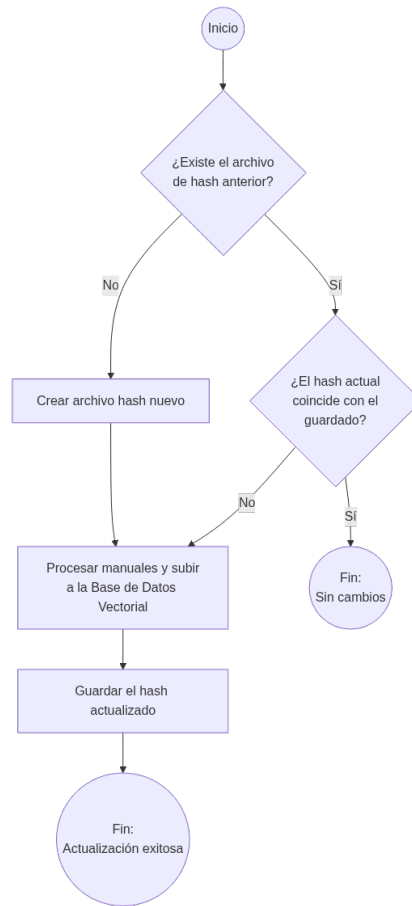
```
[
  {
    "id": "wifi_windows",
    "title": "Conexión WiFi Windows",
    "content": "Para configurar WiFi, abre Configuración > Red e Internet > WiFi.",
    "images": [
      "https://soporte.unidrogas.co/images/wifi_paso1.png",
      "https://soporte.unidrogas.co/images/wifi_paso2.png"
    ],
    "tags": ["wifi", "internet", "conexión", "windows"]
  },
  {
    "id": "error_impresora",
    "title": "Solución Error de Impresora",
    "content": "Si tu impresora no imprime, revisa conexión USB, tinta y reinstala drivers.",
    "images": [
      "https://soporte.unidrogas.co/images/impresora_error1.png"
    ],
    "tags": ["impresora", "error", "driver", "usb"]
  },
  {
    "id": "correo_outlook",
    "title": "Configurar Correo en Outlook",
    "content": "Abre Outlook > Archivo > Agregar Cuenta, introduce correo y contraseña.",
    "images": [
```

Nota: Fragmento del archivo JSON donde se observan los campos clave (id, title, content, images, tags) estructurados como objetos independientes listos para su vectorización.

Una vez estructurados los datos, se procedió con la vectorización y la ingesta de la información en el motor ChromaDB mediante un programa desarrollado en Python. Con el fin de optimizar los recursos computacionales del servidor y evitar el doble procesamiento de información redundante, la lógica del programa implementa un sistema de validación de estado basado en el cálculo de un *hash* MD5.

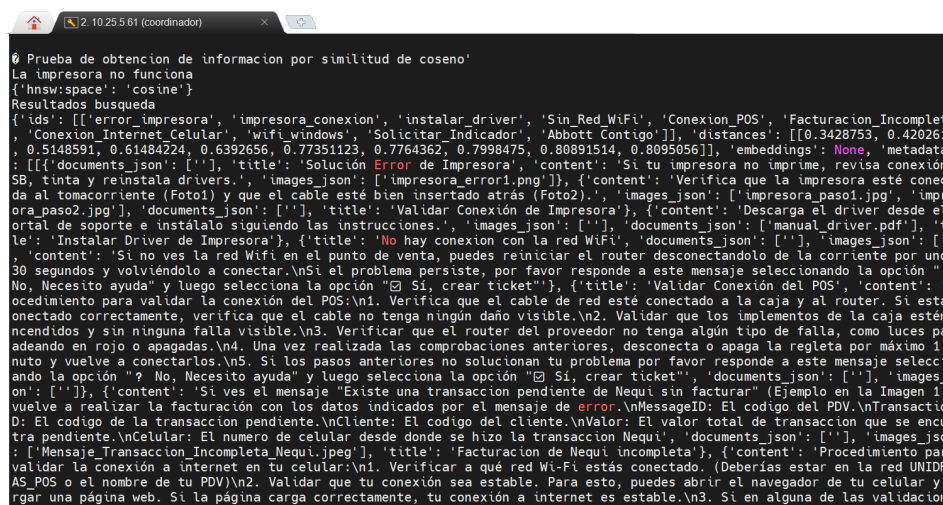
El flujo de ejecución del programa inicia con la lectura del repositorio de manuales en formato JSON y el cálculo inmediato de su *hash*. A continuación, compara el valor calculado con el *hash* almacenado durante la última ejecución exitosa. Si ambos valores coinciden, significa que no se ha modificado el JSON y, por lo tanto, no es necesario cargar la información a la base de datos. Por el contrario, si los valores difieren o no existe un *hash* previo al cual comparar, el sistema subirá el repositorio a ChromaDB y finalizará guardando el *hash* actualizado en el archivo local. En la Figura 6 se detalla visualmente la arquitectura lógica de este proceso.

Figura 6. Diagrama de flujo del programa de validación e ingesta de datos



Para validar el correcto funcionamiento y la accesibilidad del repositorio, se ejecutaron pruebas de consumo a través de la API de ChromaDB. Como se evidencia en la Figura 7, las pruebas confirman que el sistema almacena los *embeddings* y también puede recuperar la información satisfactoriamente.

