PONTIFICIA UNIVERSIDAD CATÓLICA DEL ECUADOR

FACULTAD DE HABITAT, INFRAESTRUCTURA Y CREATIVIDAD

INGENIERÍA EN SISTEMAS DE INFORMACIÓN

DESARROLLO DE UN PROTOTIPO DE SISTEMA BASADO EN ARQUITECTURA DE RECUPERACIÓN AUMENTADA POR GENERACIÓN (RAG) PARA LA CONSULTA INTELIGENTE DE LA NORMATIVA ACADÉMICA INSTITUCIONAL

AUTOR

ALEJANDRO MANUEL CHICAIZA NAVARRETE

DIRECTORA

PhD. Susana Masapanta Carrión

QUITO DM, junio 2026

Dedicatoria

Dedico el presente trabajo de titulación a mi madre y a mi padre, que se encuentra en el cielo. De igual forma, a mi hermana junto a mi sobrino, a mis tíos, primos y abuelitos, y a todas las personas que me dieron su apoyo, tanto en lo material como en lo emocional. Estuvieron ahí cuando más lo necesité y me dieron la fuerza para llegar a donde estoy en este momento, con charlas y enseñanzas que jamás olvidaré.

Agradecimiento

Agradezco enormemente a mis padres por su apoyo y aprecio hacia mí, ya que me brindaron la grandiosa oportunidad de mejorar mi formación académica. También agradezco a los amigos que hice durante este camino, quienes se convirtieron en personas cercanas con las que compartí muchos momentos y emociones. Finalmente, agradezco a Dios, que después de tantas circunstancias me permitió seguir en pie, y a mí mismo, por no rendirme y seguir hasta el final.

Resumen

El presente trabajo aborda la dificultad que enfrentan los estudiantes de la Pontificia Universidad Católica del Ecuador (PUCE) para consultar la normativa institucional, distribuida en archivos PDF independientes que carecen de mecanismos de búsqueda dirigida por contenido, lo que genera dependencia del personal de mentorías, coordinaciones y secretaría para resolver dudas reglamentarias. Como solución se diseñó y construyó un prototipo basado en la arquitectura RAG, capaz de responder consultas en lenguaje natural sobre dos reglamentos institucionales con citación obligatoria del artículo y documento de origen.

El desarrollo siguió la metodología CRISP-ML(Q) e integró una estrategia de extracción híbrida de texto nativo y reconocimiento óptico de caracteres, además de contar con segmentación estructural por artículo, recuperación híbrida que combina búsqueda léxica BM25 con búsqueda vectorial densa, un enrutador de intención basado en un modelo de lenguaje y una ruta dedicada para consultas estructurales sobre el corpus.

Así mismo, la validación experimental, ejecutada sobre treinta consultas distribuidas en cinco categorías, confirmó el cumplimiento de los criterios de aceptación definidos: una tasa de acierto documental del 100 %, una tasa de citación correcta del 93,3 % superando el umbral del 90 %, una declaración de ausencia del 100 % ante preguntas fuera de alcance y una latencia media de 4,12 segundos. Se concluye que la arquitectura RAG constituye una solución técnica viable para la consulta inteligente de normativa universitaria, y se identifican las mejoras necesarias para su evolución hacia una implementación institucional de producción.

Palabras clave: Generación Aumentada por Recuperación, modelos de lenguaje, recuperación de información, normativa universitaria, procesamiento de lenguaje natural.

Abstract

This work addresses the difficulty faced by students at the Pontifical Catholic University of Ecuador (PUCE) in consulting institutional regulations, which are distributed across independent PDF files that lack content-based search mechanisms, creating dependence on mentoring, coordination, and registrar staff to resolve regulatory questions. As a solution, a prototype based on the Retrieval-Augmented Generation (RAG) architecture was designed and built, capable of answering natural-language queries about two institutional regulations with mandatory citation of the source article and document.

The development followed the CRISP-ML(Q) methodology and integrated a hybrid text-extraction strategy combining native text and optical character recognition, along with structural segmentation by article, hybrid retrieval combining BM25 lexical search with dense vector search, a language-model-based intent router, and a dedicated route for structural queries about the corpus.

Likewise, the experimental validation, carried out on thirty queries distributed across five categories, confirmed compliance with the defined acceptance criteria: a document accuracy rate of 100 %, a correct citation rate of 93.3 % exceeding the 90 % threshold, an absence-declaration rate of 100 % for out-of-scope questions, and a mean latency of 4.12 seconds. It is concluded that the RAG architecture constitutes a technically viable solution for the intelligent consultation of university regulations, and the improvements necessary for its evolution toward a production-grade institutional deployment are identified.

Keywords: Retrieval-Augmented Generation, language models, information retrieval, university regulations, natural language processing.

Tabla de contenido

Dedicatoria ……I

Agradecimiento..... ……II

Resumen III

Abstract IV

CAPITULO I: INTRODUCCIÓN 1

1.1 **Planteamiento del Problema** 1

1.2 **Justificación** 1

1.3 **Objetivos** 2

1.3.1 Objetivo General 2

1.3.2 Objetivos Específicos 3

1.4 **Alcance** 3

CAPITULO II: FUNDAMENTACIÓN TEÓRICA 5

2.1 **Modelos de Lenguaje de Gran Escala (LLM)** 5

2.2 **Arquitectura RAG** 5

2.3 **Procesamiento de Lenguaje Natural** 5

2.3.1 Embeddings 5

2.3.2 Fragmentación 5

2.3.3 Prompt Engineering 6

2.4 **Bases de Datos Vectoriales** 6

2.5 **Búsqueda Híbrida (BM25 + Vectorial)** 6

2.6 **Metodología CRISP-ML(Q)** 7

2.6.1 Fases de CRISP-ML(Q) 7

2.6.1.1 Comprensión del Negocio y los Datos 7

2.6.1.2 Ingeniería de datos. 7

2.6.1.3 Ingeniería del modelo de ML. 8

2.6.1.4 Evaluación del modelo. 8

2.6.1.5 Despliegue. 8
2.6.1.6 Monitoreo y Mantenimiento. 9
2.7 **Herramientas y tecnologías del sistema** 9
2.7.1 Python 9
2.7.2 LangChain 9
2.7.3 ChromaDB 9
2.7.4 PyMuPDF 10
2.7.5 Pytesseract 10
2.7.6 HuggingFace y el modelo paraphrase-multilingual-MiniLM-L12-v2 10
2.7.7 Groq Cloud 11
2.7.8 Streamlit 11
CAPÍTULO III: ANÁLISIS Y DISEÑO DEL PROTOTIPO 12
3.1 **Análisis de requerimientos** 12
3.2 **Requisitos Funcionales** 12
3.3 Requisitos no Funcionales 14
3.4 **Especificación de requerimientos** 16
3.4.1 Diagramas casos de uso 16
3.5 **Identificación y limpieza de la documentación** 35
3.5.1 Identificación del corpus documental 35
3.5.2 Caracterización de los documentos 36
3.5.3 Estrategia de extracción híbrida 36
3.5.4 Limpieza y normalización del texto 37
3.5.5 Recorte del cuerpo normativo 37
3.5.6 Segmentación por artículos 38
3.5.7 Métricas de calidad de los datos 38
3.6 **Diseño de la arquitectura del sistema** 39
3.6.1 Capas de la arquitectura 39
3.6.2 Flujo del pipeline de indexación 40

3.6.3 Flujo del pipeline de consulta 41
3.6.4 Estrategia de recuperación híbrida y re-ranking 41
3.7 Selección del conjunto tecnológico 42
CAPÍTULO IV: CONSTRUCCIÓN Y VALIDACIÓN DEL PROTOTIPO ... 43
4.1 Preparación del entorno de desarrollo 43
4.1.1 Estándares de codificación 44
4.2 Implementación de la funcionalidad 45
4.2.1 Configuración de la base de datos vectorial 45
4.2.2 Motor de recuperación semántica 47
4.2.3 Interfaz web con separación por rol 51
4.3 Plan de pruebas 53
4.3.1 Tipos de pruebas 53
4.3.1.1 Pruebas funcionales sobre el conjunto de consultas representativas 56
4.3.2 Protocolo experimental detallado 56
4.3.2.1 Preparación del entorno de prueba 57
4.3.2.2 Conjunto completo de las 30 consultas de prueba 57
4.3.2.3 Evaluación de cada respuesta 59
4.3.2.4 Registro de resultados 60
4.3.2.5 Cálculo de métricas agregadas 65
4.3.3 Criterios de aceptación 66
4.3.3.1 Plantilla de resultados experimentales 66
4.3.4 Ejemplos representativos de respuestas 67
4.3.5 Casos límite identificados 70
4.4 Documentación técnica del sistema 71
4.4.1 Estructura del repositorio 71
4.4.2 Variables de entorno y configuración 72
4.4.3 Dependencias principales 72
4.5 Análisis de viabilidad técnica para implementación institucional 73

4.5.1 Estimación de la carga del sistema 73
4.5.2 Distribución de la carga computacional por componente 74
4.5.3 Escenario viable con la infraestructura existente 75
4.5.4 Escenario óptimo hipotético 76
4.5.5 Análisis de privacidad 78
4.5.6 Escalabilidad del corpus documental 78
CAPÍTULO V: CONCLUSIONES Y RECOMENDACIONES 80
5.1 Conclusiones 80
5.2 Recomendaciones 81
Anexos 86
REFERENCIAS BIBLIOGRÁFICAS 87

ÍNDICE DE GRÁFICOS

ILUSTRACIÓN 1 CASO DE USO REQUERIMIENTOS GENERALES: CU01.....17

ILUSTRACIÓN 2 CARGA Y PROCESAMIENTO DE REGLAMENTOS: CU-01.....20

ILUSTRACIÓN 3 CASOS DE USO DEL ESTUDIANTE.....29

ILUSTRACIÓN 4 CASOS DE USO DE GESTIÓN Y DIAGNÓSTICO.....35

ILUSTRACIÓN 5 REPRESENTACIÓN DEL ALGORITMO HNSW CON SUS CAPAS JERÁRQUICAS Y PUNTO DE ENTRADA.....47

ÍNDICE DE TABLAS

TABLA 1 CASO DE USO: CARGAR Y PROCESAR REGLAMENTOS (CU-01) *17*

TABLA 2 CASO DE USO: CONSULTAR ARTÍCULO ESPECÍFICO (CU-02)) *20*

TABLA 3 CASO DE USO: CONSULTAR POR TEMA (CU-03) *23*

TABLA 4 CASO DE USO: CONSULTAR ESTRUCTURA DEL CORPUS (CU-04) *25*

TABLA 5 CASO DE USO: COMPARAR ENTRE REGLAMENTOS (CU-05) *26*

TABLA 6 CASO DE USO: REEMPLAZAR EL CORPUS DOCUMENTAL (CU-06) *29*

TABLA 7 CASO DE USO: DIAGNOSTICAR EL SISTEMA (CU-07) *31*

TABLA 8 CASO DE USO: AUTENTICAR COMO ADMINISTRADOR (CU-08) *33*

TABLA 9 COMPARACIÓN DE LOS REGLAMENTOS DEL CORPUS *36*

TABLA 10 ARQUITECTURA POR CAPAS DEL PROTOTIPO *39*

TABLA 11 SELECCIÓN TECNOLÓGICA Y ALTERNATIVAS EVALUADAS *42*

TABLA 12 DISTRIBUCIÓN DE CONSULTAS DEL PROTOCOLO DE PRUEBAS POR CATEGORÍA *56*

TABLA 13 CONSULTAS DE LA CATEGORÍA A: CONSULTA DIRECTA DE ARTÍCULO *57*

TABLA 14 CONSULTAS DE LA CATEGORÍA B: CONSULTA TEMÁTICA *58*

TABLA 15 CONSULTAS DE LA CATEGORÍA C: META-PREGUNTAS *58*

TABLA 17 CONSULTAS DE LA CATEGORÍA E: CONSULTAS COMPARATIVAS *59*

TABLA 18 CRITERIOS DE EVALUACIÓN DEL PROTOCOLO DE PRUEBAS *59*

TABLA 19 RESULTADOS DE LA CATEGORÍA A: CONSULTA DIRECTA DE ARTÍCULO *60*

TABLA 20 RESULTADOS DE LA CATEGORÍA B: CONSULTA TEMÁTICA *61*

TABLA 21 RESULTADOS DE LA CATEGORÍA C: META-PREGUNTAS *63*

TABLA 22 RESULTADOS DE LA CATEGORÍA D: PREGUNTAS FUERA DE ALCANCE *64*

TABLA 23 RESULTADOS DE LA CATEGORÍA E: CONSULTAS COMPARATIVAS *65*

TABLA 24 RESUMEN DE RESULTADOS POR CATEGORÍA *66*

TABLA 25 DEPENDENCIAS Y VERSIONES DEL PROTOTIPO *72*

TABLA 26 MÉTRICAS ESTIMADAS DE USO INSTITUCIONAL *73*

TABLA 27 UBICACIÓN DEL PROCESAMIENTO POR COMPONENTE *74*

TABLA 28 RECURSOS DE HARDWARE RECOMENDADOS *76*

TABLA 29 COMPARACIÓN ENTRE ESCENARIO VIABLE Y ESCENARIO ÓPTIMO *77*

TABLA 30 IMPACTO DEL CRECIMIENTO DEL CORPUS Y ACCIONES REQUERIDAS *79*

CAPITULO I: INTRODUCCIÓN

1.1 Planteamiento del Problema

La PUCE cuenta con reglamentos académicos vigentes que regulan aspectos fundamentales de la vida estudiantil: el Reglamento General de Estudiantes y el Reglamento de Acceso a Becas y Ayudas Económicas. Estos documentos son de acceso interno a través de la Gaceta Institucional, no obstante, están dispersos entre múltiples archivos PDF independientes, los cuales, dificultan la localización de información específica por parte del estudiante.

Esta situación genera una dependencia directa del estudiante hacia el personal de Mentorías, Coordinaciones de Carrera y Secretaría Unificada, a fin de resolver dudas que, en muchos casos, se podrían responder consultando el reglamento correspondiente.

Pregunta principal: ¿Cómo el diseño de un sistema basado en arquitectura RAG puede mejorar la precisión y eficiencia en la consulta de reglamentos académicos institucionales de la PUCE por parte de los estudiantes?

1.2 Justificación

Ante la ausencia de herramientas institucionales especializadas, una práctica frecuente entre estudiantes universitarios es recurrir a modelos de lenguaje generativos de propósito general como ChatGPT para resolver dudas sobre reglamentos. Este comportamiento introduce un riesgo medible teniendo que, los “LLMs sin anclaje documental presentan tasas de alucinación que pueden superar el 6%’’ (Nishisako et al., 2025). Por ende, incluso en modelos avanzados que generan respuestas que parecen correctas, pueden no reflejar los reglamentos reales de la institución. Así mismo, en el contexto académico, una respuesta incorrecta sobre requisitos de becas o distintas condiciones que debería conocer el estudiante podría dar paso a decisiones perjudiciales para el mismo.

Por otra parte, la experiencia de otras instituciones confirma tanto el nivel del problema como la viabilidad de la solución, tal es el caso de la Universidad Virtual del Estado de Guanajuato, la cual implementó un asistente inteligente orientado a optimizar

la experiencia de sus estudiantes frente a consultas frecuentes de información institucional, reduciendo en un 60% la necesidad de los estudiantes de acudir directamente a canales administrativos para obtener respuestas (UVEG, 2024).

De igual forma, la Institución Universitaria Salazar y Herrera (IUSH) de Colombia desarrolló un chatbot basado en inteligencia artificial orientado específicamente al reglamento académico estudiantil, con el objetivo de hacer de este documento un recurso accesible y comprensible para los estudiantes, mejorando la comunicación entre estos y la universidad sin necesidad de intermediarios administrativos (IUSH, 2024).

Adicionalmente, Santibáñez et al. (2025) implementaron y evaluaron un sistema basado en arquitectura RAG para la consulta de legislaciones y reglamentos universitarios en la Universidad Tecnológica de la Mixteca, concluyendo que los resultados de las consultas realizadas a través del sistema facilitan el acceso a la normativa y optimizan el tiempo de consulta, otorgando un entorno más eficiente y con mejor acceso a la información.

Estos casos evidencian que la demanda de consultas reglamentarias en instituciones universitarias es alta, repetitiva, y que los estudiantes son sus principales generadores.

Frente a estos escenarios, la arquitectura RAG es la solución técnica más adecuada, al restringir las respuestas del modelo exclusivamente a los documentos oficiales vigentes, eliminando el riesgo de información medianamente fiable y garantizado trazabilidad, citando el artículo exacto que respalda cada respuesta. A diferencia de un chatbot convencional, el sistema no depende de conocimiento preentrenado, sino de los documentos reales y actualizados que la propia institución pública.

1.3 **Objetivos**

1.3.1 Objetivo General

Desarrollar un prototipo de sistema basado en arquitectura RAG para automatizar la consulta de ciertos reglamentos académicos, integrando un modelo de lenguaje y un modelo de *embeddings* con una base de datos vectorial para mejorar la interacción informativa entre la universidad y los estudiantes.

1.3.2 Objetivos Específicos

1. Identificar y procesar los reglamentos académicos vigentes de la PUCE para su transformación en formatos legibles por el sistema.
2. Diseñar la arquitectura técnica del sistema, incluyendo la selección de la base de datos vectorial y la integración del modelo de lenguaje (LLM).
3. Realizar pruebas de validación para asegurar que las respuestas del sistema sean precisas y provengan directamente de las fuentes oficiales.

1.4 **Alcance**

El propósito principal del proyecto se centra en el desarrollo de un prototipo funcional de software que aplique la arquitectura RAG para la gestión de consultas sobre el Reglamento General de Estudiantes y el Reglamento de Acceso a Becas y Ayudas Económicas de la PUCE. En base a lo propuesto se tiene que su población objetivo es principalmente estudiantes de pregrado de la PUCE, aunque su arquitectura permite la consulta por parte de personal administrativo y docente, además el prototipo se alimentará exclusivamente de documentos oficiales en formato PDF.

De igual forma, contendrá las siguientes funcionalidades técnicas junto con sus respectivas limitaciones.

Funcionalidades técnicas

1. Carga y segmentación automática de documentos PDF institucionales.
2. Implementación de una base de datos vectorial para la recuperación de fragmentos de texto basada en el contexto de la pregunta.
3. Desarrollo de una aplicación web que permita la interacción en lenguaje natural y proporcione respuestas citando el fragmento del reglamento utilizado.
4. El sistema será desplegado como un prototipo en un entorno controlado para demostración técnica.
5. Control de acceso por rol de usuario: El sistema presentará dos vistas diferenciadas dentro de una misma aplicación.

La vista del estudiante mostrará exclusivamente la interfaz de chat para la formulación de consultas en lenguaje natural. La vista del administrador tendrá, adicionalmente, los controles de carga y reemplazo de reglamentos, el panel de diagnóstico del sistema y las métricas de indexación. Esta separación garantiza que el usuario final interactúe con una interfaz limpia y enfocada en su caso de uso, sin exposición a controles técnicos que no le corresponden.

Limitaciones

1. El sistema es una herramienta de apoyo informativo; las respuestas generadas no poseen validez legal ni sustituyen la resolución oficial de una secretaría unificada.
2. El alcance no incluye la ejecución de trámites (ej. postulación automática a becas), solo la consulta de los requisitos para realizarlos.
3. No se implementará firma electrónica ni generación de certificados oficiales.
4. Dado que los reglamentos institucionales consultados son documentos de acceso público, la transmisión de fragmentos textuales al servicio de inferencia Groq Cloud no compromete información confidencial. Sin embargo, las consultas formuladas por los estudiantes podrían contener contexto personal sensible. Por ende, en una versión de producción institucional se recomienda evaluar el despliegue de un modelo de inferencia local mediante vLLM o Ollama, eliminando la dependencia de servicios externos y garantizando que ninguna consulta abandone la infraestructura de la PUCE.

CAPITULO II: FUNDAMENTACIÓN TEÓRICA

2.1 Modelos de Lenguaje de Gran Escala (LLM)

Según de Zárate et al. (2024), los “LLM son modelos de procesamiento de lenguaje natural que, en la actualidad, adoptan ampliamente la arquitectura de transformadores generativos pre-entrenados, diseñados para interpretar y responder preguntas de manera cada vez más sofisticada” (p.6-7). Teniendo estos una alta relevancia para el prototipo ya que LLM funcionará como el cerebro que procesa los reglamentos recuperados, con el fin de redactar una respuesta comprensible para el estudiante.

2.2 Arquitectura RAG

De acuerdo con Suárez (2024), la Generación Aumentada por Recuperación se define como una "tecnología híbrida de IA que combina la recuperación de información con la generación de texto para ofrecer respuestas precisas a las consultas de los usuarios" (p. 22), estableciendo una arquitectura técnica que garantiza que la respuesta entregada al usuario pase por un filtro de verificación previa analizando los reglamentos legales de la institución, evitando así alucinaciones.

2.3 Procesamiento de Lenguaje Natural

2.3.1 Embeddings

medio para representar objetos como texto, imágenes y audio como puntos en un espacio vectorial continuo donde las ubicaciones de esos puntos en el espacio son semánticamente significativas para los algoritmos de *machine learning* (ML)", esto significa que los reglamentos de la universidad dejarán de ser vista por el sistema como simples cadenas de caracteres para convertirse en coordenadas matemáticas.

2.3.2 Fragmentación

Es un método que permite dividir grandes cantidades de datos en porciones más pequeñas, con el fin, de que el modelo de IA pueda buscar y recuperar información específica con mayor eficiencia y precisión. En este proyecto este método es de gran

utilidad ya que provocará que el modelo no pierda contexto como lo hacen los grandes lenguajes al procesar gran cantidad de información debido a que su ventana de contexto es limitada (Guerrero Sarmiento, 2025).

2.3.3 Prompt Engineering

Según Díaz Sifuentes et al. (2025), este concepto se define como un "enfoque innovador que consistía en diseñar instrucciones específicas para guiar las respuestas de los modelos, mejorando su precisión y relevancia sin modificar sus parámetros internos" (p. 6), por lo que, para el prototipo se diseñarán directrices que obliguen a la IA a actuar como un asistente experto en los reglamentos de estudiantes de la PUCE. Por lo cual, esta técnica es de gran utilidad para restringir las respuestas al contexto de los reglamentos recuperados, asegurando que la información sea correcta.

2.4 **Bases de Datos Vectoriales**

Según Krantz et al. (2026), en estos sistemas:

Los puntos de datos se almacenan como matrices de números llamados ‘vectores’, que se pueden comparar y agrupar en función de la similitud, permitiendo además consultas de baja latencia ideales para aplicaciones de Inteligencia Artificial (IA), a diferencia de las bases de datos relacionales que dependen de coincidencias exactas de palabras clave.

2.5 **Búsqueda Híbrida (BM25 + Vectorial)**

Las arquitecturas RAG modernas reconocen que la búsqueda exclusivamente vectorial presenta limitaciones cuando la consulta del usuario contiene términos exactos, identificadores numéricos o nombres propios poco frecuentes, casos donde la igualdad semántica puede subestimar la importancia léxica. Por ende, para disminuir este efecto, se incorpora una estrategia de búsqueda híbrida que combina dos paradigmas complementarios: la búsqueda vectorial densa, basada en *embeddings*, y la búsqueda léxica dispersa, basada en el algoritmo BM25 (Robertson y Zaragoza, 2009). En el contexto del prototipo, esta combinación resulta esencial debido a que las consultas estudiantiles frecuentemente referencian numerales de artículos específicos (por ejemplo, “Art. 29”) y términos institucionales propios.

2.6 **Metodología CRISP-ML(Q)**

Según Studer et al. (2021), el modelo CRISP-ML(Q) es “un modelo de proceso para aplicaciones de aprendizaje automático con metodología de garantía de calidad, que ayuda a las organizaciones a aumentar la eficiencia y la tasa de éxito en sus proyectos de aprendizaje automático” (p. 14), es decir, que el desarrollo del prototipo no será un proceso lineal, sino iterativo, garantizando que cada fase, desde la recolección de los reglamentos hasta la validación de las respuestas, cumpla con estándares técnicos de confiabilidad.