Figura 7. Resultados de las pruebas de consulta y recuperación de información semántica desde la API de ChromaDB



```

0 Prueba de obtencion de informacion por similitud de coseno'
La impresora no funciona
{'hns:space': 'cosine'}
Resultados buscados
[{'ids': [['error_impresora', 'impresora_conexion', 'instalar_driver', 'Sin_Red_WiFi', 'Conexion_POS', 'Facturacion_Incompleta', 'Conexion_Internet_Celular', 'wifi_windows', 'Solicitar_Indicador', 'Abbott_Contigo']], 'distances': [[0.3428753, 0.420261, 0.5148591, 0.61484224, 0.6392656, 0.77351123, 0.7764362, 0.7998475, 0.80891514, 0.8095056]], 'embeddings': None, 'metadata': {'documents_json': [''], 'title': 'Solución Error de Impresora', 'content': 'Si tu impresora no imprime, revisa conexión SB, tinta y reinstala drivers.', 'images_json': ['impresora_error1.png']}, {'content': 'Verifica que la impresora esté conectada al tomacorriente (Foto1) y que el cable esté bien insertado atrás (Foto2).', 'images_json': ['impresora_paso1.jpg', 'impresora_paso2.jpg'], 'documents_json': [''], 'title': 'Validar Conexión de Impresora', 'content': 'Descarga el driver desde el portal de soporte e instálalo siguiendo las instrucciones.', 'images_json': [''], 'documents_json': ['manual_driver.pdf'], 'title': 'Instalar Driver de Impresora', 'title': 'No hay conexión con la red WiFi', 'documents_json': [''], 'images_json': [''], 'content': 'Si no ves la red Wifi en el punto de venta, puedes reiniciar el router desconectándolo de la corriente por unos 30 segundos y volviéndolo a conectar.\nSi el problema persiste, por favor responde a este mensaje seleccionando la opción "¿No, Necesito ayuda" y luego selecciona la opción "¿Sí, crear ticket"', 'title': 'Validar Conexión del POS', 'content': 'Procedimiento para validar la conexión del POS:\n1. Verifica que el cable de red esté conectado a la caja y al router. Si está conectado correctamente, verifica que el cable no tenga ningún daño visible.\n2. Validar que los implementos de la caja estén encendidos y sin ninguna falla visible.\n3. Verificar que el router del proveedor no tenga algún tipo de falla, como luces parpadeando en rojo o apagadas.\n4. Una vez realizadas las comprobaciones anteriores, desconecta o apaga la regleta por máximo 1 minuto y vuelve a conectarlos.\n5. Si los pasos anteriores no solucionan tu problema por favor responde a este mensaje seleccionando la opción "¿No, Necesito ayuda" y luego selecciona la opción "¿Sí, crear ticket"', 'documents_json': [''], 'images_json': [''], 'content': 'Si ves el mensaje "Existe una transacción pendiente de Nequi sin facturar" (Ejemplo en la Imagen 1), vuelve a realizar la facturación con los datos indicados por el mensaje de error.\nMensajeID: El código del PDV.\nTransactionID: El código de la transacción pendiente.\nCliente: El código del cliente.\nValor: El valor total de transacción que se encuentra pendiente.\nCelular: El número de celular desde donde se hizo la transacción Nequi', 'documents_json': [''], 'images_json': ['Mensaje_Transaccion_Incompleta_Nequi.jpeg'], 'title': 'Facturación de Nequi incompleta', 'content': 'Procedimiento para validar la conexión a internet en tu celular:\n1. Verificar a qué red Wi-Fi estás conectado. (Deberías estar en la red UNIDRO AS_POS o el nombre de tu PDV)\n2. Validar que tu conexión sea estable. Para esto, puedes abrir el navegador de tu celular y cargar una página web. Si la página carga correctamente, tu conexión a internet es estable.\n3. Si en alguna de las validaciones

```

Nota: Captura de pantalla de la respuesta (JSON) retornada por el motor de búsqueda vectorial.

Al ejecutar una consulta semántica, el motor vectorial procesa la entrada del usuario y la compara matemáticamente contra el repositorio indexado. Como resultado de esta operación, el sistema retorna un objeto estructurado en formato JSON que contiene la información recuperada. Este paquete de datos consolida los metadatos de la incidencia y los fragmentos de texto (*chunks*) listos para ser inyectados en el modelo de lenguaje, y adicionalmente expone una métrica de distancia calculada a partir de la similitud del coseno entre los embeddings almacenados en la base de datos y la solicitud original. Este valor numérico resulta fundamental; al representar una distancia espacial, los valores menores indican una mayor cercanía semántica, validando de forma más estricta la pertinencia del documento seleccionado antes de generar una respuesta.

A continuación, en la Tabla 4, se desglosa la función y el propósito de cada uno de los parámetros retornados por el sistema.

Tabla 4. *Estructura de respuesta de la API de ChromaDB tras la recuperación semántica*

Objeto	Descripción
ids	Devuelve el identificador único del documento recuperado.
distances	Representa la distancia espacial calculada bajo la configuración de similitud del coseno. Dado que es una métrica de distancia, los valores menores indican una mayor proximidad semántica entre la intención de la consulta del usuario y el documento recuperado.
metadata	Retorna intacta la estructura original de la incidencia (título, contenido y multimedia). Es fundamental para que el sistema pueda extraer y adjuntar imágenes en la respuesta final de WhatsApp.
documents	Contiene el fragmento de texto procesado que incluye el título, las etiquetas y el contenido concatenado. Este es el insumo de contexto exacto que se le inyectará al LLM para generar la respuesta final.
included	Lista de parámetros de confirmación que ChromaDB fue instruido para devolver en la consulta.

La correcta estructuración y disponibilidad de estos parámetros confirma la viabilidad del proceso de ingesta y recuperación. Estos datos en conjunto constituyen el contexto semántico validado que, en las etapas posteriores de la ejecución, servirá como base de conocimiento para que el modelo de inteligencia artificial estructure la respuesta hacia el usuario final.

3.3 Integración del modelo de lenguaje y WhatsApp

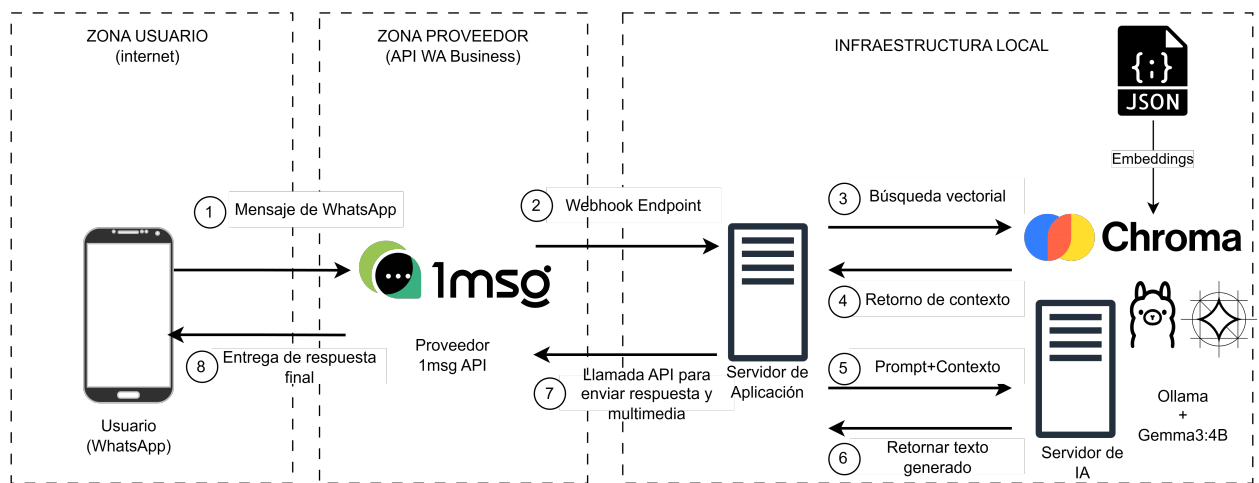
En la tercera fase del proyecto, los datos recuperados de ChromaDB se utilizan para estructurar la respuesta final. El sistema toma la información de los manuales y los recursos multimedia, y los envía como contexto al modelo de lenguaje (LLM).

Al integrar estos datos en las instrucciones al *prompt* del modelo, la inteligencia artificial redacta una solución técnica basada estrictamente en la información de la empresa. Finalmente, esta respuesta es enviada al usuario a través de la interfaz de WhatsApp, cerrando así el ciclo de atención automatizada.

Para gestionar esta comunicación bidireccional, se utilizó 1msg como proveedor de la API de WhatsApp. La recepción de los mensajes entrantes se implementó mediante la configuración

de un *webhook* en la plataforma del proveedor, el cual apunta a un *endpoint* expuesto en un servidor con salida a internet. Esta arquitectura de red permite que el sistema escuche y capture las consultas de los usuarios en tiempo real desencadenando el flujo de procesamiento para retornar la solución directamente al chat de origen. En la Figura 8 se ilustra la arquitectura lógica y el flujo secuencial de los datos, detallando la interacción entre los servicios de terceros y la infraestructura distribuida propia.

Figura 8. Diagrama de arquitectura y flujo de procesamiento para la generación de respuestas

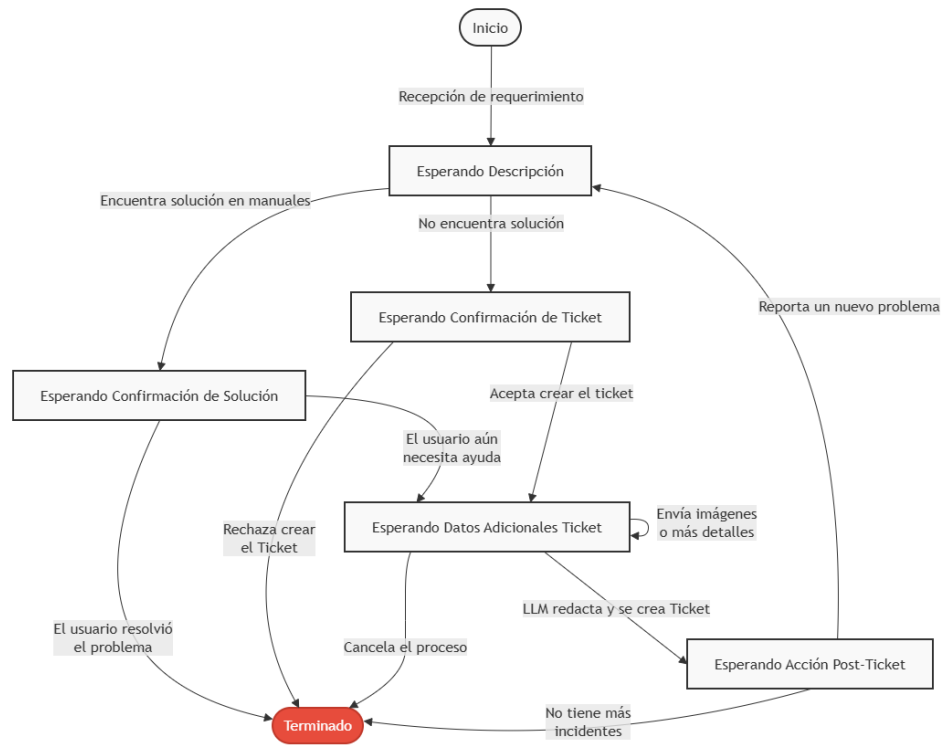


Para comprender a cabalidad la arquitectura distribuida del sistema expuesta en la Figura 8, el flujo de procesamiento se ha estructurado en ocho etapas secuenciales.