2.6.1 Fases de CRISP-ML(Q)

2.6.1.1 Comprensión del Negocio y los Datos

De acuerdo con Studer et al. (2021), esta primera fase combina la comprensión del negocio y de los datos, ya que la disponibilidad de los datos afecta directamente la viabilidad del proyecto. Además, en esta etapa se definen los objetivos del sistema, los criterios de éxito medibles a través de KPIs, y las restricciones bajo las que debe operar la solución, por lo que, para el presente prototipo, esta fase comprende la identificación del problema central, así como el análisis de los documentos reglamentarios disponibles, evaluando su formato, extensión y calidad textual.

2.6.1.2 Ingeniería de datos.

Según Studer et al. (2021), esta fase abarca la selección, limpieza, ingeniería de características y estandarización de los datos, con el objetivo de prepararlos para la etapa de modelado. En el contexto de este prototipo, la ingeniería de datos abarca la extracción del contenido textual de los reglamentos en formato PDF, la aplicación de OCR sobre aquellos documentos que se encuentren en formato de imagen escaneada, y la fragmentación del texto mediante segmentación estructural por artículo, de modo que cada fragmento conserva una unidad normativa completa, aplicando subdivisión con solapamiento únicamente en los artículos más extensos. Este proceso garantiza que la información ingresada al sistema sea coherente, estructurada y libre de ruido que pueda afectar la precisión de las respuestas generadas.

2.6.1.3 Ingeniería del modelo de ML.

En esta fase se propone la selección y ajuste del modelo que mejor equilibre velocidad y precisión, considerando los hiperparámetros adecuados para el caso de uso. Teniendo que, para este prototipo, la ingeniería del modelo implica la configuración del *pipeline* RAG mediante LangChain, la selección del modelo de lenguaje, Groq Cloud con Llama 3.3 70B Versatile como motor de inferencia en la nube, el diseño del *prompt* que instruye al modelo a responder únicamente con base en los reglamentos recuperados, y la definición del número de *chunks* a recuperar por consulta. Cada una de estas decisiones técnicas afecta directamente en la calidad y trazabilidad de las respuestas entregadas al estudiante (Studer et al., 2021).

2.6.1.4 Evaluación del modelo.

Studer et al. (2021) establecen que esta fase determina si el modelo es confiable ante datos nunca vistos, evaluando si su comportamiento se mantiene dentro de los criterios de éxito definidos en la primera fase. En este prototipo, la evaluación consiste en someter al sistema a un conjunto de preguntas reales sobre los reglamentos cargados, verificando que las respuestas sean precisas, estén respaldadas por el artículo citado y que, ante preguntas fuera del alcance documental, el sistema indique explícitamente que no dispone de información suficiente en lugar de generar una respuesta inventada. Esta última condición es especialmente crítica para garantizar la veracidad de la información entregada al estudiante.

2.6.1.5 Despliegue.

El despliegue implica exponer la funcionalidad del modelo en un entorno de producción, lo que puede tomar la forma de dashboards interactivos, componentes integrados en arquitecturas de software o *endpoints* de servicios web. Puesto que el presente trabajo corresponde a un prototipo, el despliegue se realiza de forma limitada con el propósito de validar el funcionamiento del sistema en un entorno controlado, sin implicar una integración institucional formal con la PUCE, evitando así comprometer la validez técnica de los resultados obtenidos (Studer et al.,2021).

2.6.1.6 Monitoreo y Mantenimiento.

Studer et al. (2021) otorgan especial atención a esta última fase, señalando que un modelo en producción opera en entornos cambiantes que pueden degradar su desempeño con el tiempo, fenómeno conocido como *data drift.* Esta fase excede el alcance del presente prototipo, no obstante, se reconoce su importancia para una eventual implementación institucional debido a que, en ese escenario, el monitoreo implicaría detectar actualizaciones en los reglamentos oficiales de la PUCE para mantener la base documental vigente, garantizando que el sistema no responda con base en normativa desactualizada.

2.7 Herramientas y tecnologías del sistema

2.7.1 Python

Es un lenguaje de programación de alto nivel interpretado, además es el lenguaje base para varios proyectos relacionados con inteligencia artificial y aprendizaje automático. En este proyecto Python va a ser el que integre tanto el procesamiento de documentos como la gestión de la base de datos vectorial y la interfaz de usuario en un único *pipeline* funcional.

2.7.2 LangChain

"LangChain es un marco de trabajo para desarrollar aplicaciones que utilizan modelos de lenguaje extensos, y su objetivo es permitir a los desarrolladores utilizar cómodamente otras fuentes de datos e interactuar con otras aplicaciones" (robal & Akinci, 2023, p.2). Siendo de gran importancia para el prototipo porque será el coordinador del flujo completo, empezando desde la recepción de la pregunta del estudiante hasta la entrega de la respuesta con su cita reglamentaria que le corresponda.

2.7.3 ChromaDB

Es una base de datos vectorial de código abierto optimizada para aplicaciones de inteligencia artificial. De igual manera, ChromaDB permite almacenar, indexar y consultar *embeddings* de forma eficiente, con soporte para búsqueda por similitud

semántica. En este sistema, ChromaDB va a almacenar los fragmentos de los reglamentos de la PUCE como vectores numéricos, permitiendo recuperar los artículos más relevantes ante cada consulta del estudiante.

2.7.4 PyMuPDF

PyMuPDF, cuyo módulo de importación en Python se llama fitz, es una biblioteca que permite la extracción y manipulación de documentos en formato PDF, de igual forma, ayuda en la obtención de texto, imágenes y metadatos guardando la estructura original del documento. En el presente prototipo cumple dos funciones que se complementan: por un lado, extrae el contenido textual de los reglamentos que cuentan con texto digitalmente incrustado mediante el método get_text("blocks"), el cual respeta la agrupación visual por párrafos y bloques, por otro, rasteriza las páginas escaneadas a una resolución de 300 DPI mediante get_pixmap(), preparándolas para su procesamiento por el motor OCR.

2.7.5 Pytesseract

Según Smith (2007) Tesseract es uno de los motores OCR de propósito general con mayor precisión disponibles para texto impreso multilingüe. Por tanto, En el caso de que los documentos reglamentarios se encuentren en formato de imagen escaneada, se utiliza Pytesseract, un envoltorio Python sobre el motor Tesseract OCR de código abierto.

Para alimentar al motor OCR se debe realizar una rasterización previa de cada página, que se hará a través de PyMuPDF, evitando dependencias externas adicionales.

2.7.6 HuggingFace y el modelo paraphrase-multilingual-MiniLM-L12-v2

HuggingFace es una plataforma que centraliza la distribución de modelos preentrenados de procesamiento de lenguaje natural. mediante la integración langchain-huggingface, el prototipo carga el modelo sentence-*transformers*/paraphrase-multilingual-MiniLM-L12-v2, una red neuronal optimizada para tareas de similitud semántica multilingüe. Este modelo fue seleccionado por tres razones técnicas: opera sobre 50 idiomas, incluido el español, lo que asegura precisión en la representación de los reglamentos institucionales, presenta un tamaño compacto que permite ejecutarlo sobre CPU sin requerir aceleración por

GPU y produce vectores de 384 dimensiones, balance adecuado entre una fiabilidad semántica y costo computacional de almacenamiento e indexación en ChromaDB (Reimers y Gurevych, 2019).

2.7.7 Groq Cloud

La LPU es hardware diseñado específicamente para la inferencia ya que preserva la calidad al tiempo que elimina los cuellos de botella arquitectónicos que generan latencia en primera instancia (Groq Inc., 2025). Por lo que, se tiene que Groq es una plataforma de inferencia en la nube, la cual su arquitectura de hardware fue diseñada desde su origen para la inferencia de modelos de lenguaje, a diferencia de las unidades de procesamiento gráfico (GPU), diseñadas para el renderizado de imágenes y después adaptadas a tareas de inteligencia artificial. En el presente prototipo, Groq se integra mediante el adaptador langchain-groq para ejecutar el modelo Llama 3.3 70B Versatile, encargado de generar las respuestas finales a partir del contexto reglamentario recuperado por el sistema.

2.7.8 Streamlit

Es un *framework* de Python que facilita el desarrollo de aplicaciones web orientadas a datos e inteligencia artificial, teniendo como principal ventaja, el permitir construir interfaces interactivas con pocas líneas de código. En este proyecto, Streamlit conforma la capa de presentación del prototipo, ofreciendo al estudiante una interfaz de chat donde puede formular sus preguntas en lenguaje natural y recibir respuestas con trazabilidad reglamentaria.

Adicionalmente, Streamlit permite implementar **r**enderizado condicional basado en estado de sesión, lo que permite construir distintas vistas de la interfaz dentro de una misma aplicación según el rol del usuario autenticado, sin la necesidad de desplegar aplicaciones independientes.

CAPÍTULO III: ANÁLISIS Y DISEÑO DEL PROTOTIPO

El presente capítulo expone el análisis y diseño técnico del prototipo desarrollado, siguiendo las dos primeras fases de la metodología CRISP-ML(Q): la comprensión del negocio y los datos y la ingeniería de datos (Studer et al., 2021). En estas fases se traduce la problemática institucional vista en el Capítulo I a un conjunto de requisitos verificables, se caracteriza el corpus documental disponible, se diseña la arquitectura general del sistema y se justifica la selección de cada componente tecnológico.

3.1 Análisis de requerimientos

Un análisis de requerimientos consiste en la identificación de las necesidades del usuario y la determinación de los mecanismos tecnológicos adecuados para atenderlas. En este caso, los estudiantes de la Pontificia Universidad Católica del Ecuador enfrentan una limitación práctica al consultar la normativa institucional: los reglamentos se publican en formato PDF y carecen de mecanismos de búsqueda inteligente que permitan localizar una disposición específica de forma directa.

Como parte de la fase de comprensión del negocio, la ingeniería de requisitos define el alcance funcional del sistema y los criterios de éxito medibles. A continuación, se detallan los requisitos funcionales y los requisitos no funcionales identificados a partir del análisis del prototipo.

3.2 Requisitos Funcionales

- Carga de reglamentos y procesamiento automático

El sistema debe permitir al administrador cargar uno o varios reglamentos en formato PDF y procesar su contenido de forma automática. La extracción del texto debe realizarse mediante una estrategia híbrida, ya sea la extracción de texto nativo cuando el documento lo permite o por reconocimiento óptico de caracteres (OCR) cuando se trata de páginas escaneadas, con el fin de segmentarlo automáticamente por artículos, conservando como metadatos el número de artículo, su título y el documento de origen.

- Indexación dual: vectorial y léxica

Para garantizar la recuperación efectiva de diferentes consultas, el sistema debe construir dos índices que se complementen sobre los mismos fragmentos, primero un índice vectorial denso, basado en *embeddings* generados con el modelo paraphrase-multilingual-MiniLM-L12-v2, persistido en ChromaDB; y segundo un índice léxico disperso, construido en memoria mediante el algoritmo BM25 a través de la biblioteca rank-bm25. Esta dualidad permite cubrir tanto consultas semánticas como consultas con términos institucionales específicos o numerales de artículo.

- Consulta en lenguaje natural con citación obligatoria

El sistema debe permitir al estudiante formular preguntas en español mediante una interfaz de chat conversacional. Las respuestas generadas deben provenir exclusivamente del corpus indexado y deben incluir, en formato [Art. N — Título — documento], la cita textual del artículo o artículos consultados. Cuando la información no esté disponible en el corpus, el sistema debe declarar explícitamente la ausencia de información en lugar de fabricar contenido.

- Enrutamiento de intención y búsqueda híbrida

El sistema debe clasificar automáticamente la intención de cada consulta mediante un *router* basado en el modelo de lenguaje, distinguiendo entre consultas por artículo específico, consultas temáticas, consultas comparativas, saludos y meta-preguntas. Según el tipo de consulta, debe activar la estrategia de recuperación más adecuada entre las cuales están: la coincidencia exacta por número de artículo, búsqueda léxica BM25, búsqueda vectorial semántica o una combinación ponderada de las tres.

- Manejo de meta-preguntas sobre la estructura del corpus

El sistema debe atender consultas sobre la estructura del corpus tales como “cuántos artículos tiene el Reglamento de Acceso a Becas y Ayudas Económicas” o “lista los artículos del Reglamento General de Estudiantes” mediante el acceso directo a la metadata indexada, sin la necesidad de invocar el *pipeline* de recuperación generativa. Esto permite responder consultas estructurales con precisión absoluta, independientemente del valor de *top-K* de los algoritmos de recuperación.

- Memoria conversacional

El sistema debe mantener el historial de la conversación durante la sesión y reformular automáticamente las preguntas referenciales (por ejemplo, "y el siguiente", "explícame más") en función de los mensajes previos. Así mismo, la memoria conversacional trabaja sobre los seis mensajes más recientes equivalentes a tres turnos de diálogo, valor que equilibra la coherencia contextual con el uso eficiente de la ventana de contexto del modelo generativo. Por otra parte, al momento de cerrar la sesión o refrescar la página, el historial se pierde.

- Gestión administrativa del corpus y diagnóstico técnico

El sistema debe ofrecer al administrador una vista diferente con controles que le permitan la carga inicial de reglamentos, el reemplazo completo del corpus, un panel de diagnóstico con cantidad de fragmentos indexados y la opción de reindexación, siendo que, estos controles no deben estar a la vista del estudiante.

- Control de acceso por rol

El sistema debe diferenciar dos vistas de interfaz según el rol del usuario: la vista del estudiante expone exclusivamente el chat conversacional y estadísticas básicas del corpus, por otra parte, la vista del administrador que incluye todas las opciones previamente vistas con anterioridad. El acceso a la vista de administrador se protege mediante contraseña gestionada por variable de entorno.

3.3 Requisitos no Funcionales

- Precisión documental

Las respuestas generadas deben provenir exclusivamente de los fragmentos recuperados del corpus institucional ya que cuando los fragmentos no contengan información suficiente para responder la consulta, el sistema debe declarar explícitamente la ausencia de información, evitando la fabricación de contenido (alucinaciones).

- Trazabilidad de las respuestas

Toda afirmación contenida en la respuesta debe ir acompañada de la cita del artículo y documento de origen, en el formato [Art. N — Título — documento]. Esta cita debe permitir al estudiante verificar manualmente la información contra el reglamento original.

- Latencia controlada

El tiempo de respuesta debe mantenerse por debajo de los diez segundos en consultas estándar, bajo la conexión a internet residencial. Esto garantiza una experiencia conversacional fluida para el estudiante.

- Seguridad de credenciales

Las claves de acceso a servicios externos (Groq, HuggingFace) y la contraseña de administrador deben gestionarse mediante variables de entorno (.env) y no exponerse en el código fuente. El archivo .env debe ser excluido del control de versiones.

- Usabilidad

La interfaz del estudiante debe permitir el uso del sistema sin conocimientos técnicos previos.

El flujo consulta–respuesta debe completarse sin configuración adicional, manteniendo así una experiencia conversacional natural.

- Escalabilidad documental

La arquitectura debe permitir agregar reglamentos adicionales sin la necesidad de modificaciones estructurales del código. Tanto ChromaDB como el índice BM25 deben escalar adecuadamente al menos hasta diez reglamentos sin una baja notable del rendimiento.

3.4 **Especificación de requerimientos**

3.4.1 Diagramas casos de uso

Un diagrama de caso de uso es una forma de plasmar en un diagrama la interacción que tiene los usuarios (actores) con la funcionalidad del sistema, por tanto, se toma en cuenta las funcionalidades sin entrar mucho en detalles sobre la implementación técnica.

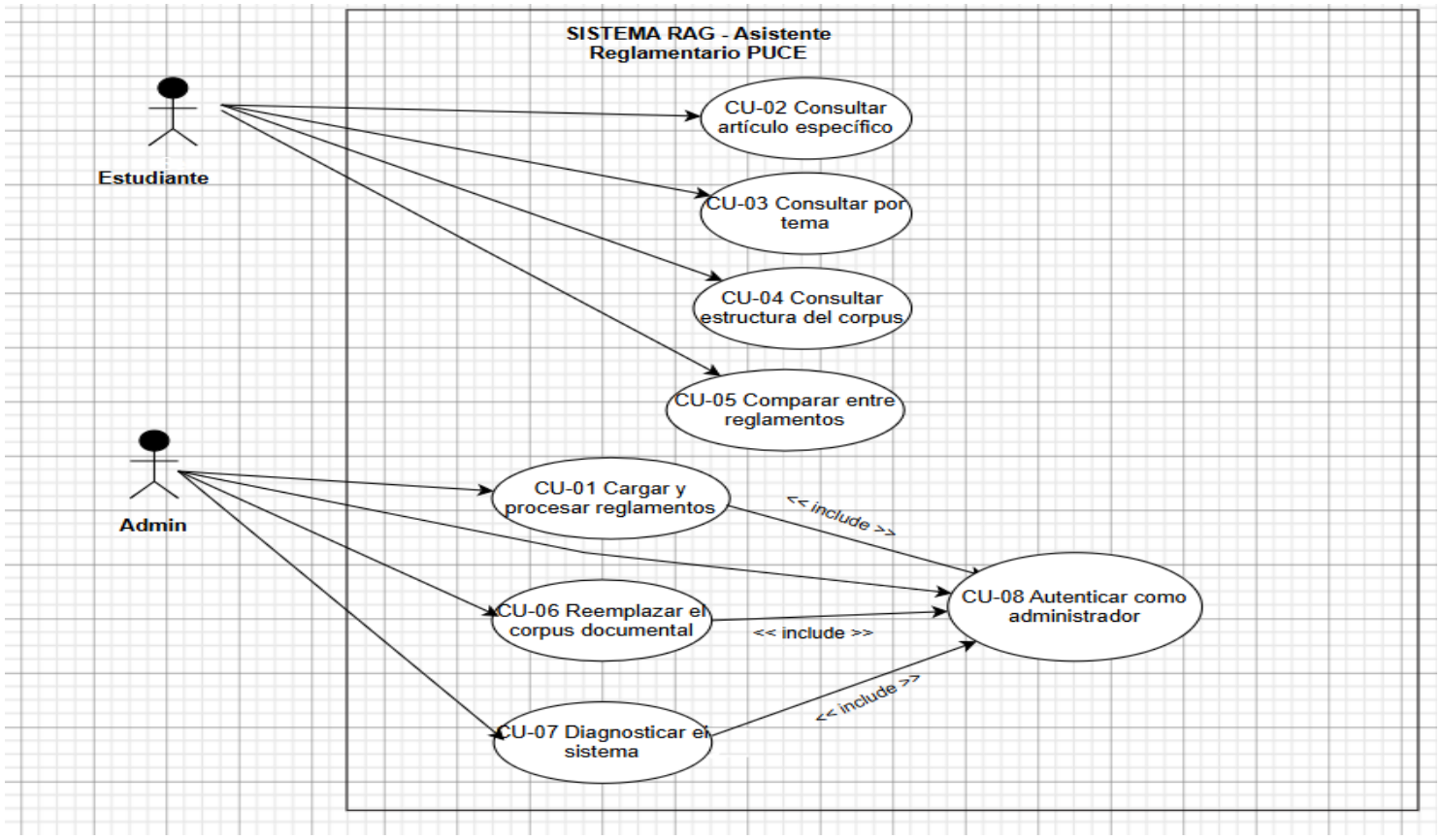
El sistema involucra dos actores con responsabilidades y vistas de interfaz diferentes. El primero es el estudiante, usuario final del prototipo, quien accede a la interfaz de consulta conversacional sin necesidad de autenticación. El segundo es el administrador del sistema, el cual, es responsable de la carga, reemplazo y diagnóstico del corpus documental, cuyo acceso a los controles de gestión está protegido mediante contraseña gestionada por variable de entorno.

ID: CU-01

Nombre: Requerimientos Generales del Sistema

Como se puede ver en la **Ilustración 1**, se presenta la interacción total de los usuarios con el sistema, distinguiendo las funciones del estudiante de las del administrador.

ILUSTRACIÓN 1 CASO DE USO REQUERIMIENTOS GENERALES: CU01



Nota: Elaboración propia. Diagrama caso de uso requerimientos generales del prototipo.

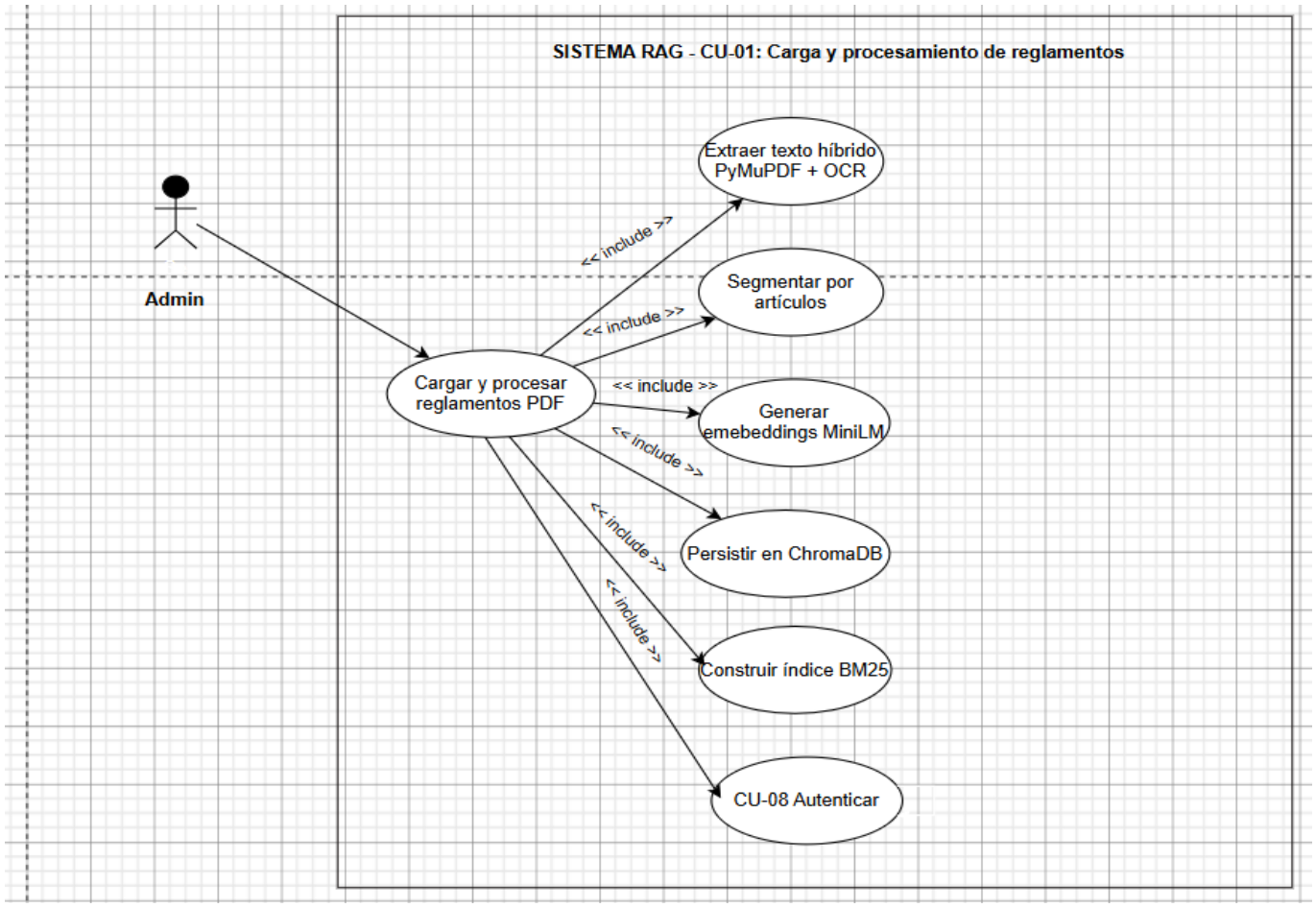
TABLA 1 CASO DE USO: CARGAR Y PROCESAR REGLAMENTOS (CU-01)

Nombre del caso de uso: Cargar y procesar reglamentos	**ID Único:** CU-01
Área: -	
Actor(es): Administrador del sistema	
Interesados: Estudiantes, Administrador.	
Nivel: Primario	
Descripción: Este caso de uso describe el proceso completo en donde se adjunta un nuevo reglamento institucional al sistema. El administrador carga uno o varios archivos PDF, haciendo que el sistema extraiga automáticamente su contenido textual aplicando OCR sobre las páginas escaneadas cuando es necesario, además segmenta el texto por artículos, genera *embeddings* vectoriales y los persiste en ChromaDB; paralelamente construye en memoria un índice BM25 que se reconstruye automáticamente al iniciar cada sesión. Al finalizar, el corpus queda disponible para que los estudiantes formulen consultas.	