1. **Recepción del mensaje:** El usuario escribe el mensaje y dispara un evento automático (*webhook*) dirigido hacia el *endpoint* configurado con el servidor de aplicación local.
2. **Captura vía *webhook*:** El servidor de aplicación responde la petición al proveedor inmediatamente para no causar reintentos de envío.

3. Búsqueda vectorial: El servidor extrae el cuerpo del mensaje, lo convierte en *embeddings* utilizando el modelo configurado (embeddinggemma) y ejecuta una consulta basándose en la similitud de coseno en la base de datos vectorial ChromaDB.
4. Retorno de contexto: ChromaDB devuelve la información junto con los metadatos asociados.
5. Inyección en el *prompt*: El servidor de aplicación construye una instrucción estructurada que concatena el texto recuperado junto a directrices estrictas de comportamiento. Esta información se envía al servidor de inteligencia artificial para ser procesada por el modelo Gemma 3 a través de Ollama.
6. Generación de la respuesta: El LLM procesa la información y redacta una solución coherente, limitándose al contexto proporcionado para evitar alucinaciones o el uso de datos externos a la organización.
7. Llamada a la API para respuesta y multimedia: El servidor local recibe el texto generado. Si los metadatos recuperados incluyen rutas a archivos, el sistema adjunta estos archivos multimedia locales al paquete de envío.
8. Entrega de respuesta: El proveedor envía el mensaje al usuario, culminando el proceso.

Para gestionar la interacción con el usuario y lograr la resolución de su incidencia, el sistema implementa una máquina de estados que rige la lógica de la comunicación. Como se detalla a continuación en la Figura 9, este flujo conversacional define las diferentes etapas y transiciones desde la recepción inicial del requerimiento, contemplando distintos escenarios prácticos como la escalación mediante la creación de un ticket de soporte si el sistema no logra encontrar una respuesta adecuada en la base de conocimiento local.

Figura 9. Diagrama de estados del flujo conversacional para la gestión de incidencias

De acuerdo con la Figura 9, el ciclo de asistencia comienza cuando el sistema registra la solicitud del usuario e ingresa al estado “Esperando Descripción”. A partir de este punto, el flujo algorítmico se bifurca basándose en los resultados de la búsqueda vectorial en la base de conocimiento:

- Flujo de resolución automática: Si el sistema encuentra una respuesta pertinente en los manuales, la estructura y la envía, transitando al estado “Esperando Confirmación de Solución”. Si el usuario valida que su problema fue resuelto con la información brindada, la máquina de estados da por cerrado el ciclo y pasa al estado “Terminado”.
- Flujo de escalación: Si desde la primera etapa el sistema no logra encontrar una solución adecuada, o si tras enviar una posible respuesta el usuario indica que aún necesita ayuda,

el flujo se redirige de manera automática hacia el estado “Esperando Confirmación de Ticket”.

En la etapa de escalación, el sistema solicita autorización para derivar el caso al área de soporte técnico humano. Si el usuario rechaza la opción, el proceso finaliza. Si el usuario acepta, el sistema avanza al estado “Esperando Datos Adicionales Ticket”. Cabe destacar que este es un estado iterativo, lo que significa que el usuario puede enviar múltiples mensajes con descripciones adicionales, o adjuntar imágenes detalladas del error.

Una vez el usuario termina de suministrar el contexto, el modelo de lenguaje (LLM) entra nuevamente en acción para analizar, resumir y redactar formalmente el ticket de soporte. Tras la creación exitosa del reporte, el sistema transita a “Esperando Acción Post-Ticket”, donde evalúa si el usuario da por concluida la interacción o si desea reportar un inconveniente distinto. En este último escenario, el sistema demuestra su flexibilidad al reiniciar el ciclo, regresando de inmediato al estado de “Esperando Descripción” para atender la nueva solicitud sin necesidad de que el usuario vuelva a iniciar la conversación manualmente.

3.4 Alcance y limitaciones

Para una correcta interpretación de los resultados y del alcance del presente proyecto, es importante establecer los límites operativos y metodológicos bajo los cuales se desarrollaron y evaluaron las pruebas:

- *Enfoque de la evaluación:* Las pruebas documentadas midieron exclusivamente la exactitud de la etapa de recuperación semántica, es decir, la capacidad del modelo para identificar y extraer el documento correcto desde la base de conocimiento. No se estableció

una métrica cuantitativa para evaluar la calidad, coherencia o precisión técnica de la respuesta final generada por el modelo de lenguaje.