Evento desencadenador: El administrador, tras autenticarse correctamente, accede al panel de administración y selecciona uno o más archivos PDF mediante el componente de carga.	
Tipo de desencadenador: ☑ Externo ☐ Temporal	
Pasos realizados (ruta principal)	**Información para los pasos**
1. El administrador se autentica mediante contraseña configurada en variable de entorno.	Contraseña de administrador (ADMIN_PASSWORD).
2. El administrador selecciona uno o varios archivos PDF a través del componente st.file_uploader.	Archivos PDF de reglamentos.
3. El sistema procesa cada PDF página por página, aplicando extracción nativa con PyMuPDF o, si la página tiene menos de cincuenta caracteres útiles, reconocimiento óptico de caracteres con Tesseract a 300 DPI.	Configuración OCR: lang="spa", PSM 6.
4. El sistema descarta el preámbulo y la sección "CONSIDERANDO" mediante la función recortar_a_cuerpo_normativo.	Expresión regular que identifica "Artículo 1.- [Mayúscula]" como marcador de inicio del cuerpo normativo.
5. El sistema segmenta el texto por encabezados de artículo y genera fragmentos (*chunks*), aplicando sub-fragmentación con solapamiento de 200 caracteres si el artículo supera 2.500 caracteres.	Expresión regular que detecta encabezados "Artículo N.- Título" como puntos de corte entre fragmentos.
6. El sistema genera *embeddings* de 384 dimensiones para cada fragmento mediante el modelo MiniLM y los persiste en ChromaDB con su metadata asociada.	Modelo paraphrase-multilingual-MiniLM-L12-v2.
7. El sistema construye en memoria el índice BM25 sobre los fragmentos textuales.	Algoritmo Okapi BM25.

8. El sistema notifica al administrador el resultado de la indexación, incluyendo cantidad de artículos detectados y rango numérico.	Estadísticas de indexación.
Precondiciones: El administrador debe haber autenticado su sesión (CU-08). Además, que deben existir archivos PDF de los reglamentos en formato accesible y con contenido textual extraíble o reconocible mediante OCR.	
Postcondiciones: La base documental queda disponible para consulta por parte del estudiante. ChromaDB conserva los *embeddings* entre sesiones, evitando reprocesamiento al reiniciar la aplicación.	
Suposiciones: Los archivos PDF cumplen con un formato reglamentario reconocible, con artículos numerados y títulos que comienzan con mayúscula. De igual forma, el equipo del administrador cuenta con conexión estable a internet para invocar a Groq y HuggingFace durante el procesamiento.	
Garantía de éxito: Todos los artículos del PDF son detectados, indexados y disponibles para consulta siendo que en el panel de diagnóstico se muestra una cobertura completa sin huecos numéricos.	
Garantía mínima: Al menos los artículos con encabezados estándar (Artículo N.- Título) son indexados; si alguno falla, el sistema lo reporta en el panel de diagnóstico como artículo faltante.	
Requerimientos cumplidos: Permitir la carga, extracción híbrida, segmentación, indexación dual y notificación de resultado al administrador.	
Cuestiones pendientes: ¿Se implementará procesamiento por lotes asíncrono para corpus de mayor tamaño? ¿Se incorporará detección automática de cambios entre versiones del reglamento?	
Prioridad: Alta	
Riesgo: Medio	

ILUSTRACIÓN 2 CARGA Y PROCESAMIENTO DE REGLAMENTOS: CU-01



Nota: Elaboración propia. Diagrama caso de uso de la carga y procesamiento de reglamentos PDF por parte del administrador.

TABLA 2 CASO DE USO: CONSULTAR ARTÍCULO ESPECÍFICO (CU-02)

Nombre del caso de uso: Consultar artículo específico	ID Único: CU-02
Área: -	
Actor(es): Estudiante	
Interesados: Estudiantes	
Nivel: Primario	
Descripción: Este caso de uso describe la consulta directa sobre un artículo específico de los reglamentos cargados, teniendo que el estudiante formula una pregunta referenciando un número de artículo (por ejemplo, "qué dice el Art. 29"), el sistema	

identifica la intención como consulta por artículo, recupera el fragmento correspondiente mediante coincidencia exacta por metadata y el modelo generativo redacta la respuesta con la cita formal del artículo consultado.	
Evento desencadenador: El estudiante escribe una consulta en el componente de chat (st.chat_input) que contiene una referencia explícita a un número de artículo.	
Tipo de desencadenador: ☑ Externo ☐ Temporal	
Pasos realizados (ruta principal)	**Información para los pasos**
1. El estudiante escribe la consulta en lenguaje natural en el campo de chat.	Texto de la consulta.
2. El sistema descarta la pregunta de la ruta de saludos y meta-preguntas tras evaluar los patrones regulares existentes.	Expresiones regulares que reconocen: - Saludos: mensajes que contienen solo palabras como "hola", "gracias", "bye" - Meta-preguntas: frases sobre estructura del corpus como "cuántos artículos", "lista los reglamentos", "tabla de contenidos".
3. El sistema invoca el *router* de intención, el cual clasifica la consulta como tipo “articulo” e identifica el número solicitado.	Respuesta JSON del *router* con campos: tipo = "articulo", articulos = [7].
4. El sistema asigna prioridad máxima (*score* 10.000) a los fragmentos cuyo número de artículo coincide exactamente con el solicitado.	Coincidencia exacta por metadata.
5. El sistema construye el contexto con el fragmento recuperado y lo envía al modelo generativo junto con las catorce reglas del *prompt.*	*Prompt* estructurado con 14 reglas de citación y fragmentos del reglamento como contexto.
6. El modelo Llama 3.3 70B genera una respuesta restringida al contenido del artículo, incluyendo la cita en formato [Art. N — Título — documento].	Configuración temperature=0. temperature=0 garantiza que la misma pregunta reciba siempre la misma respuesta.

7. El sistema muestra la respuesta en la interfaz de chat y agrega tanto la pregunta como la respuesta al historial conversacional.	La respuesta se renderiza en pantalla y se almacena temporalmente en st.session_state.historial para mantener el contexto mientras la sesión está activa. Al cerrar el navegador o expirar la sesión, el historial se pierde.
Precondiciones: Existe al menos un reglamento indexado en ChromaDB. El estudiante tiene acceso a la interfaz mediante navegador web. No se requiere autenticación.	
Postcondiciones: El estudiante recibe el contenido textual del artículo solicitado, junto con su cita formal verificable contra el reglamento original.	
Suposiciones: El estudiante formula su consulta en español con una mención explícita o implícita del número de artículo. Así mismo que el servicio Groq Cloud esté disponible.	
Garantía de éxito: El estudiante recibe el contenido exacto del artículo solicitado con la cita correcta.	
Garantía mínima: Si el artículo no existe en el corpus, el sistema responde explícitamente “No encontré información sobre eso en los reglamentos cargados” en lugar de inventar contenido.	
Requerimientos cumplidos: Consulta en lenguaje natural, recuperación por coincidencia exacta, generación con citación obligatoria.	
Cuestiones pendientes: ¿Se incorporará una vista interactiva que permita expandir el artículo completo al hacer clic sobre la cita?	
Prioridad: Alta	
Riesgo: Bajo	

TABLA 3 CASO DE USO: CONSULTAR POR TEMA (CU-03)

Nombre del caso de uso: Consultar por tema	**ID Único:** CU-03
Área: -	
Actor(es): Estudiante	
Interesados: Estudiantes	
Nivel: Primario	
Descripción: Este caso de uso describe la consulta temática del estudiante, donde la pregunta no menciona un artículo específico sino un concepto general (por ejemplo, "¿qué pasa si pierdo mi beca?"). El sistema clasifica la consulta como temática, expandiendo sinónimos mediante el *router* de intención, de igual manera, ejecuta una búsqueda híbrida combinando BM25 y vectorial, fusiona y elimina los candidatos duplicados, permitiendo que el modelo generativo de una respuesta muy acertada, citando todos los artículos relevantes encontrados.	
Evento desencadenador: El estudiante escribe en el chat una consulta temática sin referencia explícita a un artículo.	
Tipo de desencadenador: ☑ Externo ☐ Temporal	
Pasos realizados (ruta principal)	**Información para los pasos**
1. El estudiante escribe una consulta temática (por ejemplo, "causas de pérdida de beca").	Texto de la consulta.
2. El *router* clasifica la consulta como tipo "tema" y genera una expansión de sinónimos.	Respuesta JSON del *router* con: - tipo = "tema" - temas = lista de sinónimos expandidos (ej: "pérdida", "causales", "suspensión") - reformulada = versión clara de la consulta
3. El sistema tokeniza la consulta y ejecuta la búsqueda BM25 sobre el corpus indexado, recuperando los doce fragmentos con mayor coincidencia léxica.	Tokenización con re.findall(r'\b\w+\b', ...)
4. El sistema genera el *embedding* de la consulta reformulada con MiniLM y ejecuta búsqueda vectorial en ChromaDB, recuperando los doce fragmentos más cercanos semánticamente.	Modelo de embeddings, distancia coseno.

5. El sistema fusiona los candidatos de ambas fuentes, deduplica por la tupla (documento, artículo, prefijo del texto) y ordena por puntaje agregado.	Fusión de candidatos de BM25 y búsqueda vectorial. Ocurre una deduplicación, ordenamiento descendente por puntaje agregado y selección de los 18 fragmentos más relevantes
6. El sistema selecciona los dieciocho fragmentos finales y construye el contexto final para el modelo generativo.	Top-18 final
7. El modelo Llama 3.3 70B sintetiza una respuesta correcta citando cada artículo relevante con su formato estándar.	*Prompt* con los 18 fragmentos recuperados como contexto y reglas de citación múltiple obligatoria: cada afirmación debe incluir [Art. N — Título — documento]
Precondiciones: Existe al menos un reglamento indexado y que el servicio de Groq Cloud esté disponible.	
Postcondiciones: El estudiante recibe una respuesta consolidada que cubre todos los artículos relevantes al tema, con citas explícitas a cada uno.	
Suposiciones: El estudiante formula la consulta de manera comprensible y se refiere a un tema cubierto por algún reglamento.	
Garantía de éxito: La respuesta incluye todos los artículos pertinentes al tema, sin omitir ninguno que este en los fragmentos recuperados.	
Garantía mínima: Si el tema no está documentado, el sistema declara explícitamente la ausencia de información.	
Requerimientos cumplidos: Recuperación híbrida (BM25 + vectorial), expansión de sinónimos, citación múltiple obligatoria.	
Cuestiones pendientes: ¿Se incorporará un *reranker cross-encoder* para mejorar la precisión del top-18?	
Prioridad: Alta	
Riesgo: Medio	

TABLA 4 CASO DE USO: CONSULTAR ESTRUCTURA DEL CORPUS (CU-04)

Nombre del caso de uso: Consultar estructura del corpus	**ID Único:** CU-04
Área: Consulta meta-estructural	
Actor(es): Estudiante	
Interesados: Estudiantes.	
Nivel: Secundario	
Descripción: Este caso de uso describe las consultas del estudiante sobre la estructura del corpus, cantidad total de artículos, listado completo, tabla de contenidos, que el sistema responde sin invocar al *pipeline* de generación, accediendo directamente a la metadata indexada en sesión, puesto que, este flujo se diseñó específicamente para superar la limitación inherente de los sistemas RAG, que solo recuperan los *top-K* fragmentos.	
Evento desencadenador: El estudiante escribe una consulta sobre la estructura del corpus (por ejemplo, "cuántos artículos tiene el Reglamento de Acceso a Becas y Ayudas Económicas").	
Tipo de desencadenador: ☑ Externo ☐ Temporal	
Pasos realizados (ruta principal)	**Información para los pasos**
1. El estudiante escribe una consulta sobre la estructura.	Texto de la consulta
2. El sistema evalúa la consulta contra cinco patrones regulares de meta-preguntas: contar artículos, contar documentos, listar artículos, listar documentos, índice.	Patrones de expresiones regulares: _PATRON_CONTAR_ARTS, _PATRON_LISTAR_ARTS, etc.
3. Al identificar coincidencia, el sistema accede directamente a st.session_state.chunks_arts y chunks_titulos.	Estado de sesión Streamlit
4. El sistema agrupa los artículos por documento mediante la función auxiliar _agrupar_arts_por_doc.	Diccionario {documento: {número: título}}
5. El sistema detecta huecos en la numeración y los reporta como advertencias.	Análisis de continuidad numérica

6. El sistema responde con el formato apropiado al tipo de meta-pregunta, sin invocar al modelo generativo.	Respuesta directa desde metadata
Precondiciones: Existe al menos un reglamento indexado.	
Postcondiciones: El estudiante recibe información estadística verificable sobre el corpus, con detalle por documento y advertencias sobre artículos no detectados si las hay.	
Suposiciones: Las consultas estructurales del estudiante caen dentro de los patrones reconocidos por las expresiones regulares definidas.	
Garantía de éxito: Se responde con conteos exactos y listados completos, sin truncamiento por *top-K*.	
Garantía mínima: Si la meta-pregunta no es reconocida, el sistema redirige la consulta al *pipeline* RAG normal.	
Requerimientos cumplidos: Manejo de meta-preguntas sin invocación al modelo generativo, acceso a metadata completa.	
Cuestiones pendientes: ¿Se ampliará el catálogo de patrones para cubrir nuevas formulaciones lingüísticas?	
Prioridad: Media	
Riesgo: Bajo	

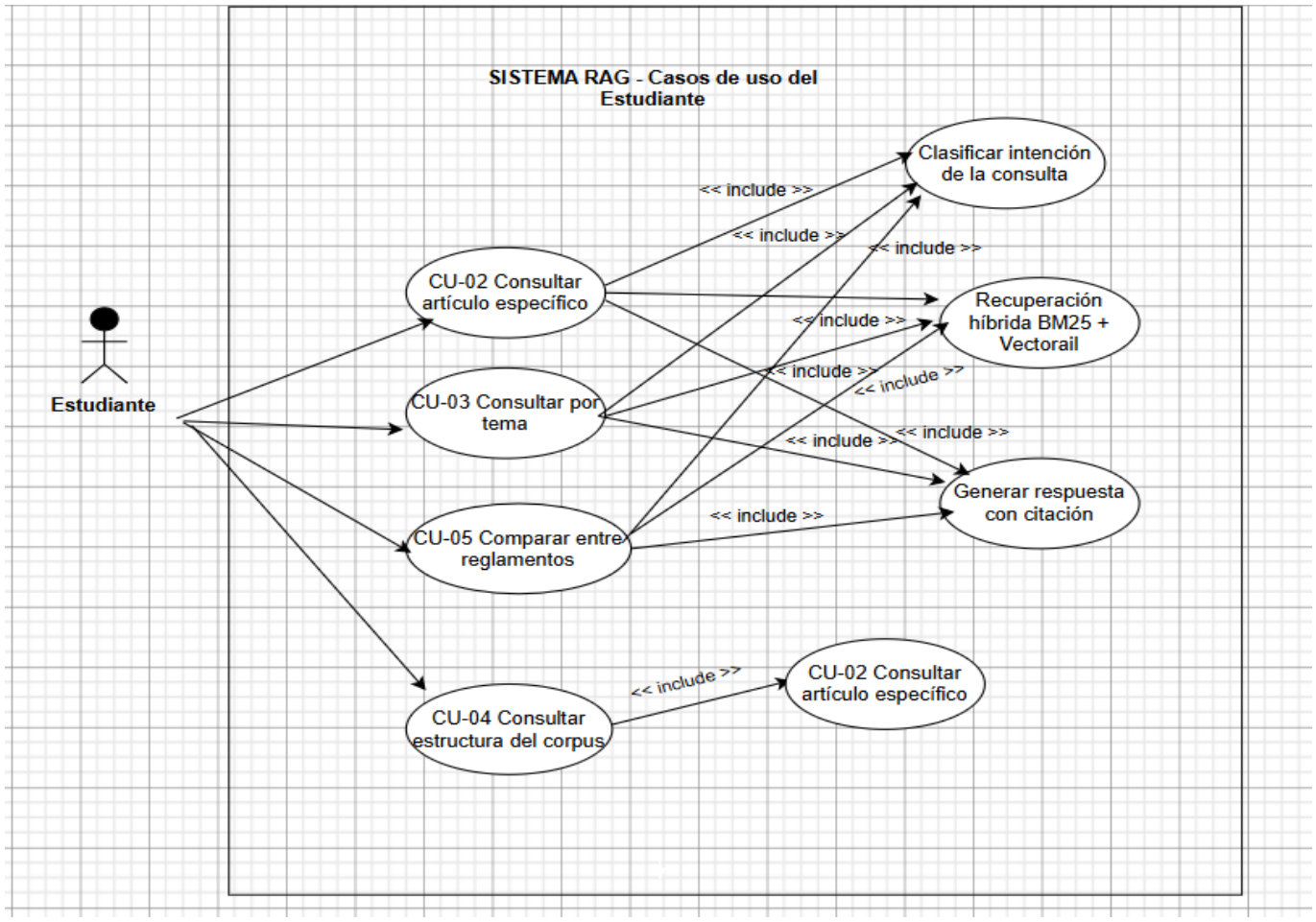
TABLA 5 CASO DE USO: COMPARAR ENTRE REGLAMENTOS (CU-05)

Nombre del caso de uso: Comparar entre reglamentos	**ID Único:** CU-05
Área: Consulta comparativa	
Actor(es): Estudiante	
Interesados: Estudiantes	
Nivel: Secundario	
Descripción: Este caso de uso describe las consultas en las que el estudiante solicita comparar dos artículos o dos temas provenientes de reglamentos distintos, considerando que el *router* identifica la intención comparativa, el sistema recupera	

<table>
<tr><td colspan="2">fragmentos de ambos documentos y el modelo generativo elabora una respuesta contrastiva con citas explícitas a cada documento.</td></tr>
<tr><td colspan="2">Evento desencadenador: El estudiante formula una consulta que requiere comparar contenidos entre reglamentos (por ejemplo, “diferencia entre el Art. 28 del Reglamento de Acceso a Becas y Ayudas Económicas y el Art. 28 del Reglamento General de Estudiantes”).</td></tr>
<tr><td colspan="2">Tipo de desencadenador: ☑ Externo ☐ Temporal</td></tr>
<tr><td>Pasos realizados (ruta principal)</td><td>Información para los pasos</td></tr>
<tr><td>1. El estudiante escribe la consulta comparativa.</td><td>Texto de la consulta</td></tr>
<tr><td>2. El router clasifica la consulta como tipo “comparativa” e identifica los artículos o temas a comparar.</td><td>Respuesta JSON del router con: - tipo = "comparativa" - articulos = [28] - documento_filtro = vacío ya que se sobreentiende que involucra ambos documentos - reformulada = versión explícita de la comparación.</td></tr>
<tr><td>3. El sistema recupera fragmentos correspondientes a ambos documentos.</td><td>Búsqueda híbrida que se lleva a cabo en los dos documentos.</td></tr>
<tr><td>4. El sistema construye el contexto que incluye las fuentes de cada fragmento.</td><td>Cabeceras [Documento: X | Artículo Y]</td></tr>
<tr><td>5. El modelo Llama 3.3 70B genera una respuesta contrastiva destacando las similitudes y diferencias, con cita explícita a cada documento.</td><td>Prompt con fragmentos de ambos documentos etiquetados con [Documento: X | Artículo N]. La citación del documento de origen es obligatoria en toda respuesta y más cuando dos artículos comparten el mismo número, pero provienen de reglamentos distintos, evitando así ambigüedad para el estudiante.</td></tr>
<tr><td colspan="2">Precondiciones: Existen al menos dos reglamentos indexados en el corpus.</td></tr>
</table>

Postcondiciones: El estudiante recibe una respuesta estructurada que contrasta el contenido de ambos reglamentos, con citas claramente diferenciadas.
Suposiciones: Los reglamentos comparados contienen información sobre el tema o artículos solicitados.
Garantía de éxito: La respuesta presenta claramente las similitudes y diferencias entre los artículos comparados.
Garantía mínima: Si uno de los documentos no contiene información sobre el tema, el sistema lo declara explícitamente.
Requerimientos cumplidos: Recuperación de múltiples documentos, generación contrastiva, citación diferenciada.
Cuestiones pendientes: ¿Se incorporará una visualización tabular paralela de las cláusulas comparadas?
Prioridad: Media
Riesgo: Medio

ILUSTRACIÓN 3 CASOS DE USO DEL ESTUDIANTE



Nota: Elaboración propia. Diagrama casos de uso del Estudiante

TABLA 6 CASO DE USO: REEMPLAZAR EL CORPUS DOCUMENTAL (CU-06)

Nombre del caso de uso: Reemplazar el corpus documental	**ID Único:** CU-06
Área: -	
Actor(es): Administrador del sistema	
Interesados: Administrador, Estudiantes	
Nivel: Primario	
Descripción: Este caso de uso describe el flujo mediante el cual el administrador reemplaza completamente el corpus documental cargado, por lo que, es necesario cuando se actualiza la versión de un reglamento o cuando se desea cambiar el conjunto de documentos disponible para consultar.	

El sistema elimina la colección previa de ChromaDB, reinicia el estado de sesión y procesa los nuevos documentos como en CU-01.	
Evento desencadenador: El administrador, después de autenticarse, activa la opción “Cargar nuevos PDFs” en la vista de administrador.	
Tipo de desencadenador: ☑ Externo ☐ Temporal	
Pasos realizados (ruta principal)	**Información para los pasos**
1. El administrador autenticado activa la opción “Cargar nuevos PDFs”.	Vista de administrador, estado es_admin=True
2. El sistema elimina la colección actual de ChromaDB mediante client.delete_collection(COLLECTION_NAME).	Cliente ChromaDB
3. El sistema reinicia el estado de sesión: limpia el historial conversacional, las listas de *chunks* y el índice BM25.	st.session_state
4. El sistema solicita los nuevos archivos PDF y los procesa siguiendo el flujo descrito en CU-01.	Archivos PDF nuevos
5. El sistema confirma el reemplazo mostrando estadísticas de indexación del nuevo corpus.	Resumen de indexación
Precondiciones: El administrador ha autenticado su sesión (CU-08) y que los nuevos archivos PDF estén disponibles.	
Postcondiciones: La nueva base documental queda disponible y la anterior es completamente eliminada. La integridad de los datos se mantiene gracias a la atomicidad de la operación.	
Suposiciones: Los nuevos archivos cumplen con el formato reglamentario esperado.	
Garantía de éxito: El nuevo corpus se indexa completamente y queda disponible para consulta.	
Garantía mínima: Si la nueva indexación falla parcialmente, el sistema reporta los errores y el administrador puede reintentar.	
Requerimientos cumplidos: Reemplazo atómico del corpus y limpieza del estado de sesión.	

Cuestiones pendientes: ¿Se implementará versionado para mantener histórico de corpus anteriores?
Prioridad: Alta
Riesgo: Medio

TABLA 7 CASO DE USO: DIAGNOSTICAR EL SISTEMA (CU-07)

Nombre del caso de uso: Diagnosticar el sistema	**ID Único:** CU-07
Área: -	
Actor(es): Administrador del sistema	
Interesados: Administrador, Equipo de desarrollo	
Nivel: Secundario	
Descripción: Este caso de uso describe la consulta del administrador al panel de diagnóstico, donde el sistema presenta estadísticas detalladas sobre el corpus indexado: cantidad de fragmentos, artículos únicos por documento, rango numérico, posibles huecos en la numeración y estado del índice BM25, siendo que, esta vista es esencial para validar la calidad de la indexación y detectar problemas como artículos no extraídos correctamente.	
Evento desencadenador: El administrador autenticado despliega el panel de diagnóstico en la vista lateral.	
Tipo de desencadenador: ☑ Externo ☐ Temporal	
Pasos realizados (ruta principal)	**Información para los pasos**
1. El administrador autenticado expande el panel de diagnóstico en el *sidebar*.	Componente st.expander
2. El sistema muestra el total de fragmentos indexados y los artículos únicos por documento	Conteo de len(chunks_texto) para fragmentos totales y agrupación sobre st.session_state.chunks_fuentes para artículos únicos por documento.

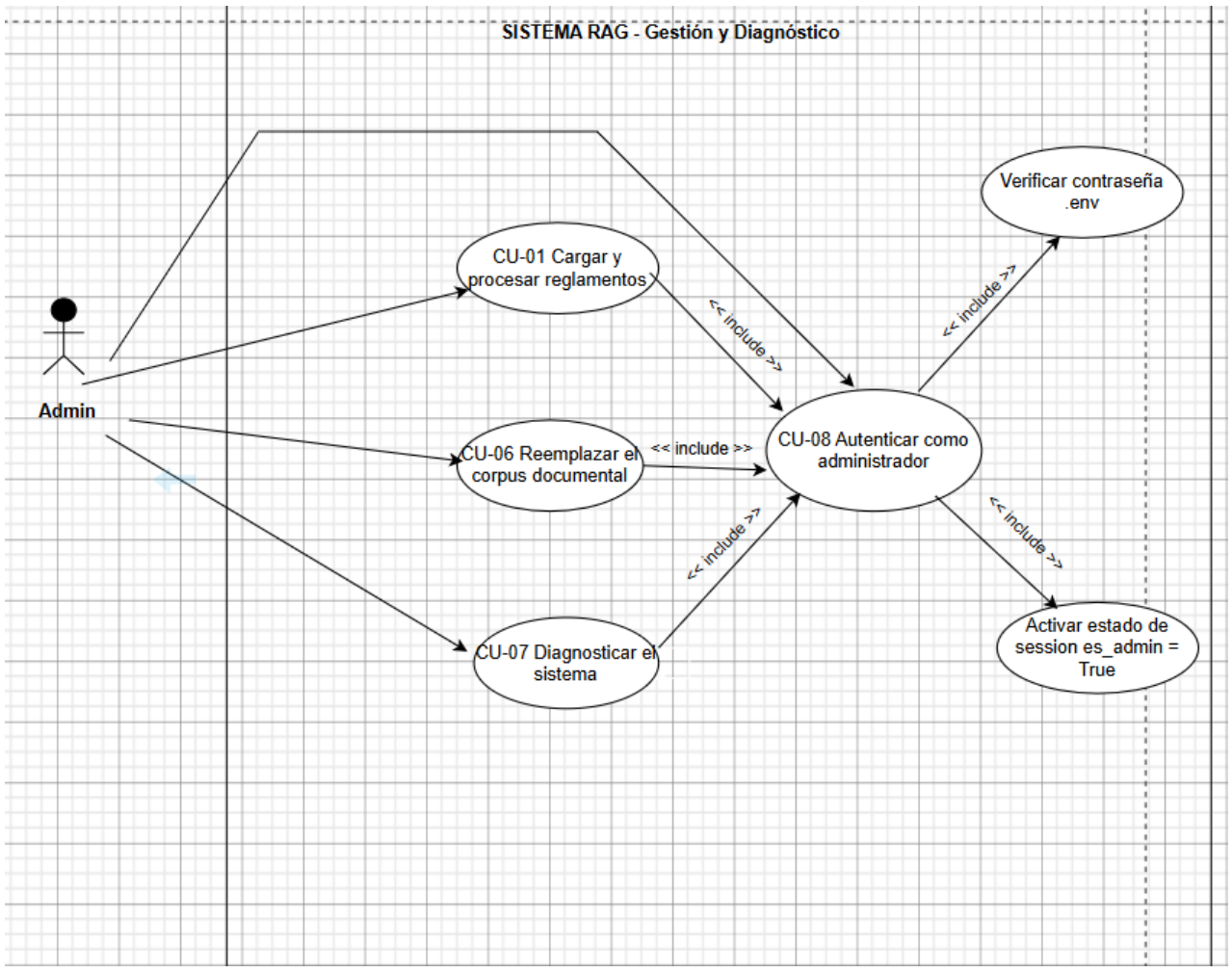
	Resultado visible: 107 fragmentos, 29 artículos en el Reglamento de Acceso a Becas y Ayudas Económicas, 70 en el Reglamento General de Estudiantes
3. El sistema identifica los artículos únicos detectados por documento y su rango numérico.	Función _agrupar_arts_por_doc
4. El sistema detecta huecos en la numeración (artículos esperados, pero no encontrados).	Análisis de continuidad
5. El sistema reporta el estado del índice BM25 y su coherencia con los *chunks* indexados.	len(bm25.doc_freqs) vs len(chunks_texto)
Precondiciones: El administrador ha autenticado su sesión (CU-08) y existe al menos un reglamento indexado.	
Postcondiciones: El administrador obtiene visibilidad completa del estado interno del corpus y puede actuar para corregir problemas detectados.	
Suposiciones: El administrador tiene conocimientos técnicos suficientes para interpretar las métricas presentadas.	
Garantía de éxito: Toda la información de diagnóstico se presenta correctamente y permite la detección de problemas.	
Garantía mínima: Si el cálculo de alguna métrica falla, el panel muestra “No disponible” sin afectar al resto.	
Requerimientos cumplidos: Métricas adecuadas para el diagnóstico técnico completo.	
Cuestiones pendientes: ¿Se incorporarán métricas de performance histórico (latencia media por tipo de consulta)?	
Prioridad: Media	
Riesgo: Bajo	

TABLA 8 CASO DE USO: AUTENTICAR COMO ADMINISTRADOR (CU-08)

Nombre del caso de uso: Autenticar como administrador	**ID Único:** CU-08
Área: -	
Actor(es): Administrador del sistema	
Interesados: Administrador	
Nivel: Primario	
Descripción: Este caso de uso describe el proceso de autenticación del administrador para acceder a las funciones de gestión del corpus. La autenticación se realiza mediante contraseña configurada en una variable de entorno (ADMIN_PASSWORD), comparada en tiempo de ejecución, por tanto, tras una autenticación exitosa, el estado de sesión cambia y aparecen los controles de gestión que se encontraban ocultos.	
Evento desencadenador: El usuario despliega el panel de acceso de administrador, el cual se encuentra en la parte inferior del *sidebar*, e ingresa una contraseña.	
Tipo de desencadenador: ☑ Externo ☐ Temporal	
Pasos realizados (ruta principal)	**Información para los pasos**
1. El usuario expande el panel “Acceso administrador” en el *sidebar*.	Componente st.expander colapsado por defecto
2. El usuario ingresa la contraseña en el campo de texto enmascarado.	Componente st.text_input(type="password")
3. El usuario pulsa el botón “Ingresar”.	Disparador del flujo de validación
4. El sistema compara la contraseña ingresada con la variable de entorno ADMIN_PASSWORD.	Comparación de cadenas
5. Si las contraseñas coinciden, el sistema establece st.session_state.es_admin = True y ejecuta st.rerun().	Estado de sesión actualizado
6. La interfaz se re-renderiza mostrando los controles de gestión: carga de PDFs y el panel de diagnóstico.	Renderizado condicional

7. Si las contraseñas no coinciden, el sistema muestra un mensaje de error sin revelar información adicional.	Mensaje genérico “Contraseña incorrecta”
Precondiciones: La variable ADMIN_PASSWORD está configurada en el archivo .env. El archivo .env está protegido del control de versiones.	
Postcondiciones: Después de autenticarse exitosamente, el administrador accede a las funciones de gestión (CU-01, CU-06, CU-07), además la sesión administrativa puede cerrarse en cualquier momento pulsando “Cerrar sesión admin”, restaurando la vista del estudiante.	
Suposiciones: El administrador conoce la contraseña configurada y la maneja con la confidencialidad apropiada.	
Garantía de éxito: El administrador autenticado tiene acceso completo a las funciones administrativas durante la sesión.	
Garantía mínima: Una contraseña incorrecta nunca va a permitir el acceso, haciendo que el sistema muestre la vista del estudiante.	
Requerimientos cumplidos: Control de acceso por rol, separación de interfaz por rol, seguridad de credenciales fuera del código fuente.	
Cuestiones pendientes: ¿Se migrará a un esquema de autenticación basado en OAuth o LDAP institucional en futuras versiones?	
Prioridad: Alta	
Riesgo: Medio	

ILUSTRACIÓN 4 CASOS DE USO DE GESTIÓN Y DIAGNÓSTICO



Nota: Elaboración propia. Diagrama caso de uso de gestión y diagnóstico.

3.5 **Identificación y limpieza de la documentación**

Esta sección desarrolla la fase de ingeniería de datos, cuyo objetivo es preparar el corpus documental para su consumo por parte del modelo. Siendo que, en proyectos basados en RAG, la calidad de los datos de entrada influye directamente en la precisión de las respuestas generadas, lo que se debe realizar un proceso riguroso de identificación, normalización y limpieza.

3.5.1 Identificación del corpus documental

El corpus institucional tomado para el prototipo está conformado por dos documentos oficiales de la Pontificia Universidad Católica del Ecuador, ambos vigentes y publicados a través de la Gaceta Institucional:

- Reglamento Específico de Becas y Ayudas Económicas para Carreras y Programas, versión 01.01, vigente desde el 25 de octubre de 2024.
- Reglamento General de Estudiantes, versión 01.03, vigente desde el 29 de julio de 2025.

3.5.2 Caracterización de los documentos

El análisis de ambos documentos mostró diferencias estructurales significativas que condicionaron el diseño del módulo de extracción.

TABLA 9 COMPARACIÓN DE LOS REGLAMENTOS DEL CORPUS

Característica	**Reglamento de Acceso a Becas y Ayudas Económicas**	**Reglamento General de Estudiantes**
Extensión	17 páginas	39 páginas
Tipo de PDF	Imagen escaneada	Texto nativo
Cantidad de artículos	29	70
Estrategia de extracción	OCR mediante Tesseract	Extracción nativa con PyMuPDF
Estructura	Considerando + 8 títulos + disposiciones	Considerando + 11 capítulos + disposiciones

La heterogeneidad en el formato de origen impuso la necesidad de un mecanismo híbrido que decidiera, página por página, la estrategia óptima de extracción.

3.5.3 Estrategia de extracción híbrida

El prototipo implementa una solución adaptativa para cada página, se aplica primero la extracción nativa con PyMuPDF y si el resultado contiene menos de cincuenta caracteres no blancos, se asume que la página está escaneada y se aplica OCR mediante Tesseract a una rasterización de 300 DPI, teniendo que esta estrategia permite optimizar el tiempo de procesamiento, evitando

aplicar OCR innecesariamente sobre páginas que ya contienen texto extraíble, y garantiza la obtención completa del corpus independientemente del formato de origen.

3.5.4 Limpieza y normalización del texto

El texto extraído contiene elementos que se consideran como ruido para el modelo de lenguaje y que deben eliminarse antes de la indexación, teniendo que, se identificaron tres categorías de ruido:

- Encabezados y pies de página recurrentes: Aparecen en cada página del PDF (logotipos institucionales, código del documento, número de versión, paginación, lema institucional). Se eliminan mediante un patrón de expresión regular que reconoce líneas completas con estos contenidos.
- Anotaciones de reformas legales: Frases como “Artículo reformado por resolución del Consejo Superior…” se descartan al no otorgar contenido normativo, sino metadatos administrativos.
- Disposiciones reformadas o derogadas.

3.5.5 Recorte del cuerpo normativo

Una clave importante durante la ingeniería de datos fue la presencia, en la sección “CONSIDERANDO” de los reglamentos, de referencias a artículos de leyes externas (Ley Orgánica de Educación Superior, Ley Orgánica de Discapacidades, entre otras). Estas referencias siguen el patrón “el artículo N de la Ley…”, estructura que puede ser confundida con encabezados reales por algoritmos basados en expresiones regulares no sensibles a mayúsculas.

Para disminuir esta fuente de error, se diseñó una función recortar_a_cuerpo_normativo que descarta todo contenido anterior a la primera ocurrencia del patrón Artículo 1.- [Letra mayúscula], donde la mayúscula inicial del título es un indicador confiable del comienzo del cuerpo reglamentario y no aparece en la tabla de contenidos ni en los considerandos. De igual forma, las expresiones regulares de segmentación se diseñaron sin la opción IGNORECASE (no distinga entre mayúsculas y minúsculas), garantizando que únicamente las apariciones de “Artículo” en mayúscula correspondientes a encabezados reales se interpreten como inicio de un nuevo segmento.

3.5.6 Segmentación por artículos

A diferencia de las técnicas de fragmentación de texto basadas en ventanas deslizantes de tamaño fijo, el prototipo toma una segmentación semántica natural ya que cada artículo del reglamento es un fragmento. Esta decisión es coherente con la naturaleza del corpus, donde los artículos son unidades que contienen un objeto normativo específico.

Para artículos extensos con más de 2.500 caracteres, se aplica una sub-fragmentación con solapamiento de 200 caracteres, conservando en cada sub-fragmento un encabezado contextual que mantiene la trazabilidad, proporcionando que cada fragmento conserve la siguiente metadata estructurada:

- source: nombre del documento de origen.
- articulo: número del artículo, como cadena de caracteres.
- título: título del artículo extraído, truncado a 150 caracteres.
- sub_chunk: índice del sub-fragmento, en caso de existir más de uno por artículo.

3.5.7 Métricas de calidad de los datos

En referencia con la fase de ingeniería de datos, se establecieron indicadores de calidad medibles sobre el corpus indexado:

- Cobertura de artículos: Cantidad de artículos detectados respecto al total documentado en cada reglamento. Valor objetivo: 100%.
- Tasa de falsos positivos: La tasa de falsos positivos mide qué proporción de los fragmentos indexados corresponden a referencias legislativas externas detectadas erróneamente como artículos del reglamento, por ende, el valor objetivo de 0% se alcanza mediante el recorte del cuerpo normativo y la restricción de sensibilidad a mayúsculas en el patrón de segmentación.
- Continuidad numérica: Verificación de la secuencia 1, 2, …, N sin saltos. Permite detectar artículos no extraídos.
- El panel de diagnóstico del prototipo expone estas métricas al administrador, facilitando la verificación visual de la integridad del corpus tras cada reindexación.

3.6 **Diseño de la arquitectura del sistema**

La arquitectura del prototipo se organiza en cinco capas funcionales, dirigidas por un orquestador central basado en LangChain.

3.6.1 Capas de la arquitectura

TABLA 10 ARQUITECTURA POR CAPAS DEL PROTOTIPO

Capa	**Componentes**	**Responsabilidad**
Presentación	Streamlit (dos vistas por rol)	Vista estudiante: es el chat conversacional. Vista administrador: es donde se realiza la carga de PDFs, diagnóstico y gestión del corpus. El cambio de vista se gestiona mediante estado de sesión y autenticación por contraseña.
Orquestación	LangChain	Coordinación del *pipeline* RAG mediante tres plantillas de *prompt* especializadas: una para clasificar la intención de la consulta, otra para reformular preguntas que tengan relación con el historial y una tercera para generar la respuesta final con las reglas de citación obligatoria.

Inferencia	Groq Cloud (Llama 3.3 70B)	Generación de respuesta y clasificación de intención
Recuperación	ChromaDB + rank-bm25	ChromaDB: recupera fragmentos por similitud vectorial. MiniLM vectoriza la consulta en tiempo real y compara contra *embeddings* persistidos en disco rank-bm25: búsqueda léxica sobre *chunks* en RAM (session_state). Índice reconstruido al iniciar sesión.
Procesamiento de datos	PyMuPDF + Tesseract	Extracción híbrida de texto

3.6.2 Flujo del pipeline de indexación

La indexación se realiza una única vez por cada conjunto de documentos cargado y tiene los siguientes pasos:

1. Lectura del archivo PDF.
2. Extracción página a página mediante una estrategia híbrida.
3. Limpieza del texto resultante.
4. Recorte del cuerpo normativo.
5. Segmentación por artículos con captura de metadata.
6. Generación de *embeddings* vectoriales.
7. Persistencia en ChromaDB.
8. Construcción del índice BM25 en memoria.

3.6.3 Flujo del pipeline de consulta

El *pipeline* de consulta se realiza por cada interacción del usuario y se organiza en tres rutas posibles, las cuales son:

- Ruta de saludos: Las consultas que coinciden con patrones de saludo se atienden mediante una respuesta predefinida, sin invocar al modelo.
- Ruta de meta-preguntas: Las consultas sobre la estructura del corpus se atienden directamente desde la metadata indexada en sesión, evitando la limitación de los algoritmos de recuperación que retornan únicamente los fragmentos *top-K*.
- Ruta de RAG completo: Las consultas de contenido semántico y léxico se procesan mediante el *pipeline* completo, es decir con contextualización del historial, clasificación de intención, recuperación híbrida, fusión de candidatos, construcción de *prompt* y generación de respuesta.

3.6.4 Estrategia de recuperación híbrida y re-ranking

La recuperación se desarrolla en tres niveles complementarios:

- Coincidencia exacta por número de artículo: Cuando el *router* identifica un número específico, el sistema asigna un puntaje de 10.000 a los fragmentos correspondientes, garantizando su prioridad absoluta sobre cualquier resultado semántico.
- Búsqueda léxica BM25: Extrae los doce fragmentos de texto estadísticamente más relevantes, basándose en la coincidencia exacta de las palabras de la consulta y sus respectivos sinónimos.
- Búsqueda vectorial densa: Recupera los doce fragmentos semánticamente más cercanos al *embedding* de la consulta reformulada.

Los candidatos de las tres fuentes se unifican mediante una función de deduplicación basada en la tupla (documento, artículo, prefijo del texto), después se ordenan por puntaje y se truncan a los dieciocho fragmentos más relevantes, los cuales son el contexto final para el modelo generativo.

3.7 Selección del conjunto tecnológico

La selección de cada componente del conjunto tecnológico responde a criterios técnicos verificables.

TABLA 11 SELECCIÓN TECNOLÓGICA Y ALTERNATIVAS EVALUADAS

Componente	**Tecnología seleccionada**	**Alternativas evaluadas**	**Criterio determinante**
Lenguaje	Python 3.11+	JavaScript, Java	Ecosistema maduro de IA y NLP
Modelo generativo	Llama 3.3 70B (Groq)	Mistral local (Ollama), GPT-4	Latencia baja sin dependencia de hardware local
Base de datos vectorial	ChromaDB	Pinecone, FAISS	Persistencia local + *open source*
Embeddings	paraphrase-multilingual-MiniLM-L12-v2	OpenAI Ada, BGE-M3	Soporte multilingüe en CPU
Recuperación léxica	rank-bm25	Elasticsearch, Whoosh	Seleccionado por su naturaleza embebida ya que opera como una biblioteca Python sin requerir servicios externos
Orquestador	LangChain	LlamaIndex, Haystack	Adopción generalizada y modularidad
Procesamiento PDF	PyMuPDF (fitz)	pdfplumber, pypdf	API unificado para texto y rasterización
OCR	Tesseract (vía pytesseract)	EasyOCR, PaddleOCR	Madurez y soporte para español
Interfaz web	Streamlit	Flask, FastAPI + *frontend*	Velocidad de prototipado
Gestión de configuración	python-dotenv	os.environ directo	Separación de credenciales sensibles

CAPÍTULO IV: CONSTRUCCIÓN Y VALIDACIÓN DEL PROTOTIPO

El presente capítulo documenta la implementación del prototipo siguiendo las fases tres, cuatro y cinco de CRISP-ML(Q): ingeniería del modelo, evaluación y despliegue (Studer et al., 2021). En lo que respecta a la fase de despliegue, el presente trabajo documenta los artefactos técnicos necesarios para su futura implementación institucional, ya que el alcance definido en el Capítulo I pertenece a un prototipo funcional en entorno controlado para demostración técnica. Además, en este capítulo se describen los componentes funcionales construidos, la interfaz desarrollada, el plan de pruebas aplicado y la documentación técnica entregable.

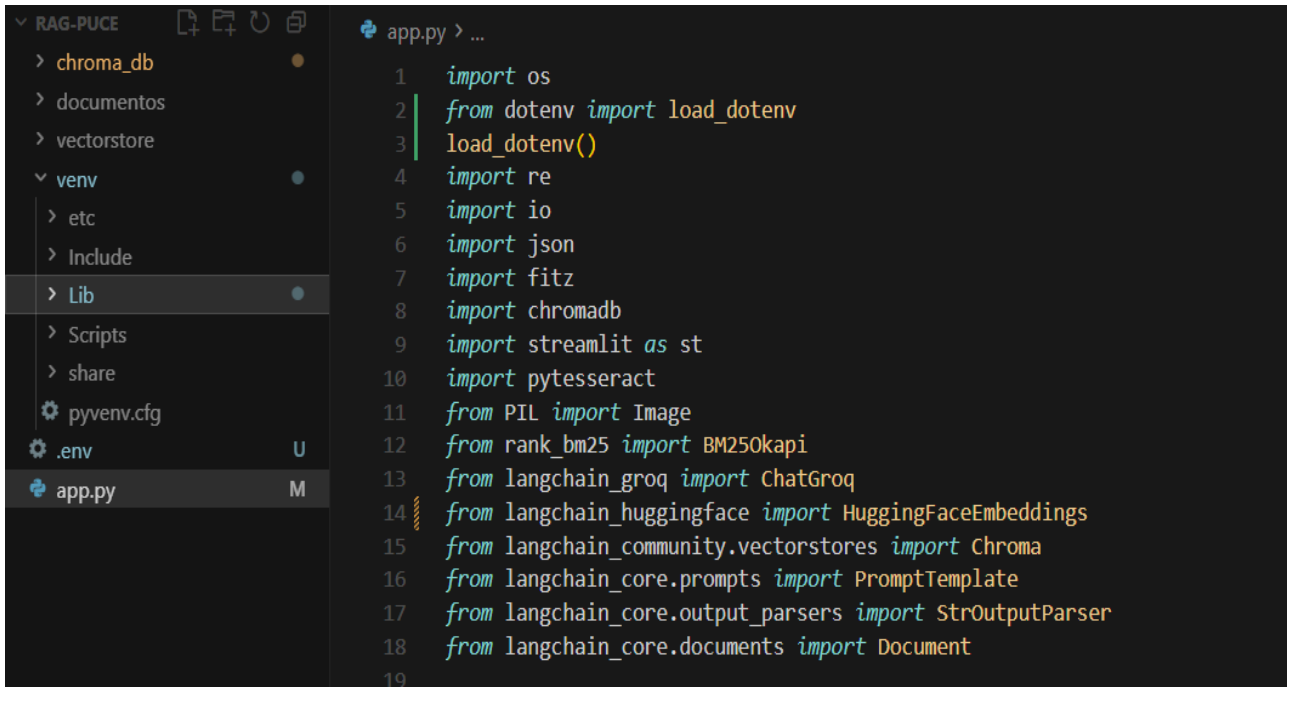
4.1 Preparación del entorno de desarrollo

Para preparar el entorno de desarrollo del prototipo, fue necesario considerar los criterios del ecosistema tecnológico de Python para sistemas de inteligencia artificial y procesamiento de lenguaje natural, por consiguiente, durante la fase de selección tecnológica, se evaluó inicialmente Ollama como motor de inferencia local, herramienta que permite ejecutar modelos de lenguaje de gran escala directamente en hardware del desarrollador sin dependencias de servicios externos (Rajesh et al., 2025). Sin embargo, esta opción fue descartada por dos razones técnicas: primero, la ejecución local de un modelo de 70.000 millones de parámetros requiere hardware especializado con al menos 40 GB de VRAM, recursos no disponibles en el entorno de desarrollo utilizado; segundo, la latencia de inferencia sobre CPU resultó incompatible con los requisitos de tiempo de respuesta definidos en RNF-03.

Por lo que, estas restricciones llevaron a la selección de Groq Cloud como motor de inferencia para el prototipo, plataforma que delega el procesamiento computacional a servidores especializados y permite operar el modelo Llama 3.3 70B Versatile desde hardware convencional con menos latencia. No obstante, esta decisión implica una limitación de privacidad, debido a que las consultas formuladas por los estudiantes se transmiten a servidores externos para su procesamiento, aspecto que se menciona como recomendación en el Capítulo V.

Así mismo, se evaluaron diferentes alternativas en cuanto al modelo de lenguaje, motor de *embeddings* y base de datos vectorial, obteniendo que tras la evaluación, se optó por el uso de **Python 3.12.6** como lenguaje principal por su ecosistema maduro en aprendizaje automático y procesamiento de lenguaje natural, **Streamlit** para la construcción de la interfaz web, **LangChain** como marco de orquestación, **Groq Cloud** con el modelo **Llama 3.3 70B Versatile** como motor de inferencia, el modelo

paraphrase-multilingual-MiniLM-L12-v2 para la generación de embeddings, **ChromaDB** como base de datos vectorial persistente y la biblioteca **rank-bm25** para el índice léxico complementario. Como entorno de desarrollo integrado se utilizó Cursor, y para el control de versiones, un repositorio en GitHub.