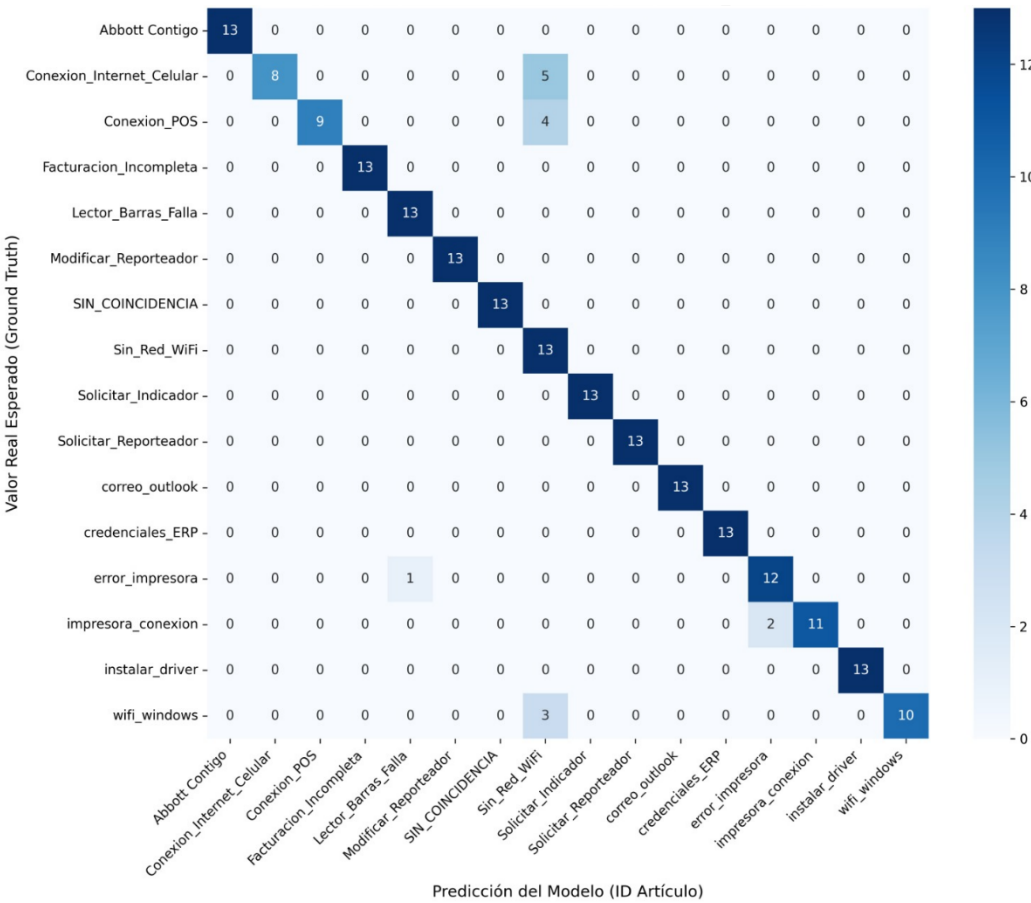
- *Distribución del banco de pruebas:* El banco de evaluación compuesto por 208 consultas fue construido de manera controlada y con una distribución balanceada (13 consultas por cada requerimiento). Aunque este diseño es idóneo para medir el rendimiento del algoritmo de clasificación, no refleja la distribución estadística ni la frecuencia real de las incidencias reportadas en la operación diaria de la empresa.
- *Comparativa de modelos de lenguaje:* Las pruebas de inferencia para la selección del modelo base se realizaron conservando las configuraciones por defecto de cada arquitectura, sin aplicar un proceso de igualación de parámetros de ejecución. El objetivo fue evaluar el rendimiento en su estado original sobre las limitaciones del hardware, mas no realizar una prueba comparativa estandarizada sobre su idoneidad para la tarea específica.
- *Concurrencia de usuarios:* Las validaciones del prototipo funcional integrado con la API de WhatsApp se ejecutaron en un entorno controlado con un único usuario interactuando a la vez. El sistema no fue sometido a pruebas de estrés o escenarios de concurrencia que permitan evidenciar el comportamiento de la latencia y la memoria gráfica ante peticiones simultáneas masivas. M

4. Resultados

Con el propósito de medir la precisión y el rendimiento del sistema, se estructuró un banco de pruebas compuesto por 208 consultas automatizadas, distribuidas equitativamente en 13 casos por cada intención o categoría. Cada solicitud de prueba fue emparejada previamente con el

documento específico que el modelo debía recuperar. Una interacción se catalogó como exitosa (acierto) únicamente cuando el sistema logró asociar la consulta ingresada con el identificador (id) exacto del artículo esperado en la base de conocimiento, adjuntando la información correcta. En la Figura 10 se muestran los resultados de la prueba

Figura 10. Matriz de confusión de la tasa de aciertos en la identificación de requerimientos



Nota: La matriz ilustra el desempeño de la etapa de búsqueda semántica. El eje vertical representa la intención original de la consulta de prueba, mientras que el eje horizontal indica el documento finalmente recuperado por el sistema. Los valores concentrados en la diagonal principal corresponden a los aciertos.

El análisis de la Figura 10 revela que la etapa de recuperación de información posee una alta capacidad de generalización y comprensión del lenguaje natural, logrando recuperar

correctamente la gran mayoría de las incidencias. Los casos en los que el sistema falló se deben al solapamiento semántico. Un ejemplo evidente es la categoría “*Sin_Red_WiFi*”, la cual atrajo falsos positivos de “*Conexion_POS*”, “*Conexion_Internet_Celular*” y “*wifi_windows*” debido a la alta similitud de su vocabulario.

Para complementar este análisis visual y cuantificar de manera exacta el desempeño del modelo, se extrajeron las métricas de clasificación estándar para cada una de las intenciones evaluadas. A continuación, la Tabla 5 detalla estos resultados.