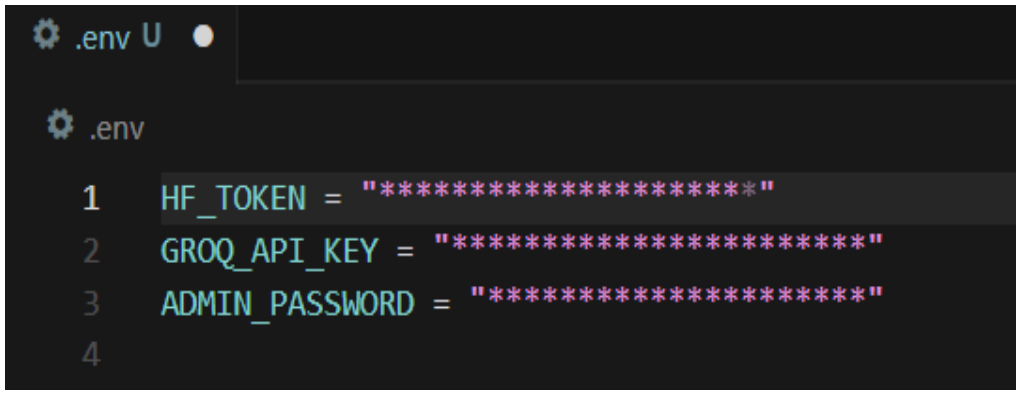


4.1.1 Estándares de codificación

Se establecieron las siguientes convenciones de codificación para el desarrollo del prototipo, basadas en PEP 8, el estándar oficial de estilo de código Python que establece convenciones sobre nomenclatura, espaciado, longitud máxima de línea y organización del código para asegurar legibilidad y consistencia (van Rossum et al., 2001). Estas convenciones son:

- Nombres de variables y funciones en español dentro del código de aplicación.
- Funciones en estilo snake_case siguiendo PEP 8, donde las palabras se separan con guiones bajos en lugar de mayúsculas intermedias (van Rossum et al., 2001). Por ejemplo, construir_bm25 en lugar de construirBM25.
- Constantes en mayúsculas con guiones bajos como separadores, también según PEP 8: GROQ_API_KEY, CHROMA_PATH, COLLECTION_NAME.
- Separación lógica del código en bloques temáticos comentados entre los cuales están, la configuración, recursos cacheados, helpers, limpieza, extracción, recuperación e interfaz.

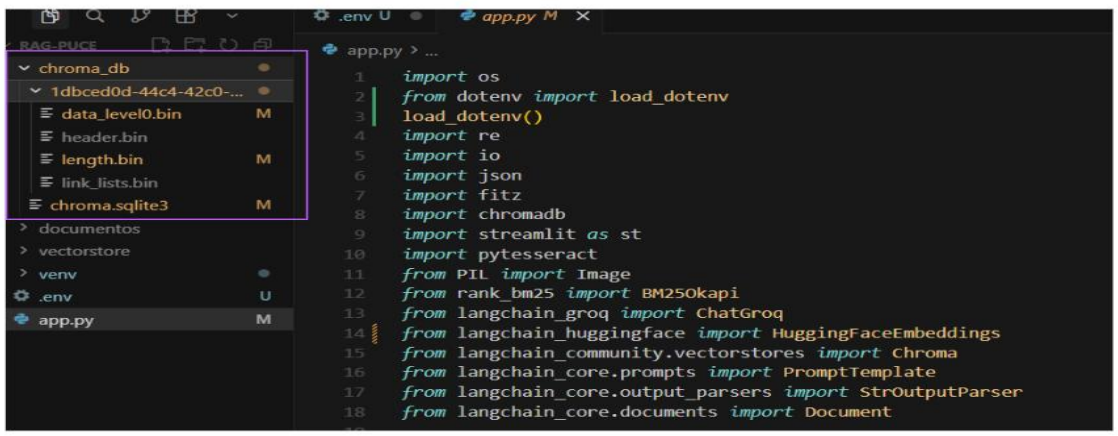
- Uso de variables de entorno gestionadas con python-dotenv para todas las credenciales sensibles, evitando exposición en el código fuente.



4.2 Implementación de la funcionalidad

4.2.1 Configuración de la base de datos vectorial

ChromaDB se configura como un cliente persistente, almacenando su contenido en una carpeta local denominada chroma_db, ubicada en el mismo directorio del archivo principal del aplicativo, por ende, la inicialización del cliente ChromaDB y del modelo de *embeddings* se lo coloca con @st.cache_resource, garantizando una única instancia activa durante la vida útil de la sesión. Este mecanismo implementa el patrón Cache-Aside descrito por Gamma et al. (1994), donde el recurso costoso se carga una única vez y se almacena en memoria para su reutilización en invocaciones posteriores, reduciendo la latencia de inicialización a partir de la segunda ejecución.



Por consiguiente, cada documento indexado en ChromaDB almacena tres campos de metadata de tipo cadena de caracteres (string): source (nombre del archivo PDF de origen), articulo (número del artículo) y titulo (título extraído). Aunque ChromaDB acepta enteros como valores de metadata, el número de artículo se almacena intencionalmente como cadena de caracteres para garantizar consistencia en las comparaciones de recuperación, evitando fallos por incompatibilidad de tipos entre el valor almacenado y el valor consultado por el *router*. Esta decisión es beneficiosa para el prototipo porque permite ejecutar filtrados directos sobre los campos de metadata durante la búsqueda, por ejemplo cuando el *router* detecta que el estudiante menciona un reglamento específico, el sistema pasa el filtro {"source": "reglamento-becas"} a la función de búsqueda vectorial, haciendo que ChromaDB descarte automáticamente los fragmentos del Reglamento General de Estudiantes antes de ejecutar la comparación de vectores, de este modo, la búsqueda opera únicamente sobre los 29 fragmentos del corpus relevante en lugar de los 107 totales, mejorando tanto la precisión de los resultados como la eficiencia de la operación.

Internamente, ChromaDB implementa búsqueda aproximada de vecinos más cercanos mediante el algoritmo HNSW (Grafo Jerárquico de Mundos Pequeños Navegable). El algoritmo organiza todos los vectores en un grafo de múltiples capas apiladas verticalmente. La capa inferior contiene la totalidad de los 107 vectores del corpus, conectados cada uno con sus vecinos más cercanos en el espacio vectorial, cada capa superior contiene un subconjunto reducido de esos mismos vectores, pero con conexiones que abarcan distancias mayores en el espacio, permitiendo desplazamientos más amplios entre zonas del grafo.

Cuando se ejecuta una búsqueda, el algoritmo comienza en la capa superior más escasa, donde las conexiones largas permiten recorrer rápidamente grandes porciones del espacio vectorial y llegar a la zona aproximada donde se encuentran los candidatos relevantes. Una vez identificada esa zona, el algoritmo desciende a la capa inferior, que tiene más nodos y conexiones más cortas, permitiendo realizar una búsqueda con mayor precisión, este proceso se repite capa por capa hasta llegar a la parte inferior donde están todos los vectores, momento en el que el algoritmo ya está operando en la zona correcta y puede identificar los vecinos más cercanos exactos con pocos pasos adicionales.

La ventaja de este enfoque es que evita comparar el vector de consulta contra la totalidad del corpus de forma exhaustiva, lo que tomaría tiempo proporcional al número de vectores almacenados. En cambio, la navegación jerárquica permite llegar a los candidatos relevantes en menor tiempo independientemente del tamaño de la colección (Malkov & Yashunin, 2018), siendo que,

esta eficiencia ha convertido a HNSW en el algoritmo de referencia para búsqueda vectorial en bases de datos como ChromaDB y Qdrant.

Por otra parte, la colección puce_normativa agrupa los *embeddings* de todos los reglamentos cargados. Cuando el administrador reemplaza el corpus, la colección previa se elimina explícitamente antes de la nueva indexación, garantizando la atomicidad del reemplazo.

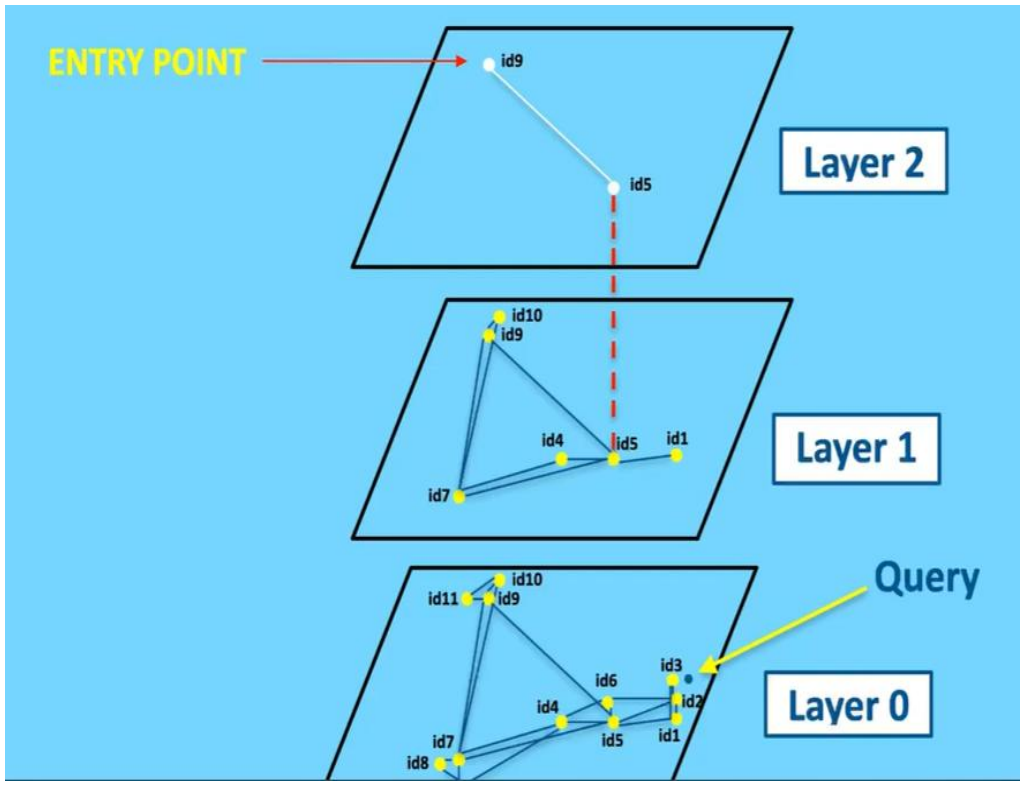


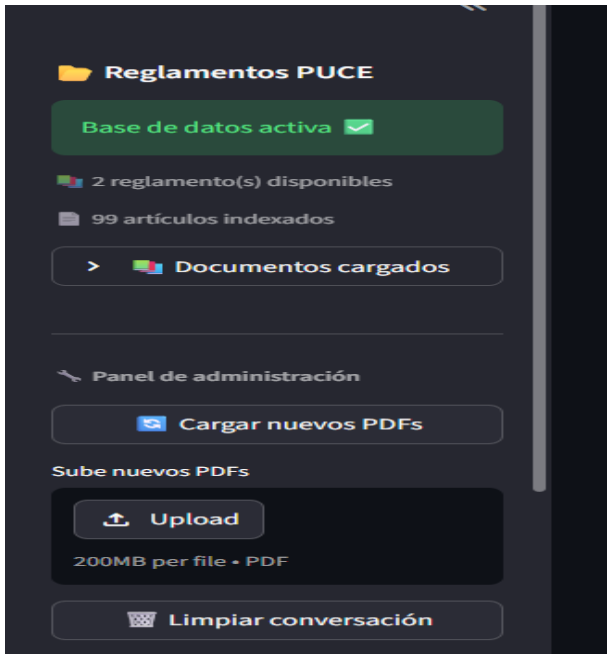
ILUSTRACIÓN 5 REPRESENTACIÓN DEL ALGORITMO HNSW CON SUS CAPAS JERÁRQUICAS Y PUNTO DE ENTRADA.

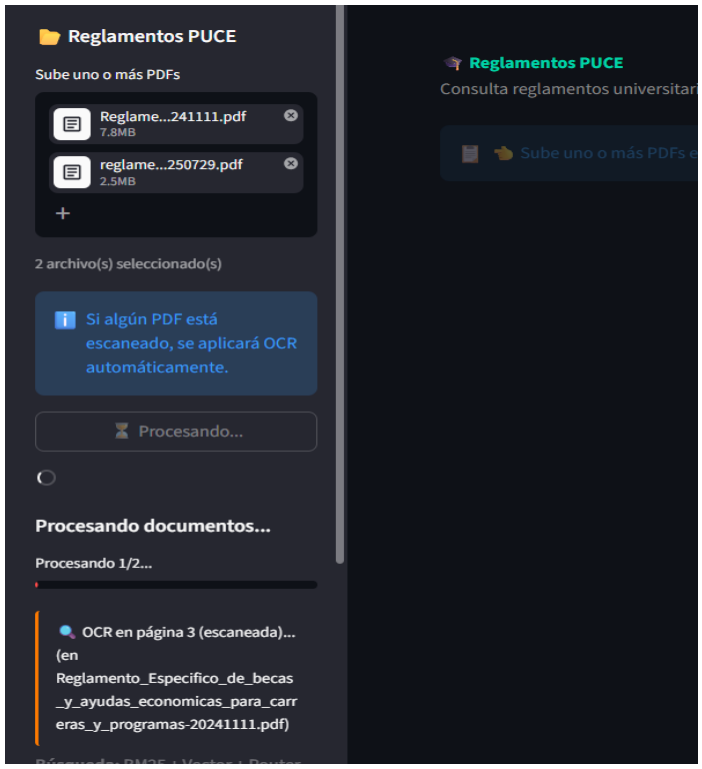
Nota. Adaptado de Hierarchical Navigable Small World (HNSW) en Chroma - Cómo funciona: Visualización del algoritmo, por APRENDER IA, 2024, YouTube (https://www.youtube.com/watch?v=b7XLTy72qrA). Captura de pantalla con fines académicos.

4.2.2 Motor de recuperación semántica

El motor de recuperación se implementa mediante tres componentes coordinados: el procesamiento de PDFs, la segmentación por artículos y la generación del índice BM25.

El procesamiento se lo hace de forma secuencial sobre cada archivo cargado, aplicando el método de extracción híbrida. Para cada documento, se registra en el panel lateral de Streamlit el progreso y el número de páginas que requirieron OCR, proceso mediante el cual una imagen digitalizada de texto es convertida a texto editable y sea procesable a nivel computacional.





La función dividir_en_articulos, aplcia una expresión regular sin sensibilidad a mayúsculas activada, garantizando que únicamente los encabezados reales (Artículo N.- Título) se interpreten como divisores. Una expresión regular es un patrón formal que describe la estructura que debe tener una cadena de texto para ser reconocida, técnica utilizada en procesamiento de texto. Después de la segmentación, cada fragmento se valida verificando que comience efectivamente con un encabezado, descartando falsos segmentos.

El índice BM25 (Best Matching 25) se construye en memoria sobre la lista completa de fragmentos textuales mediante la biblioteca rank-bm25. BM25 es un algoritmo de recuperación de información basado en el modelo probabilístico de relevancia desarrollado por Robertson y Sparck Jones desde la década de 1970 y formalizado por Robertson y Zaragoza (2009). A diferencia

de la búsqueda vectorial que compara significados, BM25 busca coincidencias de palabras exactas y asigna a cada fragmento del corpus un puntaje que refleja qué tan bien responde a la consulta del estudiante.

Para calcular ese puntaje, BM25 considera tres aspectos del fragmento evaluado. El primero es cuántas veces aparece la palabra buscada, si un fragmento menciona la palabra "beca" cinco veces, es más probable que sea relevante que uno donde aparece una sola vez. Sin embargo, BM25 aplica un límite de rendimiento decreciente, la segunda aparición aporta menos que la primera, y la décima aporta menos que la segunda. Esto impide que un fragmento que repite artificialmente la misma palabra domine los resultados por encima de fragmentos más informativos.

El segundo aspecto es qué tan rara o común es la palabra en el corpus completo. Una palabra que aparece en casi todos los fragmentos, como "estudiante" o "universidad", no sirve para distinguir cuál fragmento es más relevante porque está en todos. En cambio, una palabra que solo aparece en tres o cuatro fragmentos, como FOPEDEUPO o Beca San Ignacio, indica con mucha más precisión de qué trata cada fragmento. BM25 premia las palabras raras y resta importancia a las comunes.

El tercer aspecto es el largo del fragmento. Un fragmento de tres mil caracteres tiene más probabilidad de contener cualquier palabra simplemente por su volumen, no necesariamente porque sea el más relevante sobre ese tema. BM25 ajusta el puntaje teniendo en cuenta si el fragmento es más largo o corto que el promedio del corpus, de modo que un fragmento corto y preciso compita en igualdad de condiciones con uno largo y general.

Esta combinación hace que BM25 sea especialmente efectivo cuando el estudiante menciona términos exactos del reglamento, como el número de un artículo o un nombre propio institucional, complementando así la búsqueda vectorial que opera sobre similitud semántica y reconoce sinónimos, pero puede pasar por alto coincidencias léxicas exactas (Robertson & Zaragoza, 2009).

En este prototipo BM25 se implementa mediante la función construir_bm25, la cual opera sin el decorador @st.cache_resource de Streamlit. Esta decisión responde a un comportamiento específico del sistema de caché: Streamlit excluye del cálculo de la clave de caché los parámetros cuyos nombres comienzan con guion bajo, tratando todas las llamadas como idénticas independientemente del contenido real de los fragmentos recibidos. Esto generaba un índice BM25 desactualizado cuando el administrador reemplazaba el corpus, causando errores de índice fuera de rango durante la recuperación, por ende, al eliminar

el decorador, la función reconstruye el índice en cada invocación garantizando coherencia permanente entre el índice y el corpus activo. Este costo es mínimo dado que la reconstrucción toma menos de 50 milisegundos sobre el corpus actual, tiempo inferior al costo de lectura desde cualquier medio de almacenamiento secundario. En una versión de producción con múltiples usuarios concurrentes, el índice BM25 se implementaría como una instancia global del servidor construida una única vez al iniciar el servicio.

La función central recuperar_contexto integra los tres modos de recuperación: coincidencia exacta por número de artículo, búsqueda BM25 y búsqueda vectorial. Teniendo que, el *router* de intención se implementa como una invocación al modelo de lenguaje con un *prompt* especializado que solicita la clasificación en formato JSON, formato estándar de intercambio de datos estructurados especificado por la Internet Engineering Task Force en el RFC 8259 (Bray, 2017). A partir de la intención clasificada, el sistema ejecuta los tres motores de recuperación y reúne sus candidatos en una lista unificada. La fusión de estos candidatos se realiza sin normalización de escala entre BM25 y búsqueda vectorial, decisión adecuada para el corpus actual de 107 fragmentos donde el sesgo de escala tiene impacto mínimo en la calidad de los resultados.

Por otra parte, las consultas sobre la estructura del corpus se identifican mediante un conjunto de expresiones regulares y se atienden directamente desde la metadata en sesión, evitando la limitación fundamental de los sistemas RAG estándar que solo recuperan fragmentos *top-K* (Karpukhin et al., 2020).

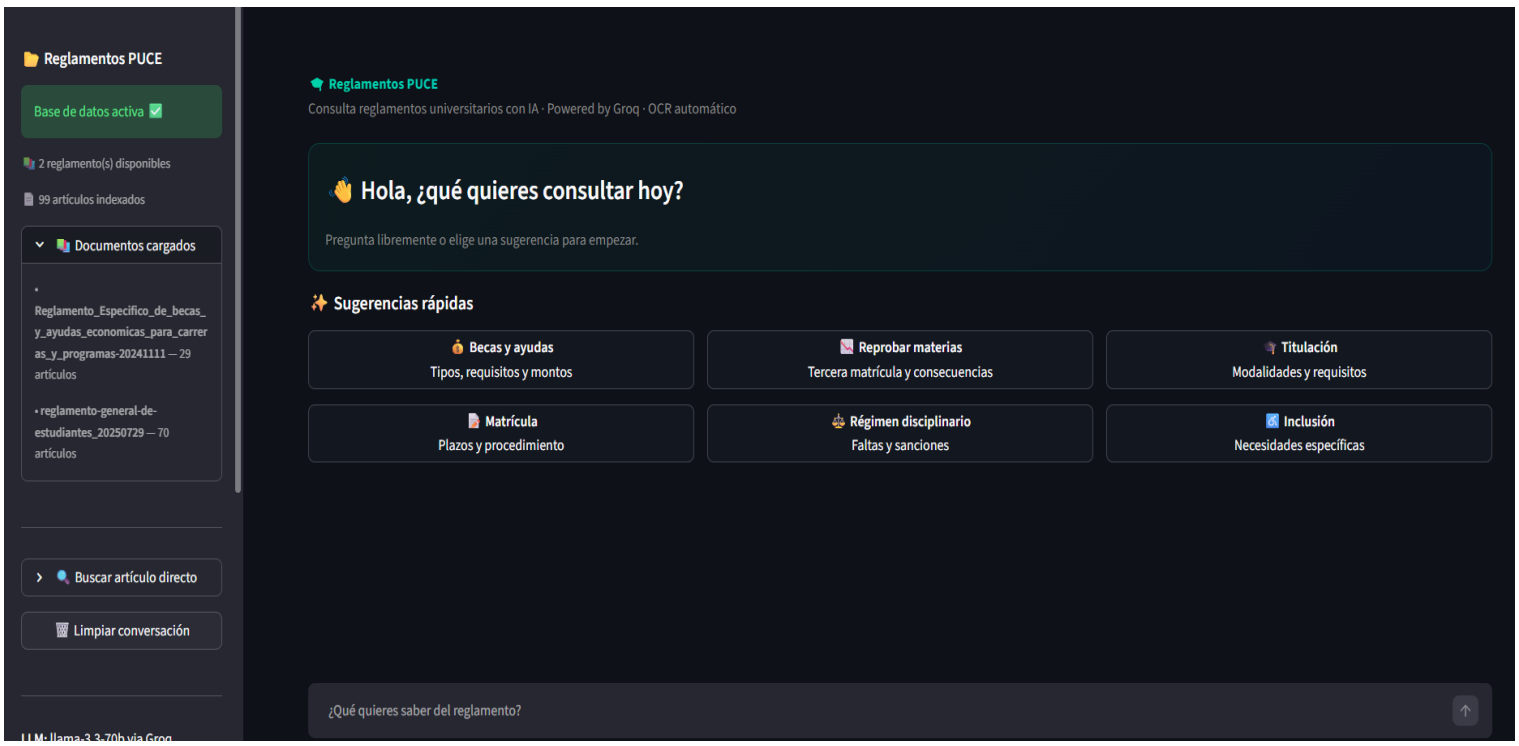
Finalmente, la generación de la respuesta final se realiza mediante un *prompt* estructurado con reglas obligatorias de citación que el modelo debe cumplir, incluyendo el formato [Art. N — Título — documento] y la indicación explícita de declarar la ausencia de información cuando los fragmentos no contengan datos suficientes. Así mimo, la configuración del parámetro temperature=0 va a permitir forzar al modelo a seleccionar siempre la continuación de mayor probabilidad en cada paso de generación, eliminando la aleatoriedad inherente al proceso y garantizando que consultas idénticas sobre el mismo contexto produzcan respuestas iguales, propiedad muy importante en un sistema de consulta normativa donde la consistencia documental es prioritaria (Wang et al., 2023).

4.2.3 Interfaz web con separación por rol

La interfaz web implementa una arquitectura de dos vistas diferenciadas por rol dentro de una única aplicación Streamlit. Esta decisión evita el mantenimiento de dos aplicaciones independientes, reduciendo la complejidad de despliegue sin sacrificar la separación funcional entre el perfil del estudiante y el del administrador.

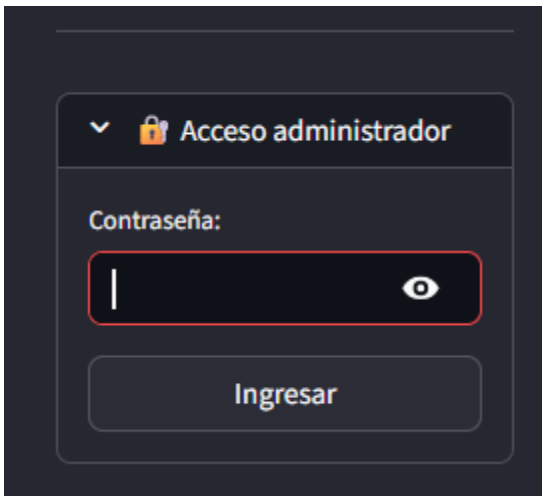
La separación se implementa mediante renderizado condicional sobre la variable *booleana* st.session_state.es_admin. Cuando su valor es False siendo este su estado por defecto, la interfaz va a presentar únicamente los componentes de consulta accesibles para el estudiante, sin embargo, cuando el valor de es_admin cambia a True tras una autenticación exitosa, se revelan adicionalmente los controles administrativos.

Interfaz Estudiante



Nota: Contiene las opciones básicas del prototipo en el *sidebar*

La autenticación del administrador se gestiona mediante un panel colapsado en la parte inferior del *sidebar*, donde se solicita la contraseña configurada en la variable de entorno ADMIN_PASSWORD. Tras la autenticación exitosa, el cambio de vista es inmediato gracias al mecanismo st.rerun() que dispara una re-ejecución completa del script con el nuevo estado de sesión.

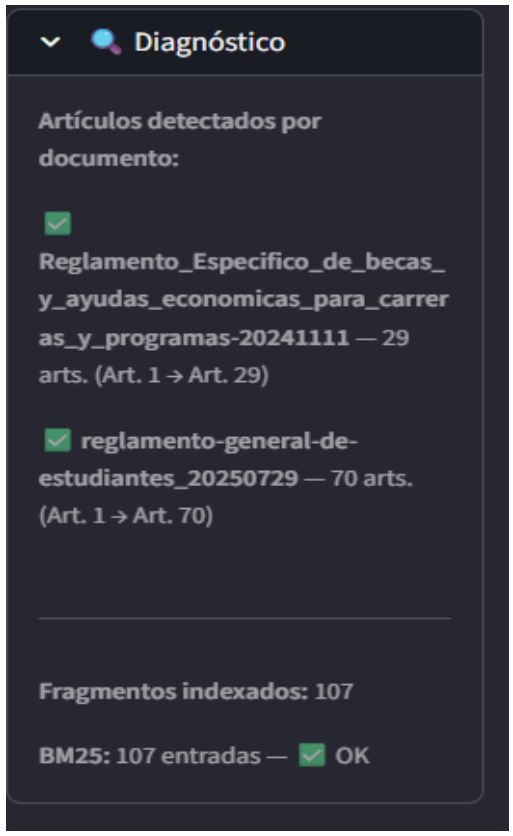


El panel de diagnóstico es visible únicamente para el administrador ya que expone métricas detalladas del corpus indexado: cantidad de artículos detectados por documento, rango numérico, posibles huecos en la numeración y estado del índice BM25.

Interfaz Admin



Nota: Contiene todas las opciones propuestas para el administrador en el *sidebar*.



4.3 Plan de pruebas

4.3.1 Tipos de pruebas

Probar software es el proceso mediante el cual se evalúa y verifica que el sistema en desarrollo cumpla con los criterios definidos al inicio del proyecto, además en ingeniería de software se reconocen múltiples tipos de pruebas, cada una orientada a verificar un aspecto distinto del sistema. Antes de justificar la estrategia adoptada, es conveniente definir cada tipo y explicar por qué resulta o no aplicable al contexto de un prototipo de investigación basado en arquitectura RAG.

- **Pruebas unitarias.** Verifican el comportamiento de funciones individuales de forma aislada, aunque el sistema contiene funciones de prueba unitaria, como dividir_en_articulos o construir_bm25, el valor de un sistema RAG no está en la corrección aislada de cada función sino en la calidad de la respuesta final generada por la interacción de todos los componentes. Una función de segmentación puede operar correctamente de forma aislada y aun así producir respuestas deficientes si el modelo de lenguaje interpreta mal el contexto recuperado.

- **Pruebas de integración.** Verifican que los módulos funcionen correctamente al combinarse. En el presente sistema, la integración entre la recuperación híbrida, el router de intención y el modelo generativo se valida implícitamente en las pruebas funcionales, dado que cada consulta ejecuta la totalidad del flujo integrado de extremo a extremo.
- **Pruebas funcionales.** Verifican que el sistema produzca las salidas esperadas ante entradas determinadas, evaluando su comportamiento sin examinar la implementación interna. Este tipo de prueba es el más adecuado para un sistema RAG porque permite evaluar directamente el atributo de calidad principal del sistema, el cual es, la fidelidad documental de las respuestas generadas.
- **Pruebas de extremo a extremo.** Validan flujos completos de usuario desde la entrada hasta la salida final. En el contexto de este prototipo, las pruebas funcionales sobre consultas representativas equivalen en sí mismas a pruebas de extremo a extremo, dado que cada consulta recorre todo el sistema desde la interfaz hasta la generación de la respuesta citada.
- **Pruebas de aceptación.** Verifican que el sistema cumpla con los requisitos del usuario final, estas pruebas requieren la participación de usuarios reales en un entorno de producción, condición que excede el alcance de un prototipo desplegado en entorno controlado para demostración técnica.
- **Pruebas de rendimiento.** Evalúan el comportamiento del sistema bajo carga, midiendo latencia y capacidad ante múltiples usuarios concurrentes, siendo que, el prototipo opera con un único usuario en entorno controlado y que el análisis de concurrencia se lo toma de forma teórica en la sección 4.5, las pruebas de rendimiento bajo carga real corresponden a una fase de producción posterior.
- **Pruebas de humo.** Verifican rápidamente que las funcionalidades críticas operen antes de pruebas más exhaustivas. Estas se realizaron de manera informal durante el desarrollo iterativo, pero no constituyen un protocolo formal documentable.

Dentro del presente prototipo se optó por implementar un protocolo de pruebas funcionales sobre un conjunto de consultas representativas, complementadas con verificación manual de las respuestas generadas. Esta decisión responde a las siguientes razones técnicas.

- **La naturaleza no determinista de los sistemas basados en modelos de lenguaje.** A diferencia del software tradicional, donde una entrada produce siempre una salida idéntica que puede verificarse automáticamente comparando el resultado obtenido con el resultado esperado, los sistemas RAG generan respuestas en lenguaje natural donde la corrección no puede determinarse

mediante esa misma comparación directa. Esto ocurre porque una misma pregunta puede tener múltiples respuestas válidas expresadas con palabras distintas, por ejemplo: "el artículo 7 establece la Beca San Ignacio" es equivalente a decir "según el artículo 7, existe la Beca San Ignacio". Por esta razón, las métricas tradicionales de aprendizaje automático como precisión, *recall* y F1, que miden qué tanto coinciden dos textos carácter por carácter, no son directamente aplicables a sistemas RAG ya que una respuesta puede ser semánticamente correcta y fiel al reglamento aunque sus palabras exactas contrasten de la respuesta de referencia, en consecuencia, para el alcance de este prototipo la verificación manual contra el documento original resulta la estrategia más adecuada, dado que solo un evaluador humano puede determinar si el contenido transmitido por la respuesta generada refleja fielmente lo que establece el reglamento, independientemente de cómo esté redactado.

- **La inviabilidad de las herramientas de evaluación automatizada en este alcance.** *Frameworks* especializados como RAGAS permitirían medir la fidelidad documental sin intervención humana continua en escenarios con corpus extenso. Sin embargo, la implementación de herramientas de evaluación automatizada para sistemas RAG, entre las que se encuentran RAGAS, TruLens y DeepEval, no resultaba viable en este prototipo por razones propias a la categoría del proceso, no a una herramienta específica. Estas herramientas comparten la misma arquitectura de evaluación, es decir, todas requieren construir previamente un conjunto de preguntas con sus respuestas correctas verificadas contra el documento original, conocida como conjunto de referencia, y todas utilizan un modelo de lenguaje como juez interno para calcular sus métricas semánticas.

No obstante, la primera limitación es que la construcción del conjunto de referencia escala directamente con el tamaño del corpus, a mayor número de documentos, mayor número de consultas representativas se requieren para cubrir adecuadamente la diversidad temática del sistema. Sobre los dos reglamentos actuales, ese esfuerzo es comparable al protocolo manual ya definido, por lo que estas herramientas no representarían una ventaja real en este alcance.

Herramientas como RAGAS ofrecen módulos de generación sintética que utilizan un modelo de lenguaje para producir automáticamente ese conjunto de referencia leyendo los documentos originales, sin embargo, esto conduce directamente a la segunda limitación. Estas herramientas utilizan un modelo de lenguaje como juez interno que realiza múltiples llamadas por cada consulta evaluada para calcular sus métricas semánticas, y la generación sintética del conjunto de referencia añade un costo adicional proporcional al volumen del corpus, en consecuencia, el consumo combinado de ambas etapas crece de forma

directamente proporcional al tamaño del corpus y al número de consultas evaluadas, haciendo inviable su ejecución sobre el plan gratuito de Groq sin migrar a un plan de pago con límites superiores.

- **La proporcionalidad respecto al alcance del proyecto.** El alcance definido en el Capítulo I corresponde a un prototipo funcional en entorno controlado, no a un sistema de producción. Implementar una infraestructura completa de pruebas automatizadas, suites de integración continua y pruebas de carga distribuida constituiría un esfuerzo desproporcionado frente al objetivo de demostrar la viabilidad técnica de la arquitectura RAG para la consulta normativa, siendo que la estrategia de pruebas debe ser proporcional a los riesgos y al alcance del sistema evaluado.

4.3.1.1 Pruebas funcionales sobre el conjunto de consultas representativas

Se construyó un conjunto de pruebas compuesto por treinta consultas distribuidas en cinco categorías que reflejan los casos de uso identificados:

TABLA 12 DISTRIBUCIÓN DE CONSULTAS DEL PROTOCOLO DE PRUEBAS POR CATEGORÍA

Categoría	Cantidad	Descripción
Consulta directa de artículo	10	Preguntas que referencian un número específico (CU-02)
Consulta temática	10	Preguntas sobre conceptos sin mencionar artículo (CU-03)
Meta-pregunta	5	Preguntas sobre la estructura del corpus (CU-04)
Pregunta fuera de alcance	3	Preguntas cuya respuesta no está en el corpus
Comparativa	2	Preguntas que requieren contraste entre reglamentos (CU-05)

4.3.2 Protocolo experimental detallado

El protocolo experimental se ejecuta con los siguientes pasos secuenciales que garantizan la consistencia y la trazabilidad de los resultados obtenidos.

4.3.2.1 Preparación del entorno de prueba

Antes de ejecutar las pruebas, se garantiza un estado limpio del sistema mediante el siguiente procedimiento:

1. Cerrar la aplicación si está en ejecución.
2. Verificar que la carpeta chroma_db/ contiene los dos reglamentos PUCE indexados.
3. Iniciar la aplicación con el comando streamlit run app.py.
4. En el navegador, verificar en el panel lateral que aparezcan los dos reglamentos cargados y que se muestre el diagnóstico.

4.3.2.2 Conjunto completo de las 30 consultas de prueba

A continuación, se presenta el conjunto completo de las 30 consultas que deben ejecutarse.

Categoría A. Consulta directa de artículo (10 consultas)

TABLA 13 CONSULTAS DE LA CATEGORÍA A: CONSULTA DIRECTA DE ARTÍCULO

#	Consulta
A1	¿qué dice el artículo 1 del reglamento de becas?
A2	art. 7
A3	¿qué dice el artículo 29 del reglamento de becas?
A4	art. 15
A5	¿cuál es el contenido del artículo 22?
A6	dime el art. 5
A7	artículo 11 del reglamento general
A8	quiero ver el artículo 50 del general
A9	¿qué dice el art. 70?
A10	art 100

Categoría B — Consulta temática (10 consultas)

TABLA 14 CONSULTAS DE LA CATEGORÍA B: CONSULTA TEMÁTICA

#	Consulta
B1	¿qué pasa si pierdo mi beca?
B2	requisitos para becas
B3	¿cómo me título?
B4	qué es FOPEDEUPO
B5	¿qué pasa si repruebo una materia?
B6	normas sobre discapacidad
B7	matrícula extraordinaria
B8	aranceles
B9	sanciones disciplinarias
B10	derechos del estudiante

Categoría C — Meta-preguntas (5 consultas)

TABLA 15 CONSULTAS DE LA CATEGORÍA C: META-PREGUNTAS

#	Consulta
C1	¿cuántos artículos tiene el reglamento de becas?
C2	¿cuántos reglamentos están cargados?
C3	lista los artículos del reglamento general
C4	índice del reglamento de becas
C5	¿qué documentos están cargados?

Categoría D — Pregunta fuera de alcance (3 consultas)

TABLA 16 — CONSULTAS DE LA CATEGORÍA D: PREGUNTAS FUERA DE ALCANCE

#	Consulta
D1	¿cómo cocino arroz con pollo?
D2	¿quién es el presidente del Ecuador?
D3	¿cuándo fundó la universidad?

Categoría E — Comparativa (2 consultas)

TABLA 17 CONSULTAS DE LA CATEGORÍA E: CONSULTAS COMPARATIVAS

#	Consulta
E1	diferencia entre el Art. 28 del reglamento de becas y el Art. 28 del general
E2	compara el Art. 7 del reglamento de becas con el Art. 7 del general

4.3.2.3 Evaluación de cada respuesta

Cada respuesta se evalúa según cuatro criterios binarios (sí/no):

TABLA 18 CRITERIOS DE EVALUACIÓN DEL PROTOCOLO DE PRUEBAS

Criterio	**Pregunta de evaluación**
Acierto documental	¿La respuesta proviene efectivamente de los reglamentos cargados?
Citación correcta	¿La respuesta incluye la cita en formato [Art. N — Título — documento]?

Declaración de ausencia	Solo para categoría D. ¿El sistema declara explícitamente la ausencia sin fabricar contenido?
Latencia aceptable	¿La latencia es menor o igual a 10 segundos?

4.3.2.4 Registro de resultados

Los resultados se registran en una hoja de cálculo con la siguiente estructura recomendada:

Categoría A. Consulta directa de artículo

TABLA 19 RESULTADOS DE LA CATEGORÍA A: CONSULTA DIRECTA DE ARTÍCULO

#	Categoría	Consulta	Acierto	Citación	Ausencia	Latencia (s)	Observaciones
1	Artículo	¿qué dice el artículo 29 del reglamento de becas?	Ok	Ok	N/A	4.4	Recurso de revisión, correcto.
2	Artículo	art. 7	Ok	Ok	N/A	4.6	Beca San Ignacio, transcripción fiel.
3	Artículo	art 100	N/A	N/A	Ok	3.1	Declaración de ausencia, sin alucinación.
4	Artículo	¿qué dice el art. 70?	Ok	Ok	N/A	3.3	Ceremonia de incorporación, correcta.
5	Artículo	quiero ver el artículo 50 del general	Ok	Ok	N/A	2.8	Identifica y cita correctamente el Art. 50, pero no reproduce la escala de equivalencias porque está incrustada como imagen, no

							como texto. Limitación del documento fuente.
6	Artículo	artículo 11 del reglamento general	Ok	Ok	N/A	3.3	Filtro respetado entre los dos documentos.
7	Artículo	dime el art. 5	Ok	Ok	N/A	4.3	Reproduce los dos Art. 5 con encabezado por documento.
8	Artículo	¿cuál es el contenido del artículo 22?	Ok	Ok	N/A	4	Segmentación limpia. Publicación web y Categorías.
9	Artículo	¿qué dice el artículo 1 del reglamento de becas?	Ok	Ok	N/A	3.3	Extracción precisa del objeto normativo del corpus.
10	Artículo	art. 15	Ok	Ok	N/A	4.7	Desambiguación fuerte entre los artículos Periodicidad y Título.

Categoría B — Consulta temática

TABLA 20 RESULTADOS DE LA CATEGORÍA B: CONSULTA TEMÁTICA

#	Categoría	Consulta	Acierto	Citación	Ausencia	Latencia (s)	Observaciones
11	Temática	¿qué pasa si pierdo mi beca?	Ok	Ok	N/A	5.3	Recupera Art. 17, 25 y 29 sobre pérdida y recurso de revisión. El Art. 17 es específico de la Beca Familia.

12	Temática	requisitos para becas	Ok	Ok	N/A	3.4	Agrupa San Ignacio - Art. 7, Familia PUCE - Art. 13-14 y otorgamiento - Art. 21. Transcripción fiel de requisitos.
13	Temática	¿cómo me título?	Ok	Parcial	N/A	3.9	Mezcla niveles: ancla en Art. 66 (posgrado) en vez de Art. 64 (pregrado) y cierra citando Art. 6 (admisión). Limitación de ambigüedad, no de recuperación.
14	Temática	qué es FOPEDEUPO	Ok	Ok	N/A	3.2	Inferencia controlada sin alucinar, deduce el concepto de los Art. 7 y 8.
15	Temática	¿qué pasa si repruebo una materia?	Ok	Ok	N/A	3.3	Tercera Matrícula - Art. 51, tres reprobaciones, cambio de carrera y asignatura misional.
16	Temática	normas sobre discapacidad	Ok	Ok	N/A	4.7	Cita correctamente Art. 27, 25, 26 del reglamento general y 11, 7 del de becas.
17	Temática	matrícula extraordinaria	Ok	Ok	N/A	7.6	Plazo de 15 días laborables e incremento de valor - Art. 18, enlazado con aumento de asignaturas - Art. 21.
18	Temática	aranceles	Ok	Ok	N/A	12.4	Definición - Art. 3, más cinco menciones cruzadas, todas

							correctas. Latencia sobre 10 s por respuesta extensa.
19	Temática	sanciones disciplinarias	Ok	Ok	N/A	4.4	Extrae menciones reales - Art. 24, 41 del reglamento general y 25 del de becas, sin inventar un régimen ausente del corpus.
20	Temática	derechos del estudiante	Ok	Ok	N/A	9.4	Lista completa y exacta de los cuatro derechos del Art. 35.

Categoría C — Meta-preguntas

TABLA 21 RESULTADOS DE LA CATEGORÍA C: META-PREGUNTAS

#	Categoría	Consulta	Acierto	Citación	Ausencia	Latencia (s)	Observaciones
21	Meta-pregunta	¿cuántos artículos tiene el reglamento de becas?	Ok	N/A	N/A	—	Reporta 29 artículos siendo estos desde el Art. 1 al 29. Coincide exactamente con el PDF. Sin RAG, respuesta desde metadata.
22	Meta-pregunta	¿cuántos reglamentos están cargados?	Ok	N/A	N/A	—	Reporta 2 reglamentos con conteo correcto: 29 y 70 artículos.
23	Meta-pregunta	lista los artículos del reglamento general	Ok	N/A	N/A	—	Lista los 70 artículos del general (Art. 1 al 70) con títulos correctos verificados contra el PDF.

24	Meta-pregunta	índice del reglamento de becas	Ok	N/A	N/A	—	Lista los 29 artículos de becas con títulos correctos. Art. 11 truncado "Becas y ayudas económicas para el cumplimiento de la Ley Orgánica de", por el límite de 150 caracteres en metadata, no es error de contenido.
25	Meta-pregunta	¿qué documentos están cargados?	Ok	N/A	N/A	—	Lista los dos documentos con sus conteos correctos.

Categoría D — Pregunta fuera de alcance

TABLA 22 RESULTADOS DE LA CATEGORIA D: PREGUNTAS FUERA DE ALCANCE

#	Categoría	Consulta	Acierto	Citación	Ausencia	Latencia (s)	Observaciones
26	Fuera de alcance	¿cómo cocino arroz con pollo?	N/A	N/A	Ok	2.8	Declara ausencia sin alucinar. Cita [No aplica - No hay artículo relevante], formato propio del sistema.
27	Fuera de alcance	¿quién es el presidente del Ecuador?	N/A	N/A	Ok	3.4	Declara ausencia correctamente. Cita ambos documentos como fuente negativa.
28	Fuera de alcance	¿cuándo fundó la universidad?	N/A	N/A	Ok	3.6	Declara ausencia correctamente. Mismo patrón de cita negativa. Sin alucinación de fecha.

Categoría E — Comparativa

TABLA 23 RESULTADOS DE LA CATEGORÍA E: CONSULTAS COMPARATIVAS

#	Categoría	Consulta	Acierto	Citación	Ausencia	Latencia (s)	Observaciones
29	Comparativa	diferencia entre el Art. 28 de becas y el Art. 28 del general	Ok	Ok	N/A	4.8	Recupera y contrasta ambos Art. 28: Promedio de Rendimiento del reglamento de becas vs Becas y ayudas económicas del reglamento general. Citas separadas por documento.
30	Comparativa	compara el Art. 7 de becas con el Art. 7 del general	Ok	Ok	N/A	5.1	Recupera y contrasta ambos Art. 7: Beca San Ignacio del reglamento de becas vs Reserva de cupo del reglamento general. Citas separadas y resumen comparativo correcto.

4.3.2.5 Cálculo de métricas agregadas

Una vez completadas las 30 consultas, se calculan las métricas por categoría con la siguiente fórmula:

Tasa de acierto por categoría = (Aciertos / Total consultas categoría) × 100

Latencia media = Σ(latencias) / Total consultas

4.3.3 Criterios de aceptación

Los criterios de aceptación son condiciones propuestas para que el producto final sea aceptado, considerando los requisitos funcionales y no funcionales definidos. Un planteamiento correcto de estos criterios ayuda a evitar confusiones y resultados inesperados.

Para las pruebas del prototipo se plantearon los siguientes criterios de aceptación, derivados de los requisitos no funcionales:

- **Tasa de acierto documental ≥ 90%:** Al menos nueve de cada diez respuestas deben provenir efectivamente de los reglamentos cargados, verificable manualmente contra el PDF original.
- **Tasa de citación correcta ≥ 90%:** Al menos noventa de cada cien respuestas deben incluir la cita en el formato [Art. N — Título — documento] y referenciar el artículo efectivamente utilizado.
- **Tasa de declaración de ausencia = 100% para preguntas fuera de alcance:** Toda pregunta sin respuesta en el corpus debe recibir la declaración explícita de ausencia, sin fabricación de contenido (alucinaciones).
- **Latencia media ≤ 10 segundos:** El tiempo de respuesta promedio debe mantenerse por debajo de diez segundos para consultas estándar.
- **Cobertura de meta-preguntas = 100%:** Toda meta-pregunta correctamente formulada debe ser detectada por las expresiones regulares y atendida sin invocar al modelo generativo.