Tabla 5. *Métricas de rendimiento de la búsqueda semántica por categoría.*

	<i>Precision</i>	<i>Recall</i>	<i>F1-score</i>	<i>Support</i>
Abbott Contigo	1.00	1.00	1.00	13
Conexion_Internet_Celular	1.00	0.62	0.76	13
Conexion_POS	1.00	0.69	0.82	13
Facturacion_Incompleta	1.00	1.00	1.00	13
Lector_Barras_Falla	0.93	1.00	0.96	13
Modificar_Reporteador	1.00	1.00	1.00	13
SIN_COINCIDENCIA	1.00	1.00	1.00	13
Sin_Red_WiFi	0.52	1.00	0.68	13
Solicitar_Indicador	1.00	1.00	1.00	13
Solicitar_Reporteador	1.00	1.00	1.00	13
correo_outlook	1.00	1.00	1.00	13
credenciales_ERP	1.00	1.00	1.00	13
error_impresora	0.86	0.92	0.89	13
impresora_conexion	1.00	0.85	0.92	13
instalar_driver	1.00	1.00	1.00	13
wifi_windows	1.00	0.77	0.87	13
accuracy			0.93	208
macro avg	0.96	0.93	0.93	208
Weighted avg	0.96	0.93	0.93	208

Nota. La columna *precision* (precisión) mide la exactitud de las predicciones del modelo, donde un valor de 1.00 indica cero falsos positivos. *Recall* (exhaustividad) evalúa la capacidad del modelo para identificar todos los casos reales de una categoría, donde 1.00 significa cero falsos

negativos. *FI-score* representa la media armónica entre la precisión y la exhaustividad, proporcionando una medida de equilibrio general. Finalmente, *support* (soporte) indica la cantidad de consultas de prueba evaluadas por cada requerimiento (13 muestras por categoría).

4.1 Resultados de recuperación semántica e identificación de requerimientos

Según los datos consolidados en la Tabla 5, el sistema alcanzó una exactitud global del 93 % en la identificación del documento esperado a través de 208 pruebas automatizadas. Las categorías con vocabulario altamente específico obtuvieron puntajes perfectos (1.00), lo cual ratifica que el modelo embeddinggemma y la configuración de similitud de coseno en ChromaDB logran aislar correctamente la intención en solicitudes claras. Para la empresa, esta métrica se traduce en una reducción directa del tiempo de búsqueda de soluciones operativas.

4.2 Análisis de errores y matriz de confusión

Para efectos de la evaluación de las consultas, la distancia espacial (distance) retornada por ChromaDB se transformó en un índice de similitud mediante la operación $1 - \text{distancia}$, estableciendo así el umbral de aceptación en 0.65.

A pesar de la alta tasa de exactitud, el análisis revela un fenómeno de solapamiento semántico en los requerimientos asociados a problemas de red. La categoría "Sin_Red_WiFi" presentó una alta exhaustividad (recall de 1.00) pero una baja precisión (0.52), atrayendo falsos positivos de categorías como "Conexion_Internet_Celular" y "wifi_windows". En un entorno de producción para UNIDROGAS S.A.S., esto implica que el usuario final con un problema celular podría recibir instrucciones genéricas de WiFi, lo cual generaría fricción en la experiencia y derivaría en la apertura innecesaria de un ticket. Esto demuestra que los manuales de red comparten una densidad terminológica que el umbral actual (0.65) no logra discriminar por completo.

4.3 Limitaciones y oportunidades de mejora

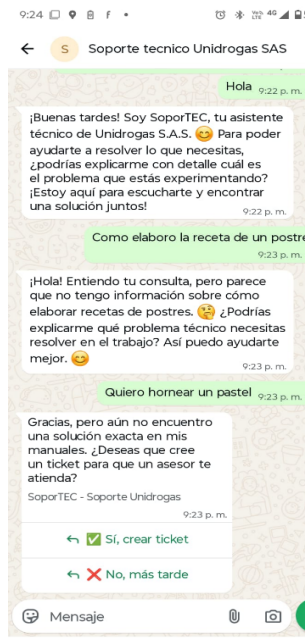
El prototipo actual presenta limitaciones frente a su paso a un entorno de producción. La dependencia de un proveedor externo para la API de WhatsApp introduce un riesgo de latencia de red ajeno al servidor local. Como oportunidad de mejora inmediata, se requiere un refinamiento manual de los metadatos (tags) en los archivos JSON para desambiguar las categorías de red y ajustar dinámicamente los umbrales de búsqueda. Respecto a esto último, la asignación inicial de 0.65 como umbral de similitud se determinó empíricamente tras observar los resultados de prueba, al identificar este valor como el punto de corte donde el sistema rechazaba las solicitudes ajenas al dominio. Frente al solapamiento documentado en las consultas de redes, la ejecución de un barrido de umbrales iterativo constituye una línea de mejora concreta. Este procedimiento permitirá identificar el valor matemático exacto para ajustar el balance entre la precisión y la exhaustividad en las categorías de conectividad, optimizando así la capacidad de discriminación del sistema.