4.3.3.1 Plantilla de resultados experimentales

TABLA 24 RESUMEN DE RESULTADOS POR CATEGORÍA

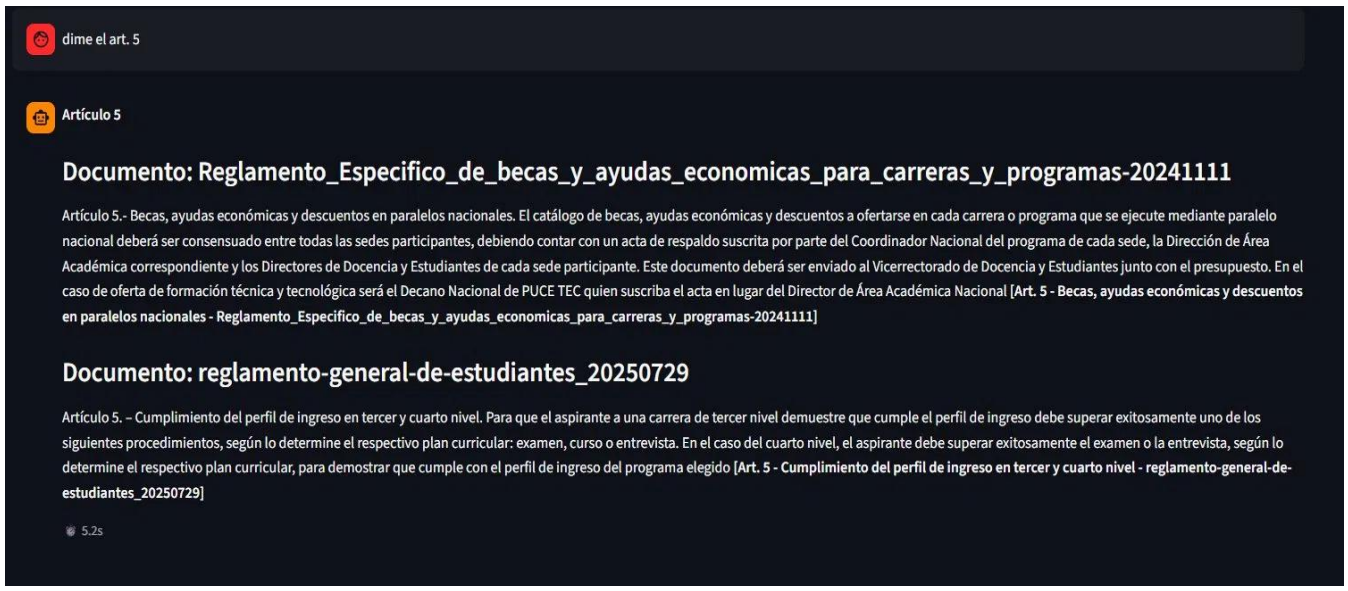
Categoría	**Acierto Documental**	**Citación Correcta**	**Declaración de Ausencia**	**Latencia Media (s)**
Consulta directa de artículo (10)	100 %	100 %	N/A	4.03
Consulta temática (10)	100 %	90 %	N/A	5.76

Meta-pregunta (5)	100 %	N/A	N/A	< 1
Pregunta fuera de alcance (3)	N/A	N/A	100 %	3.27
Comparativa (2)	100 %	100 %	N/A	4.95
Total (30)	**100 %**	**93.3 %**	**100 %**	**4.12**

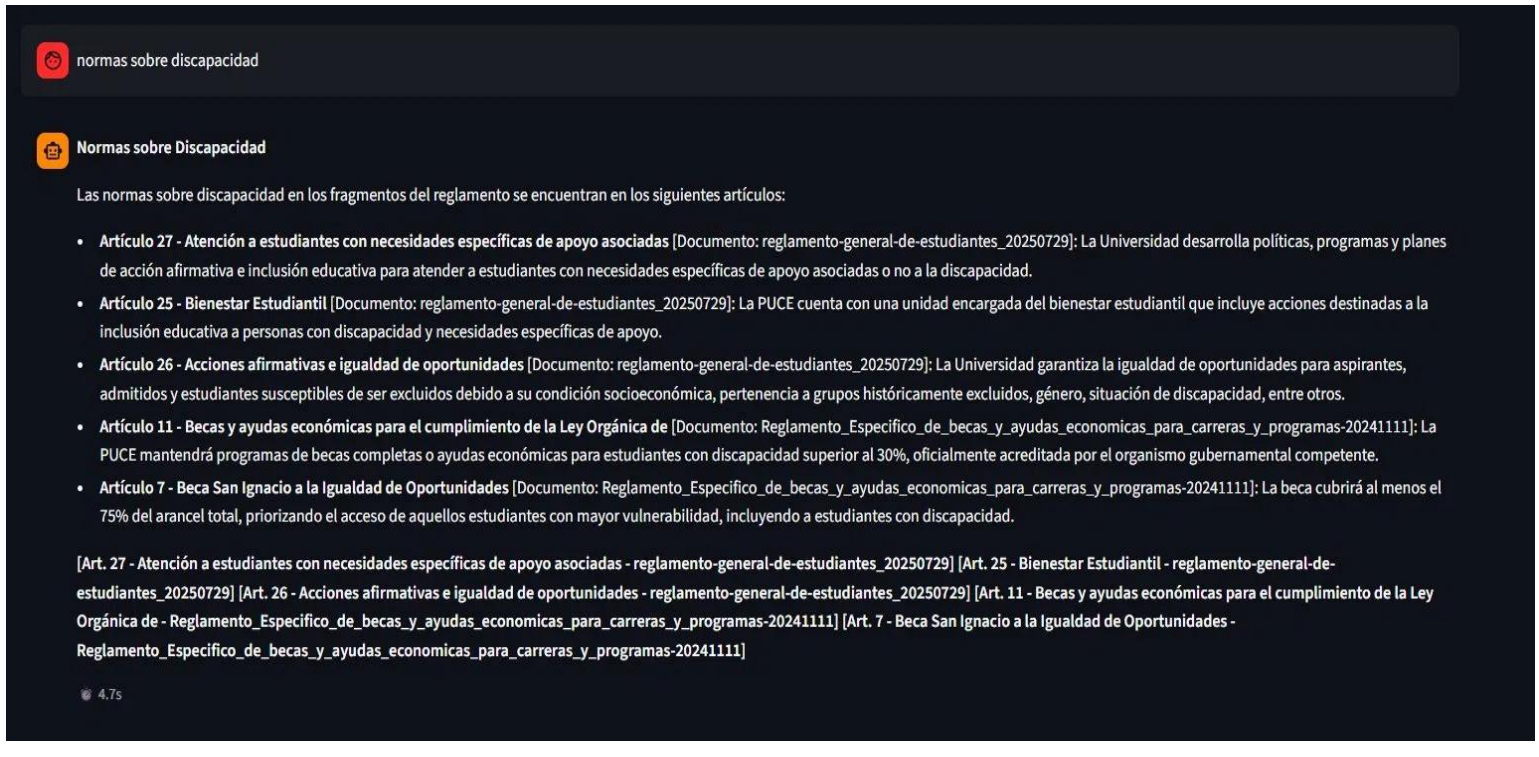
4.3.4 Ejemplos representativos de respuestas

A continuación, se presentan ejemplos representativos de cada categoría con su respuesta real generada por el prototipo.

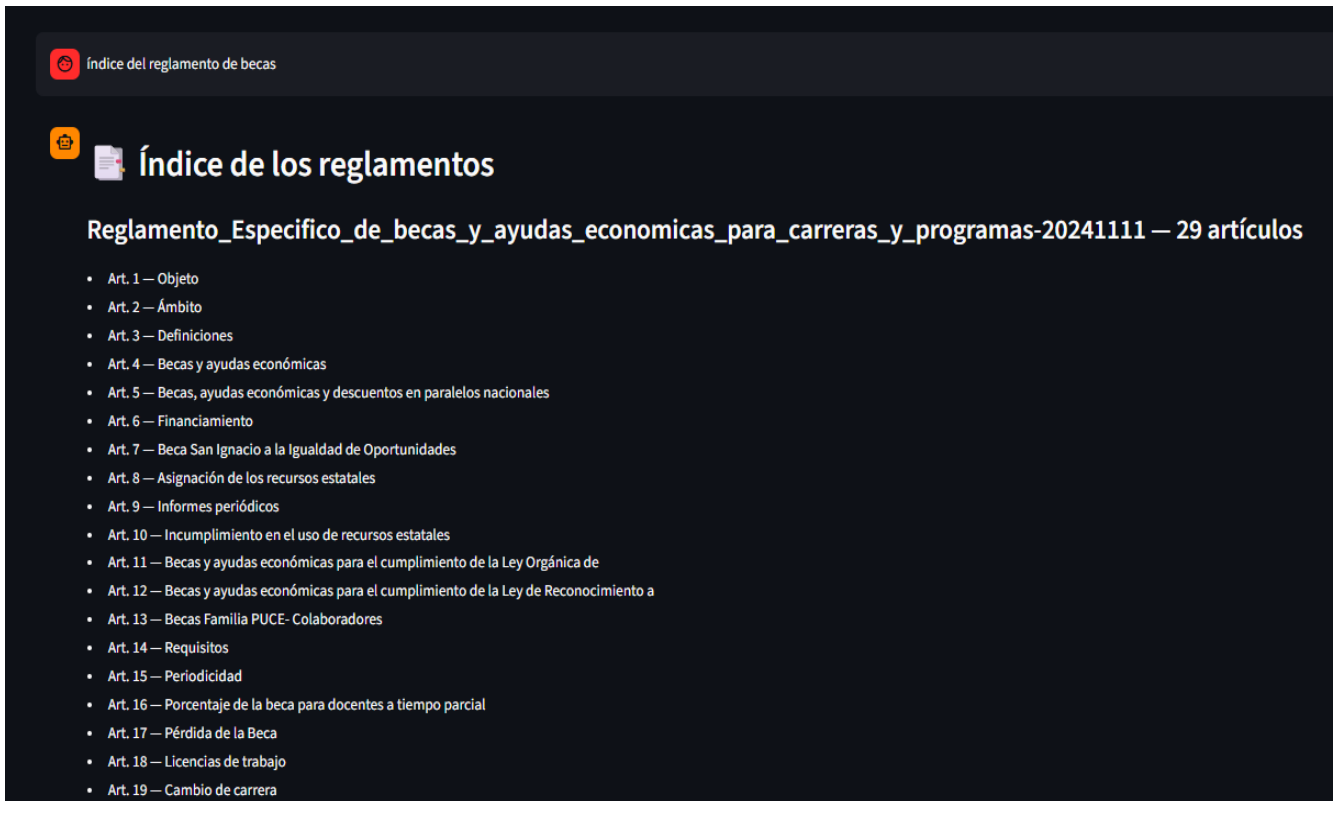
Ejemplo 1 — Consulta directa de artículo



Ejemplo 2 — Consulta temática

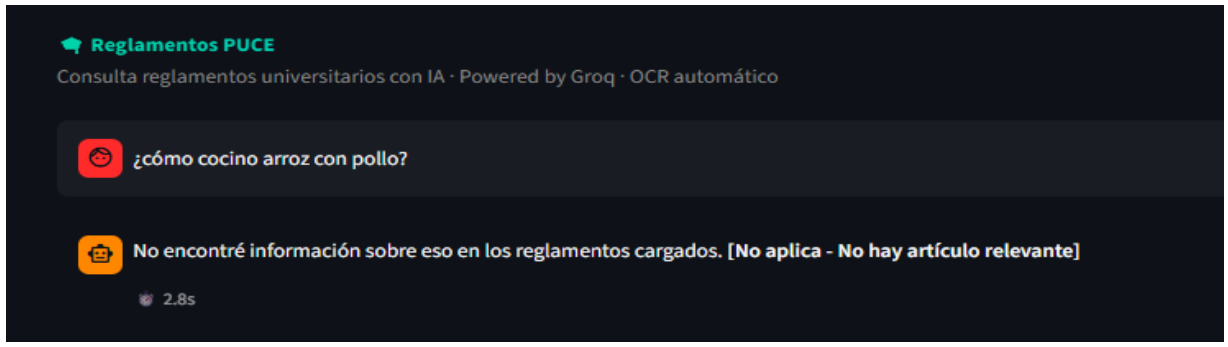


Ejemplo 3 — Meta-pregunta (CU-04)

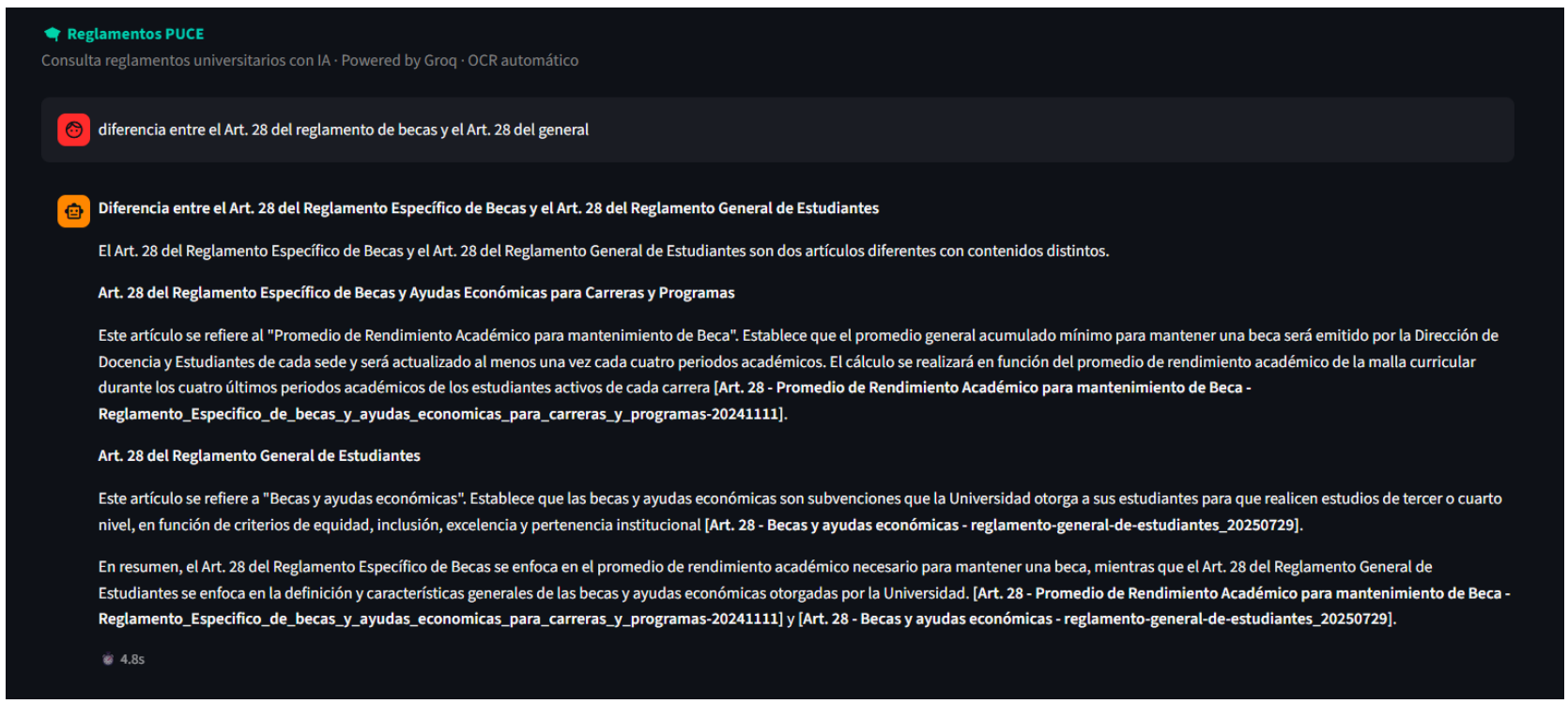


- Art. 19 — Cambio de carrera
- Art. 20 — Becas y descuentos adicionales
- Art. 21 — Otorgamiento
- Art. 22 — Publicación
- Art. 23 — Obligaciones de los becarios
- Art. 24 — Veracidad de la información y documentación
- Art. 25 — De la perdida de la beca
- Art. 26 — Alertas de riesgo académico
- Art. 27 — Tutorías Integrales
- Art. 28 — Promedio de Rendimiento Académico para mantenimiento de Beca
- Art. 29 — Recurso de revisión

Ejemplo 4 — Pregunta fuera de alcance



Ejemplo 5 — Comparativa



4.3.5 Casos límite identificados

Durante el desarrollo iterativo se identificaron algunos casos límite que orientaron mejoras sucesivas del prototipo y estos son:

- **Caso límite 1. Sobre detección de artículos en el considerando.** La presencia de referencias a leyes externas en el preámbulo Ley Orgánica de Educación Superior, Constitución, entre otras provocaba el conteo incorrecto de artículos. Antes de la corrección, el sistema reportaba más artículos de los que realmente contenía cada reglamento porque indexaba referencias como "el artículo 30 de la LOES", siendo que, esto se resolvió mediante el recorte del cuerpo normativo y la eliminación de la sensibilidad a mayúsculas en las expresiones regulares de segmentación.
- **Caso límite 2. Subconteo en respuestas de meta pregunta.** La limitación existente en los algoritmos de recuperación *top-K* provocaba que el modelo respondiera erróneamente sobre la cantidad total de artículos, ya que solo veía los 18 fragmentos recuperados, no la totalidad del corpus. Y esto se lo resolvió mediante la implementación de la ruta de meta preguntas que opera sobre la meta data completa sin invocar al modelo generativo.
- **Caso límite 3. Inconsistencia del índice BM25 tras reprocesar.** El sistema de caché de Streamlit retornaba un índice obsoleto cuando se cargaba un nuevo conjunto documental con cantidad diferente de fragmentos, generando errores de índice fuera de rango. Se resolvió mediante la eliminación del decorador de caché en la función construir_bm25 y la incorporación de verificación automática de coherencia entre el tamaño del índice y los *chunks* en sesión.
- **Caso límite 4. Tabla del Art. 50 incrustada como imagen.** El Artículo 50 del Reglamento General de Estudiantes contiene la escala de equivalencias de calificaciones en formato de tabla. Durante las pruebas se detectó que el sistema citaba correctamente el artículo, pero no reproducía los valores de la escala (A/B/C/D con sus rangos porcentuales), por lo que, el diagnóstico reveló que la tabla no existe como texto seleccionable en el PDF, es decir, está incrustada como imagen rasterizada en una página que por lo demás contiene texto nativo. En consecuencia, ni la extracción de texto con PyMuPDF ni la detección estructural de tablas mediante find_tables() pueden recuperar su contenido, y la heurística de OCR por página activada únicamente cuando el texto nativo es inferior a 50 caracteres no se dispara porque la página supera ampliamente ese umbral con 2.128 caracteres extraíbles. El sistema identifica y cita correctamente el artículo, pero la respuesta omite los valores de la tabla. Esta limitación es atribuible al formato del documento fuente y no a un defecto de la arquitectura RAG. Como trabajo futuro se propone implementar

OCR selectivo sobre imágenes incrustadas en páginas de texto, activado cuando find_tables() devuelve resultado vacío y la página contiene imágenes de dimensiones compatibles con una tabla.

- **Caso límite 5. Ambigüedad de nivel en consultas de titulación.** La consulta "¿cómo me título?" sin especificar nivel académico provocaba que el sistema anclara la respuesta en el Artículo 66 que es de maestrías con trayectoria de investigación en lugar del Artículo 64 título de tercer nivel, que corresponde a la población objetivo del prototipo
- Este comportamiento se debe a que la recuperación híbrida trae fragmentos de múltiples niveles y el modelo generativo selecciona el artículo de mayor densidad semántica, que en este caso era el de posgrado, a diferencia de los casos anteriores, esta limitación no responde a un defecto mecánico sino a la ambigüedad inherente de la consulta ya que el sistema no puede inferir el nivel académico del usuario sin información adicional. Como trabajo futuro se propone incorporar una pregunta de desambiguación cuando la intención clasificada sea "titulación" sin nivel especificado, o bien mejorar el *prompt* con instrucciones que prioricen el tercer nivel como nivel por defecto para la población objetivo.

4.4 Documentación técnica del sistema

La documentación técnica se tiene como un artefacto vivo, alineado con la fase de despliegue de CRISP-ML(Q). Su objetivo es habilitar la reproducción del entorno de ejecución por parte de un tercero.

4.4.1 Estructura del repositorio

```
rag-puce/
├── app.py              Aplicación principal Streamlit
├── .env                Variables de entorno (no versionado)
├── .gitignore          Exclusión de chroma_db, .env, venv/
├── requirements.txt    Dependencias Python
└── chroma_db/          Base de datos vectorial persistente (autogenerada)
```

El control de versiones del proyecto se gestiona mediante Git, sistema de control de versiones distribuido, alojado en una plataforma colaborativa como GitHub.

4.4.2 Variables de entorno y configuración

El archivo .env debe contener las credenciales requeridas para invocar a los servicios externos y para la autenticación del administrador:

```
GROQ_API_KEY=gsk_...
HF_TOKEN=hf_...
ADMIN_PASSWORD=clave_segura
```

La carga de estas variables se efectúa al inicio del programa mediante dotenv.load_dotenv(), antes de cualquier acceso a os.getenv. Esta separación garantiza que las credenciales no se persistan en el código fuente y permite su rotación sin modificaciones del aplicativo.

4.4.3 Dependencias principales

TABLA 25 DEPENDENCIAS Y VERSIONES DEL PROTOTIPO

Paquete	**Versión**	**Propósito**
streamlit	1.57.0	Interfaz web
chromadb	1.5.9	Base de datos vectorial
langchain-core	1.3.3	Núcleo de orquestación
langchain-community	0.4.1	Integración con ChromaDB
langchain-groq	1.1.2	Cliente para Groq Cloud
langchain-huggingface	1.2.2	Adaptador HuggingFace
sentence-transformers	5.4.1	Modelo de *embeddings*
rank-bm25	0.2.2	Recuperación léxica
pymupdf	1.27.2	Extracción y rasterización PDF
pytesseract	0.3.13	Reconocimiento óptico de caracteres

pillow	12.2.0	Procesamiento de imágenes para OCR
python-dotenv	1.2.2	Gestión de variables de entorno

4.5 Análisis de viabilidad técnica para implementación institucional

La presente sección evalúa la viabilidad técnica del despliegue del prototipo en un entorno de producción institucional, considerando dos restricciones fundamentales del contexto de la Pontificia Universidad Católica del Ecuador siendo la primera la infraestructura disponible y la segunda es que no se contempla una inversión adicional en hardware especializado. A partir de estas restricciones se analiza la carga esperada del sistema, se propone una arquitectura viable con la infraestructura existente y, posteriormente, se plantea de manera hipotética una arquitectura óptima que maximizaría el rendimiento de cada componente.

4.5.1 Estimación de la carga del sistema

El primer paso para determinar la viabilidad consiste en estimar la carga real que el sistema debería soportar, por lo que, es fundamental distinguir entre la población total de usuarios y la concurrencia efectiva, ya que ambos valores contrastan fuertemente en sistemas de consulta documental.

La población objetivo se estima en más de 5.000 estudiantes de pregrado. Sin embargo, la totalidad de los estudiantes no accede al sistema de forma simultánea, teniendo que, el patrón de uso de un asistente de consulta normativa es eventual y concentrado en periodos específicos del calendario académico, tales como el inicio de matrículas, las fechas de postulación a becas y los periodos de evaluación. Bajo este patrón, la concurrencia efectiva se estima de la siguiente manera:

TABLA 26 MÉTRICAS ESTIMADAS DE USO INSTITUCIONAL

Métrica	**Valor estimado**	**Justificación**
Estudiantes totales	5.000+	Población objetivo de pregrado
Consultas por estudiante al mes	3 a 5	Uso eventual orientado a dudas puntuales

Consultas mensuales totales	~20.000 a 25.000	Producto de población por frecuencia
Concurrencia promedio	5 a 15 usuarios	Distribución del uso a lo largo del día
Concurrencia pico	50 a 100 usuarios	Periodos de matrícula y becas

Esta estimación de concurrencia pico de entre 50 y 100 usuarios simultáneos constituye el parámetro de diseño sobre el cual se evalúa la capacidad de hardware necesaria.

4.5.2 Distribución de la carga computacional por componente

El factor clave para la viabilidad del proyecto sobre infraestructura limitada está en la distribución de la carga computacional entre los componentes del sistema, siendo que, el componente más exigente en términos de cómputo es el modelo de lenguaje de gran escala, no obstante, este no se ejecuta sobre la infraestructura institucional, sino que se delega a un servicio externo especializado como es el caso de Groq Cloud.

La siguiente tabla detalla dónde se procesa cada componente y qué recursos consume del servidor institucional.

TABLA 27 UBICACIÓN DEL PROCESAMIENTO POR COMPONENTE

Componente	**Dónde se procesa**	**Carga en el servidor PUCE**	**Recurso requerido**
Modelo de lenguaje Llama 3.3 70B	Groq Cloud	Ninguna	N/A
Generación de *embeddings* (MiniLM)	Servidor PUCE	Ligera	CPU + ~500 MB RAM
Búsqueda vectorial (ChromaDB)	Servidor PUCE	Ligera	~200 MB RAM
Búsqueda léxica (BM25)	Servidor PUCE	Mínima	~50 MB RAM
Servidor de aplicación web	Servidor PUCE	Moderada	CPU

Dado que la inferencia del modelo de lenguaje, la operación computacionalmente más costosa se procesa fuera de la infraestructura universitaria, el servidor institucional únicamente debe atender operaciones de bajo costo computacional, las cuales son la vectorización de consultas mediante un modelo ligero de 384 dimensiones, la búsqueda sobre un índice vectorial reducido y la coordinación de peticiones HTTP.

4.5.3 Escenario viable con la infraestructura existente

Considerando que la PUCE dispone únicamente de servidores basados en CPU y que no se contempla adquisición de hardware especializado, se propone la siguiente arquitectura de despliegue como una solución viable.

Decisión arquitectónica central: la inferencia del modelo de lenguaje esta delegada a Groq Cloud. Esta decisión no responde únicamente a un criterio de rentabilidad, sino a una restricción técnica inevitable. Dado que, la ejecución local del modelo Llama 3.3 70B sobre CPU produciría latencias del orden de varios minutos por consulta, ya que los modelos de esta escala requieren el paralelismo que solo proveen los aceleradores gráficos, en contraste, Groq Cloud procesa cada consulta en latencias inferiores a dos segundos mediante hardware de inferencia especializado.