4.4 Resultados de integración con WhatsApp

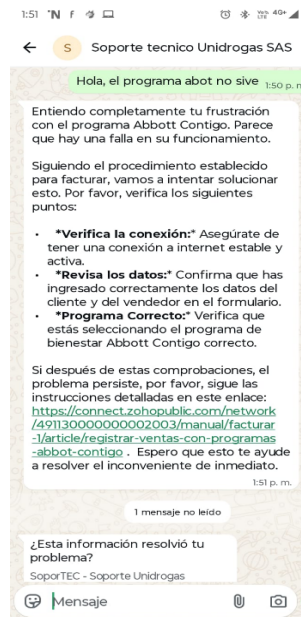
La evaluación demostró que es viable orquestar el flujo desde ChromaDB hacia el modelo Gemma 3 y posteriormente al usuario mediante WhatsApp. Es crucial precisar que el sistema evaluado opera actualmente como un prototipo funcional validado en un ambiente controlado. Si bien el modelo de lenguaje y la base de conocimiento operan sin conexión con terceros en el entorno local, el canal de comunicación vía WhatsApp depende del proveedor externo 1msg y de un *webhook* expuesto a internet. Por lo tanto, la privacidad abarca la gestión interna del conocimiento. Para dotar de evidencia cuantitativa a esta integración, se midió el tiempo de generación del LLM dentro del flujo de atención. A partir de una muestra de 42 consultas

registradas en los registros del servidor de aplicación, el modelo alcanzó un tiempo promedio de procesamiento y redacción de 26 segundos (con picos máximos de hasta 86 segundos en requerimientos de mayor densidad de contexto). Esta métrica aísla el desempeño de la infraestructura local de las posibles fluctuaciones de red atribuidas al proveedor 1msg, demostrando que la inferencia de la IA mantiene un rendimiento operativo viable para el entorno empresarial. Complementando las métricas de recuperación de información, la utilidad del prototipo radica en su capacidad para adaptar el contenido técnico. Una vez el sistema extrae el documento pertinente, el modelo de lenguaje procesa el texto para estructurar una respuesta en formato conversacional. Ejemplos de este flujo de procesamiento a través de la interfaz de WhatsApp, evaluados bajo diferentes escenarios de prueba, se ilustran a continuación en la .

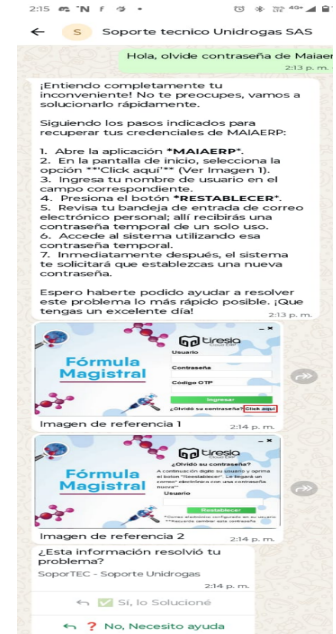
Figura 11. *Capturas de pantalla de conversación en WhatsApp*



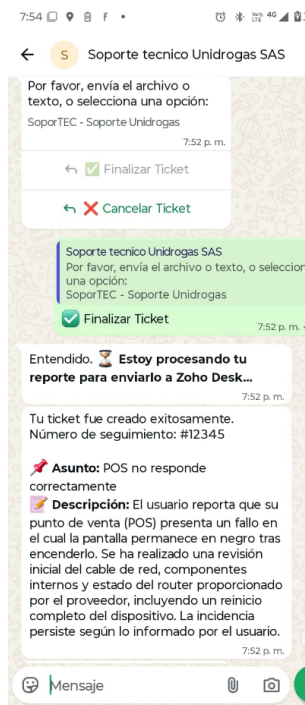
(a)



(b)



(c)



(d)

Nota: (a) Interacción con preguntas fuera del contexto de soporte donde el sistema rechaza consultas no relacionadas y ofrece crear un ticket. (b) Atención a problemas de software

demostrando tolerancia a errores ortográficos en la entrada del usuario. (c) Resolución de fallas enviando contenido multimedia al usuario. (d) Flujo de escalamiento solicitando archivos adjuntos o más detalles para la creación automatizada de un ticket de prueba.

5. Conclusiones

El desarrollo de esta práctica empresarial ratifica la factibilidad de instrumentar herramientas de inteligencia artificial generativa dentro de infraestructuras de hardware local para transformar la gestión de incidencias tecnológicas en UNIDROGAS S.A.S. A través del diseño del sistema automatizado, se estableció un mecanismo eficiente orientado a mitigar la saturación de los canales de atención, proporcionando respuestas oportunas y pertinentes al personal.

Al integrar un modelo de lenguaje optimizado mediante cuantización con un motor de búsqueda vectorial, el sistema demostró una alta eficacia en la recuperación semántica de la información a partir de requerimientos formulados en lenguaje natural. Esta arquitectura no solo facilitó la extracción adecuada de los manuales institucionales, sino que su ejecución bajo una topología on-premise permitió la retención de los datos dentro de los servidores de la compañía.

Los resultados de la evaluación indicaron la correcta clasificación de las intenciones en la mayoría de las categorías, a excepción de las consultas de redes, donde se documentó un solapamiento semántico. El valor de 0.65, definido matemáticamente como umbral de similitud y no de distancia, operó como filtro para descartar las consultas externas al dominio, aunque mantuvo limitaciones para discriminar entre categorías afines. Esta configuración detuvo el procesamiento de respuestas inconsistentes.

Desde el punto de vista funcional, la orquestación de la lógica de procesamiento con la interfaz de mensajería de WhatsApp materializó un canal de soporte accesible y dinámico, el cual

se perfila como una herramienta de fácil adopción para los usuarios en futuras etapas de despliegue y operación.