Migración del *framework* de presentación: para soportar concurrencia de múltiples usuarios, se recomienda migrar la capa de presentación de Streamlit hacia un *framework* orientado a servicios como FastAPI. En la arquitectura actual basada en Streamlit, cada sesión de usuario carga en memoria su propia copia de los fragmentos textuales del corpus y construye su propio índice BM25 de forma independiente, dado que estos residen en st.session_state que es privado por sesión, aunque los vectores y textos originales están almacenados de forma persistente en ChromaDB, el sistema requiere mantener en memoria RAM una copia de los textos para operar el índice BM25, que no puede trabajar directamente sobre disco, por ende, esto genera una replicación innecesaria de recursos cuando múltiples usuarios acceden simultáneamente. Bajo FastAPI, los fragmentos textuales y el índice BM25 se cargarían una única vez al iniciar el servicio como variables globales compartidas entre todos los usuarios concurrentes, eliminando la duplicación de memoria proporcional al número de sesiones activas.

Especificación de hardware para el escenario viable:

TABLA 28 RECURSOS DE HARDWARE RECOMENDADOS

Recurso	**Especificación recomendada**	**Justificación**
Procesador	4 a 8 núcleos	Atención concurrente de 50-100 usuarios
Memoria RAM	8 a 16 GB	Modelo de *embeddings*, ChromaDB y BM25 compartidos
Almacenamiento	20 GB	Base de datos vectorial persistente y corpus documental
GPU	No requerida	El modelo de lenguaje se procesa en Groq Cloud
Conectividad	Conexión estable a internet	Comunicación con la API de Groq

Esta especificación pertenece a un servidor de gama media o incluso a una instancia de tamaño reducido, perfectamente alcanzable con la infraestructura institucional existente sin inversión adicional en hardware, habiendo que, la viabilidad del proyecto queda así garantizada bajo las restricciones planteadas.

4.5.4 Escenario óptimo hipotético

A modo de proyección y suposición técnica, se plantea cómo se configuraría cada componente del sistema en un escenario óptimo sin restricciones presupuestarias ni de infraestructura. Este escenario no es una recomendación para el contexto actual de la PUCE, sino una referencia que muestra el potencial máximo de la arquitectura propuesta.

TABLA 29 COMPARACIÓN ENTRE ESCENARIO VIABLE Y ESCENARIO ÓPTIMO

Componente	**Escenario viable (actual)**	**Escenario óptimo (hipotético)**	**Beneficio del óptimo**
Inferencia LLM	Groq Cloud (Llama 3.3 70B)	vLLM local sobre GPU NVIDIA A100 40 GB	Privacidad total, sin dependencia externa
Base de datos vectorial	ChromaDB en disco local	Qdrant distribuido	Escalabilidad a millones de vectores
Framework web	FastAPI	FastAPI con balanceador de carga	Soporte para miles de usuarios concurrentes
Persistencia de sesión	Memoria del servidor	Redis	Historial conversacional persistente
Recuperación	BM25 + vectorial	BM25 + vectorial + *reranker* BGE	Mayor precisión en el *top-K*
Autenticación	Contraseña por variable de entorno	Integración LDAP institucional	Trazabilidad individual de usuarios
Contenerización	No aplica	Docker + orquestación Kubernetes	Elimina dependencias del sistema operativo y permite escalar horizontalmente ante incrementos de demanda

En el escenario óptimo, la migración a inferencia local mediante vLLM sobre una GPU NVIDIA A100 eliminaría por completo la dependencia de servicios externos y garantizaría que ninguna consulta abandone la infraestructura universitaria, resolviendo cualquier preocupación de privacidad. No obstante, esta configuración requeriría una inversión inicial significativa en hardware especializado, así como personal técnico capacitado para el mantenimiento del modelo.

4.5.5 Análisis de privacidad

La transmisión de datos al servicio externo Groq Cloud requiere un análisis de privacidad diferenciado ya que es importante distinguir qué información se transmite efectivamente y cuál es su nivel de sensibilidad.

El prototipo transmite a Groq Cloud únicamente dos elementos por consulta: la pregunta formulada por el estudiante y los fragmentos de reglamento recuperados como contexto. No se transmiten los documentos completos, la base de datos vectorial ni ningún dato almacenado de forma persistente, dado que los reglamentos consultados son documentos institucionales de acceso público, la transmisión de sus fragmentos no compromete información confidencial, a pesar de ello, se tiene que el único elemento potencialmente sensible es la consulta del estudiante, en el caso de que esta contenga contexto personal.

Según la documentación oficial de Groq, las solicitudes de inferencia no se retienen por defecto, y los registros temporales de entradas y salidas solo se generan cuando es necesario solucionar errores que afectan la confiabilidad de la plataforma o investigar uso indebido (Groq Inc., 2025). Bajo este marco, y considerando que la institución no dispone de capacidad de cómputo con GPU para alojar un modelo local, se concluye que el despliegue en la nube mediante Groq Cloud presenta un riesgo de privacidad bajo y aceptable para el contexto del sistema, además como medida de mitigación complementaria, se recomienda incorporar un aviso visible que recomiende a los estudiantes no incluir información personal sensible en sus consultas.

4.5.6 Escalabilidad del corpus documental

El análisis de carga presentado hasta esta sección consideró la concurrencia de usuarios, pero es igualmente necesario evaluar el efecto del crecimiento del corpus documental. El prototipo fue validado sobre dos reglamentos que generan 107 fragmentos, no obstante, una implementación institucional completa podría incorporar más de cien documentos normativos, alcanzando un volumen estimado de varios miles de fragmentos.

La siguiente tabla resume el impacto de este crecimiento sobre cada componente del sistema.

TABLA 30 IMPACTO DEL CRECIMIENTO DEL CORPUS Y ACCIONES REQUERIDAS

Componente	**Impacto del crecimiento**	**Acción requerida**
Búsqueda vectorial (HNSW)	Mínimo (tiempo casi logarítmico)	Ninguna
Índice BM25	Alto (reconstrucción lenta)	Migrar a índice persistente
Indexación inicial con OCR	Alto (proceso prolongado)	Indexación por lotes
Memoria RAM	Moderado	Escalar a 16 GB
Base de datos vectorial	Moderado a alto	Evaluar migración a Qdrant

Como se observa, el componente de búsqueda vectorial no se ve afectado significativamente dado que el algoritmo HNSW mantiene tiempos de respuesta casi logarítmicos independientemente del tamaño del corpus, en cambio, el impacto principal está sobre dos aspectos. El primero es la indexación inicial, que ante un corpus extenso con documentos escaneados constituiría un proceso prolongado, recomendándose su ejecución por lotes de forma desatendida en lugar de la carga interactiva desde la interfaz web. El segundo es el almacenamiento en disco, que para una implementación institucional completa con más de cien documentos normativos requeriría al menos 50 gigabytes para acomodar el crecimiento de la base de datos vectorial, los documentos originales y las dependencias del sistema, frente a los 20 gigabytes suficientes para el corpus actual. Estas consideraciones se desarrollan como línea de trabajo futuro en la sección 5.2.10.

CAPÍTULO V: CONCLUSIONES Y RECOMENDACIONES

El presente capítulo cierra el ciclo de la metodología CRISP-ML(Q) con la formalización de los hallazgos del proyecto, asimismo, la sección 5.1 sintetiza las conclusiones del trabajo a partir de los objetivos planteados en el Capítulo I y los resultados experimentales obtenidos en el Capítulo IV. La sección 5.2 presenta las recomendaciones técnicas que orientan la evolución del prototipo hacia una versión institucional, identificando líneas de trabajo futuro fundamentadas en limitaciones detectadas durante el desarrollo.

5.1 Conclusiones

A partir del desarrollo y validación del prototipo, se concluye que la arquitectura RAG constituye una solución técnica viable para el problema institucional de consulta inteligente sobre normativa universitaria. Los hallazgos específicos derivados del proceso se muestran a continuación.

Se desarrolló exitosamente un prototipo funcional basado en arquitectura RAG capaz de procesar dos reglamentos institucionales de la PUCE en formato PDF, atendiendo consultas en lenguaje natural sobre su contenido con citación obligatoria de los artículos consultados. De igual forma, el prototipo opera en entorno controlado y cumple con los requisitos funcionales y no funcionales especificados en el Capítulo III.

El prototipo demostró capacidad para mantener la fidelidad documental como atributo principal de calidad, restringiendo sus respuestas al contenido del corpus indexado. Esta característica se logró mediante tres mecanismos coordinados: la restricción explícita en el *prompt* del modelo, el uso de temperature=0 para garantizar determinismo y la indicación obligatoria de citación en cada afirmación generada. Los resultados experimentales obtenidos validan estos mecanismos.

La combinación de búsqueda léxica BM25 y búsqueda vectorial densa basada en *embeddings* permitió cubrir tanto consultas que referencian términos institucionales específicos (FOPEDEUPO, Beca San Ignacio) como consultas semánticas que no comparten vocabulario exacto con el corpus. La incorporación de coincidencia exacta por número de artículo garantizó el acierto en la totalidad de las consultas directas evaluadas, categoría en la que el prototipo alcanzó un 100 % de acierto documental y un 100 % de citación correcta.

El prototipo superó una limitación arquitectónica inherente a los sistemas RAG tradicionales mediante la implementación de una ruta dedicada que atiende consultas estructurales (cantidad de artículos, listados, índices) directamente desde la meta data indexada, sin invocar al modelo generativo. Esta contribución técnica alcanzó un 100 % de acierto en las cinco meta preguntas evaluadas, respondiendo correctamente sobre la cantidad total de artículos y el listado completo de cada reglamento. Generando así una ayuda a la capacidad que no es atendida por los sistemas RAG conversacionales estándar, que dependen exclusivamente de la recuperación *top-K* y por tanto solo acceden a un subconjunto del corpus en cada consulta.

La validación experimental sobre 30 consultas distribuidas en cinco categorías confirmó el cumplimiento de todos los, criterios de aceptación definidos: tasa de acierto documental del 100 %, tasa de citación correcta del 93.3 %, superando el umbral definido de 90 %, declaración de ausencia del 100 % ante preguntas fuera de alcance, latencia media de 4.12 segundos y cobertura del 100 % en meta-preguntas, además se identificaron cinco casos límite durante el desarrollo iterativo, de los cuales cuatro recibieron corrección técnica y uno se documenta como trabajo futuro.

5.2 Recomendaciones

A partir de las limitaciones identificadas durante el desarrollo y la validación del prototipo, se formulan las siguientes recomendaciones técnicas para la evolución del sistema hacia una versión institucional de producción. Cada recomendación se fundamenta en una limitación específica documentada en los capítulos anteriores.

- ### Implementación de Reciprocal Rank Fusion

La fusión actual de candidatos provenientes de BM25 y búsqueda vectorial se realiza por suma directa de puntajes, los cuales operan en escalas distintas. Aunque esta decisión es adecuada para el corpus actual de 107 fragmentos, en corpus de mayor tamaño se recomienda implementar Reciprocal Rank Fusion, que es una técnica que pondera la posición relativa de cada fragmento en los rankings individuales en lugar de los puntajes absolutos, eliminando el sesgo de escala inherente a la fusión directa. Adicionalmente, la normalización por posición relativa que introduce RRF reduciría el sesgo observado en consultas de titulación, donde artículos más extensos obtienen puntajes BM25 más altos que artículos más cortos igualmente relevantes.

- Reranker cross-encoder para mejorar precisión

Se recomienda incorporar un *reranker cross-encoder* entre la recuperación híbrida y la generación de respuesta. Modelos como BGE-Reranker-v2-M3 reordenan los 18 candidatos iniciales evaluando explícitamente la relación entre la consulta y cada fragmento, mejorando significativamente la precisión del *top-K* final.

A diferencia de la búsqueda vectorial y BM25, que evalúan la consulta y cada fragmento de forma independiente, el *reranker cross-encoder* los procesa de manera conjunta, capturando relaciones de relevancia que la recuperación inicial, optimizada para velocidad sobre grandes volúmenes, no puede detectar. No reemplaza la recuperación híbrida, sino que refina el ordenamiento de sus resultados antes de la generación.

- Desambiguación de nivel académico en consultas de titulación

Durante las pruebas funcionales se identificó que consultas genéricas sobre titulación sin especificar nivel académico, como "¿cómo me titulo?" producen respuestas ancladas en los artículos de cuarto nivel que son maestrías con trayectoria de investigación, en lugar de los de tercer nivel, que corresponden a la población objetivo del prototipo. Este comportamiento es una limitación de ambigüedad, no de recuperación ya que el sistema recupera correctamente los artículos de ambos niveles, pero el modelo generativo selecciona el de mayor densidad semántica en el fragmento.

Para una versión de producción se recomiendan dos estrategias complementarias. La primera es incorporar en el *prompt* una instrucción que establezca el tercer nivel como nivel por defecto cuando el usuario no especifique, con indicación de mencionar el cuarto nivel como alternativa y la segunda, más robusta, es implementar una pregunta de desambiguación automática cuando el *router* clasifique una consulta de titulación sin nivel explícito haciendo que el sistema preguntara "¿tu consulta es sobre pregrado (tercer nivel) o posgrado (cuarto nivel)?" antes de ejecutar la recuperación, eliminando la ambigüedad en origen.

- Extracción de contenido tabular incrustado como imagen

Durante las pruebas se identificó que el Artículo 50 del Reglamento General de Estudiantes contiene su escala de equivalencias como imagen rasterizada, no como texto seleccionable, por lo que ningún mecanismo de extracción textual del prototipo puede recuperarla (caso límite 4, sección 4.3.5). Para una versión de producción se recomienda implementar OCR

selectivo por región: cuando la detección estructural de tablas (find_tables()) devuelva resultado vacío y la página contenga imágenes de dimensiones compatibles con una tabla, descartando logotipos por umbral de tamaño, aplicar reconocimiento óptico sobre dichas imágenes con reconstrucción de la estructura tabular mediante las coordenadas de palabra que provee image_to_data de Tesseract, además, puede tenerse en cuenta una biblioteca especializada en extracción de tablas desde imágenes como img2table, siendo que, esta mejora cerraría el único punto ciego de la estrategia de extracción híbrida.

- Migración a FastAPI para múltiples usuarios concurrentes

La arquitectura de sesiones de Streamlit, adecuada para el contexto de prototipo, genera una instancia separada de st.session_state para cada usuario, replicando el índice BM25 y los *chunks* por sesión, por lo que, se recomienda migrar a un *framework* orientado a servicios como FastAPI, donde los recursos compartidos como el índice BM25 y los fragmentos del corpus se cargan una única vez al arrancar el servidor y se comparten entre todos los usuarios concurrentes sin reconstrucción por sesión.

- Contenerización con Docker

Se recomienda contenerizar la aplicación mediante Docker, incorporando Tesseract OCR como dependencia del sistema operativo del contenedor mediante el gestor de paquetes del sistema base. Esta decisión elimina la ruta de instalación local hardcodeada en el código fuente, que actualmente asume una instalación en Windows, y garantiza que el entorno de ejecución sea idéntico en cualquier infraestructura donde se despliegue el contenedor

- Persistencia del historial conversacional mediante Redis

El historial de chat se almacena temporalmente en st.session_state y se pierde cuando la sesión del usuario expira o el navegador se cierra, por tanto, en un entorno de producción con múltiples usuarios, se recomienda persistir el historial en una base de datos en memoria como Redis, identificando a cada usuario mediante un *token* de sesión único. Esta migración permitiría a los estudiantes continuar conversaciones previas y al sistema mantener contexto entre sesiones.

- Autenticación institucional mediante LDAP

El control de acceso actual se basa en una contraseña única de administrador gestionada por variable de entorno. En producción institucional se recomienda integrar el sistema con el directorio LDAP de la PUCE o con un proveedor de identidad OAuth 2.0, permitiendo autenticación individual de estudiantes y administradores con sus credenciales institucionales habituales.

- Migración a inferencia local mediante vLLM

Dado que Groq Cloud transmite las consultas de los usuarios a servidores externos ubicados en los Estados Unidos, se recomienda que una versión de producción institucional evalúe el despliegue de un motor de inferencia local mediante vLLM u Ollama sobre servidores propios de la PUCE. Esta migración eliminaría la dependencia de servicios externos y garantizaría que ninguna consulta abandone la infraestructura universitaria.

- Detección automática de actualizaciones documentales

Actualmente, la actualización del corpus requiere intervención manual del administrador, por tanto, se recomienda implementar un proceso programado que verifique periódicamente la Gaceta Institucional de la PUCE en busca de nuevas versiones de los reglamentos, descargando y reindexando automáticamente los documentos cuando se detecten cambios. Este mecanismo garantizaría que el sistema siempre opere con la normativa más reciente.

- Evaluación automatizada con RAGAS

Las pruebas funcionales del prototipo se realizan mediante verificación manual contra el PDF original. Para una versión de producción se recomienda incorporar el marco RAGAS, la cual, automatiza la evaluación de sistemas RAG mediante métricas específicas como la fidelidad documental, relevancia de la respuesta y precisión del contexto recuperado. Esta automatización permitiría ejecutar regresiones de evaluación tras cada cambio en el sistema.

Su implementación implicaría dos etapas diferenciadas, siendo estas la construcción del conjunto de referencia mediante generación sintética asistida por el modelo de lenguaje sobre los documentos originales, y la evaluación propia mediante el LLM

juez interno de RAGAS. Ambas etapas tienen un costo en tokens proporcional al tamaño del corpus y al número de consultas evaluadas.

- Escalabilidad del corpus documental

El prototipo fue validado sobre un corpus reducido de dos reglamentos que generan 107 fragmentos. Sin embargo, una implementación institucional completa podría incorporar más de cien documentos normativos, alcanzando varios miles de fragmentos. Aunque el algoritmo HNSW de la base de datos vectorial mantiene tiempos de búsqueda casi logarítmicos incluso con corpus de gran tamaño, otros componentes requerirían ajustes ante este crecimiento. En primer lugar, la reconstrucción en memoria del índice BM25, que sobre el corpus actual toma menos de 50 milisegundos, dejaría de ser viable con miles de fragmentos, por lo que debería migrarse a un índice persistente cargado una única vez al iniciar el servicio. En segundo lugar, la indexación inicial de un corpus extenso con documentos escaneados que requieren reconocimiento óptico de caracteres constituiría un proceso prolongado, incompatible con la carga interactiva desde la interfaz web, por ello, se recomienda implementar un proceso de indexación por lotes ejecutado de forma desatendida en segundo plano. En tercer lugar, el incremento del volumen de datos elevaría los requerimientos de memoria del servidor de los 8 gigabytes estimados a un mínimo de 16 gigabytes. Finalmente, para volúmenes de esta magnitud sería recomendable evaluar la migración de la base de datos vectorial hacia una solución diseñada para alta escala como Qdrant, que ofrece mejor gestión de memoria y capacidades de distribución horizontal frente a colecciones de gran tamaño.

Anexos

Anexo A — Evidencia fotográfica de las pruebas funcionales

Link Drive: https://acesse.one/cjlnq7h

Anexo B — Código fuente del prototipo

El código fuente completo del prototipo, se encuentra disponible en el siguiente repositorio:

https://github.com/Alejandro04P/RAG-PUCE

REFERENCIAS BIBLIOGRÁFICAS

Krantz, T., Holdsworth, J., & Kosinski, M. (2026, marzo 20). ¿Qué es una base de datos vectorial? Ibm.com. https://www.ibm.com/es-es/think/topics/vector-database

de Zárate, J. M. O. (s/f). Sesgos algorítmicos y representación social en los modelos de lenguaje generativo (LLM). Gob.ar. Recuperado el 3 de abril de 2026, de https://cdi.mecon.gob.ar/bases/docelec/az6612.pdf

Barnard, J. (2025, noviembre 27). ¿Qué es embedding? Ibm.com. https://www.ibm.com/mx-es/think/topics/embedding

Studer, S., Bui, T., Drescher, C., Hanuschkin, A., Winkler, L., Peters, S., & Müller, K.-R. (2021). Towards CRISP-ML(Q): A machine learning process model with quality assurance methodology. En arXiv [cs.LG]. https://doi.org/10.20944/preprints202103.0135.v1

Díaz Sifuentes, S., Paz Fernández, C., & Torres Villanueva, M. (2025). Optimización de Modelos de Lenguaje Grande (LLMs) a través del Prompt Engineering. Revista Innovación y Software, 6(2), 6-11. https://doi.org/10.48168/innosoft.s24.a212

Nishisako, S., Higashi, T., & Wakao, F. (2025). Reducing hallucinations and trade-offs in responses in generative AI chatbots for cancer information: Development and evaluation study. JMIR Cancer, 11, e70176. https://cancer.jmir.org/2025/1/e70176

Universidad Virtual del Estado de Guanajuato. (2024). UVEG obtiene el 3er lugar en Innovación y Transparencia 2024 con su Chatbot IA. https://uveg.edu.mx/index.php/es/noticias1/705-uveg-obtiene-el-3er-lugar-en-innovacion-y-transparencia-2024-con-su-chatbot-ia

Topsakal, O., & Akinci, T. C. (2023, July). Creating large language model applications utilizing langchain: A primer on developing llm apps fast. In International conference on applied engineering and natural sciences (Vol. 1, No. 1, pp. 1050-1056).

Guerrero Sarmiento, A. F. (2025). Integración de RAG en asistentes inteligentes para mejorar los procesos de enseñanza y aprendizaje en la Universidad de los Andes [Trabajo de grado, Universidad de los Andes]. Repositorio Institucional Séneca. https://repositorio.uniandes.edu.co/server/api/core/bitstreams/b96eca78-a6eb-46ef-93b3-05c5d567409a/content

Institución Universitaria Salazar y Herrera. (2023). Estudiante desarrolla chatbot para interactuar con el RAE. https://www.iush.edu.co/es/Universidad/estudiante-desarrolla-chatbot-para-interactuar-con-el-rae-20231207/noticias

Santibáñez Camarillo, C. M., Millán Hernández, C. E., & Sánchez Soto, E. (2025). Evaluación de coherencia en respuestas generadas por un chatbot para consulta de legislación y reglamentos. Pädi Boletín Científico de Ciencias Básicas e Ingenierías del ICBI, 13(Especial 2), 39–47. https://repository.uaeh.edu.mx/revistas/index.php/icbi/article/view/14676

Smith, R. (2007). An overview of the Tesseract OCR engine. En Proceedings of the Ninth International Conference on Document Analysis and Recognition (ICDAR 2007) (Vol. 2, pp. 629–633). IEEE. https://doi.org/10.1109/ICDAR.2007.4376991

Robertson, S., & Zaragoza, H. (2009). The probabilistic relevance framework: BM25 and beyond. Foundations and Trends in Information Retrieval, 3(4), 333–389. https://doi.org/10.1561/1500000019

Groq Inc. (2025, agosto 1). Inside the LPU: Deconstructing Groq's speed [Entrada de blog]. https://groq.com/blog/inside-the-lpu-deconstructing-groq-speed

Reimers, N., & Gurevych, I. (2019). Sentence-BERT: Sentence embeddings using siamese BERT-networks. arXiv preprint arXiv:1908.10084. https://arxiv.org/abs/1908.10084

Rajesh, V., Jodhpurkar, O., Anbuselvan, P., Singh, M., Jallepali, A., Godbole, S., & Shrivastava, H. (2025). Production-grade local LLM inference on Apple Silicon: A comparative study of MLX, MLC-LLM, Ollama, llama.cpp, and PyTorch MPS. arXiv preprint arXiv:2511.05502. https://arxiv.org/abs/2511.05502

van Rossum, G., Warsaw, B., & Coghlan, A. (2001). PEP 8 — Style guide for Python code. Python Software Foundation. https://peps.python.org/pep-0008/

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). Design patterns: Elements of reusable object-oriented software. Addison-Wesley Professional. https://www.pearson.com/en-us/subject-catalog/p/design-patterns-elements-of-reusable-object-oriented-software/P200000009477

Malkov, Y. A., & Yashunin, D. A. (2018). Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. IEEE Transactions on Pattern Analysis and Machine Intelligence, 42(4), 824–836. https://doi.org/10.1109/TPAMI.2018.2889473

Bray, T. (2017). The JavaScript Object Notation (JSON) data interchange format (RFC 8259). Internet Engineering Task Force. https://www.rfc-editor.org/rfc/rfc8259

Wang, X., Wei, J., Schuurmans, D., Le, Q. V., Chi, E. H., Narang, S., Chowdhery, A., & Zhou, D. (2023). Self-consistency improves chain of thought reasoning in language models. En Proceedings of ICLR 2023. https://arxiv.org/abs/2203.11171

Groq Inc. (2025). Your data in GroqCloud [Documentación técnica]. GroqDocs. https://console.groq.com/docs/your-data