Por último, el diseño, despliegue y puesta en marcha de este ecosistema tecnológico refleja la integralidad de las competencias adquiridas durante la formación en Ingeniería Mecatrónica. La articulación de lenguajes de programación, la gestión de bases de datos vectoriales, la optimización de recursos computacionales para inferencia neuronal y la integración de servicios de red, evidencia la capacidad de aplicar un enfoque sistémico para dar solución a problemáticas empresariales reales mediante arquitecturas de generación aumentada por recuperación.

6. Recomendaciones

A partir de los resultados obtenidos y del comportamiento del prototipo funcional, se formulan las siguientes recomendaciones:

- *Refinamiento de la base de conocimiento:* Se recomienda realizar una reescritura técnica de los manuales enfocados en conectividad (WiFi, POS, Internet Celular) para dotarlos de palabras clave exclusivas. Esto mitigará el solapamiento semántico y mejorará la precisión de categorías que actualmente presentan falsos positivos.
- *Transición a entorno de producción:* Antes de un despliegue masivo, es indispensable realizar una prueba piloto escalonada con un grupo de usuarios reales. Esto permitirá medir los tiempos de respuesta del hardware frente a concurrencia y validar la interacción del usuario final.
- *Actualización iterativa:* Instituir un protocolo de actualización periódica para el archivo JSON de manuales (ingesta de datos). El sistema debe crecer de forma paralela a los nuevos procedimientos de la empresa.

- *Gestión del canal de mensajería:* Para mitigar la dependencia y los riesgos asociados al uso de proveedores de terceros (como 1msg), se recomienda escalar el prototipo migrando la integración directamente hacia la API oficial de WhatsApp (WhatsApp Cloud API de Meta). Esta transición técnica eliminará intermediarios en el flujo de comunicación, reduciendo los riesgos de latencia de red y fortaleciendo la seguridad y el control sobre los datos compartidos en los chats, manteniendo al mismo tiempo la accesibilidad y familiaridad de la herramienta para los usuarios finales.

Referencias

- [1] R. T. Fielding, “Architectural Styles and the Design of Network-based Software Architectures”, 2000. Consultado: el 24 de mayo de 2026. [En línea]. Disponible en: https://roy.gbiv.com/pubs/dissertation/fielding_dissertation.pdf
- [2] J. Johnson, M. Douze, y H. Jégou, “Billion-scale similarity search with GPUs Matthijs Douze”, 2017, doi: 10.48550/arXiv.1702.08734.
- [3] Y. Gao *et al.*, “Retrieval-Augmented Generation for Large Language Models: A Survey”, *Proceedings - 2024 Conference on AI, Science, Engineering, and Technology, AIXSET 2024*, pp. 166–169, dic. 2023, doi: 10.1109/AIXSET62544.2024.00030.
- [4] T. Dettmers, A. Pagnoni, A. Holtzman, y L. Zettlemoyer, “QLoRA: Efficient Finetuning of Quantized LLMs”, *Adv. Neural Inf. Process. Syst.*, vol. 36, may 2023, Consultado: el 23 de mayo de 2026. [En línea]. Disponible en: <https://arxiv.org/pdf/2305.14314>
- [5] P. Mell y T. Grance, “The NIST Definition of Cloud Computing”, sep. 2011, doi: 10.6028/NIST.SP.800-145.
- [6] Chroma, “Update Data”. Consultado: el 24 de abril de 2026. [En línea]. Disponible en: <https://docs.trychroma.com/docs/collections/update-data>
- [7] P. Lewis *et al.*, “Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks”, *Adv. Neural Inf. Process. Syst.*, vol. 2020-December, may 2020, Consultado: el 24 de mayo de 2026. [En línea]. Disponible en: <https://arxiv.org/pdf/2005.11401>
- [8] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, y J. Dean, “Distributed Representations of Words and Phrases and their Compositionality”, *Adv. Neural Inf. Process. Syst.*, oct. 2013, Consultado: el 23 de mayo de 2026. [En línea]. Disponible en: <https://arxiv.org/pdf/1310.4546>

- [9] J. Liu, J. Lin, y Y. Liu, “How Much Can RAG Help the Reasoning of LLM?”, oct. 2024, Consultado: el 24 de abril de 2026. [En línea]. Disponible en: <https://arxiv.org/pdf/2410.02338>
- [10] “Leyes desde 1992 - Vigencia expresa y control de constitucionalidad [LEY_1581_2012]”. Consultado: el 28 de julio de 2026. [En línea]. Disponible en: http://www.secretariassenado.gov.co/senado/basedoc/ley_1581_2012.html
- [11] “Privacy policy | 1MSG”. Consultado: el 28 de julio de 2026. [En línea]. Disponible en: <https://1msg.io/privacy-policy>
- [12] A. Guío, E. Tamayo, y P. Gómez, “MARCO ÉTICO PARA LA INTELIGENCIA ARTIFICIAL EN COLOMBIA”, 2021, Consultado: el 28 de julio de 2026. [En línea]. Disponible en: <https://minciencias.gov.co/sites/default/files/marco-etico-ia-colombia-2021.pdf>
- [13] Ollama, “Ollama documentation”. Consultado: el 25 de abril de 2026. [En línea]. Disponible en: <https://docs.ollama.com/>
- [14] G. Gerganov, “llama.cpp: Quantize Tool Documentation”. Consultado: el 24 de mayo de 2026. [En línea]. Disponible en: <https://github.com/ggml-org/llama.cpp/blob/master/tools/quantize/README.md>