UNIVERSIDAD DE GUANAJUATO

CAMPUS IRAPUATO-SALAMANCA
DIVISIÓN DE INGENIERÍAS

Desarrollo de un asistente virtual por reentrenamiento de LLMs con recuperación-generación aumentada desde documentos normativos

TESIS

QUE PARA OBTENER EL TÍTULO DE:

MAESTRO EN INGENIERÍA ELÉCTRICA

PRESENTA:

Ing. Roberto García Guzmán

DIRECTORES:

Dra. Dora Luz Almanza Ojeda

Dr. Yair Alejandro Andrade Ambriz

SALAMANCA, GTO. Febrero, 2026

Agradecimientos Personales

A mis asesores, la Dra. Dora y el Dr. Yair, por guiarme en esta etapa y darme la oportunidad de desenvolverme con libertad.

A mis nuevos amigos, Jessica y Alejandro, que me acompañaron durante esta etapa y me permitieron compartir con ellos muchas experiencias.

A mi familia, por su apoyo incondicional.

A mi esposa, Isis, por su amor, paciencia y apoyo ilimitados, sin tí no hubiera sido posible.

Agradecimientos Institucionales

A la Universidad de Guanajuato, División de Ingenierías del Campus Irapuato-Salamanca (DICIS) por brindarme la formación académica y la infraestructura que me permitió llevar a cabo mis estudios.

Campus Irapuato-Salamanca | División de Ingenierías

A la Secretaría de Ciencia, Humanidades, Tecnología e Innovación, SECIHTI de México, por el apoyo económico otorgado a bajo el CVU **2006734**.

Ciencia y Tecnología
Secretaría de Ciencia, Humanidades, Tecnología e Innovación

Índice general

Índice de figuras

Índice de Tablas

Resumen

Los documentos normativos son un componente fundamental de las organizaciones, las cuales tienen la responsabilidad de difundirlos y lograr que sus miembros los comprendan. Para facilitar esta labor, es posible emplear tecnologías de cómputo modernas y técnicas de procesamiento de lenguaje natural con el fin de generar sistemas inteligentes que faciliten la consulta y comprensión de la normativa.

En este trabajo se presenta un asistente virtual, basado en LLMs de código abierto y técnicas de Recuperación-Generación Aumentada (RAG, Retreival-Augmented Generation), diseñado para responder preguntas sobre la normativa de la Universidad de Guanajuato utilizando como única fuente de información los documentos oficiales disponibles en formato PDF. El sistema alcanzó una puntuación BERT de 0.75 y un porcentaje de aciertos del 88 %. Una primera contribución de este trabajo es la implementación de un asistente capaz de ofrecer respuestas referenciadas, vinculando cada salida con los artículos y documentos correspondientes.

La segunda contribución consiste en el desarrollo de una metodología de extracción de información a partir de documentos PDF, la cual aprovecha sus elementos visuales para recuperar la estructura jerárquica de la normativa. Esto facilita la separación en fragmentos y permite relacionar correctamente cada respuesta con los artículos correspondientes de los documentos oficiales. Para la recuperación de información y la generación de respuesta, se evaluaron distintos modelos de extracción de *embeddings* y de inferen-

cia, seleccionando los más aptos considerando su desempeño y el hardware disponible: Qwen3-Embedding-8B-Q4_K_M como extractor de *embeddings* y gpt-oss-20b-MXFP4 para la inferencia.

Además, se generó un conjunto de preguntas y respuestas basadas en la normativa de la Universidad de Guanajuato con el fin de evaluar los modelos y reentrenar dos extractores de *embeddings* mediante el ajuste completo de sus parámetros. Uno de estos modelos alcanzó un recall de 0.88, lo que, si bien es inferior al 0.93 del modelo óptimo, muestra que es posible mejorar el desempeño de los modelos base mediante reentrenamiento con los datos obtenidos.

Finalmente, la tercera contribución es la implementación de una arquitectura de nube híbrida para poner el sistema en producción, aprovechando los recursos locales de la institución y permitiendo el acceso a través de una aplicación web por la cual los usuarios interactúan con los modelos de lenguaje.

Capítulo 1

Introducción

Todas las organizaciones requieren un conjunto de normas para funcionar adecuadamente, ya sea que éstas se den a conocer de forma verbal o que estén redactadas en documentos, deben cumplir el propósito de establecer funciones, aplicar reglas y, en general, regular el comportamiento de los individuos que conforman la organización.

Las organizaciones constituidas formalmente, y especialmente aquellas de gran tamaño, poseen una normativa redactada en documentos de texto que suelen ser extensos y contener un lenguaje que no es familiar para todos los miembros, lo cual puede dificultar el entendimiento completo de las normas. Por otro lado, en la última década, se han presentado avances significativos en lo que popularmente se conoce como *Inteligencia Artificial*, tratándose específicamente de una mejora notable en los modelos de lenguaje, que actualmente permiten a los sistemas computacionales comunicarse con los usuarios empleando lenguaje natural y desempeñar tareas relacionadas con la interpretación de texto.

Con el objetivo de aprovechar estas herramientas y favorecer a la comunidad universitaria, surge el propósito de generar una herramienta que ayude a conocer y entender la normativa de la Universidad de Guanajuato. Con

este fin, se emplearán técnicas modernas de procesamiento de texto para crear un asistente virtual, tipo chatbot, que tenga la capacidad de resolver dudas acerca de los documentos normativos de la universidad de forma fundamentada, y así, evitar la vulneración de los derechos de la comunidad y evitar omisiones en el cumplimiento de sus obligaciones.

1.1. Justificación

La normatividad vigente de la Universidad de Guanajuato (UG) se puede consultar en su página oficial [1], en la cual se presentan 22 documentos de texto en formato PDF que, en conjunto, pesan 6.9 MB. Si se convierten estos documentos a texto plano, se obtienen 1.1 MB de información, lo que equivale a aproximadamente 168,000 palabras. Además, la página web presenta otros documentos relevantes como el Plan de Desarrollo Institucional, la Gaceta Universitaria, acuerdos y actas de sus distintos órganos de gobierto, entre otros, los cuales abonan a la cantidad de textos que se espera que alumnos, docentes y administrativos conozcan y den seguimiento a sus actualizaciones.

Sin embargo, una dificultad que se presenta al momento de divulgar documentos de carácter oficial es la falta de cultura lectora en la sociedad mexicana. Según datos del MOLEC 2024 (Módulo de lectura del INEGI) [1], el porcentaje de población lectora disminuyó 14.6 puntos porcentuales en los últimos nueve años, pasando del 84.2 % en 2015 al 69.6 % en 2024. Aunque se observa una leve mejora de la cantidad de población lectora con respecto a 2023 (68.5 %), la falta de hábitos de lectura en la población es notable. De aquí se presenta la necesidad de proponer estrategias alternativas en la divulgación de documentos normativos, como puede ser el uso de asistentes inteligentes.

Por otra parte, el Foro Económico Mundial 2020 (WEF) presentó el concepto de Educación 4.0 [2], en la cual se reconoce que la educación debe

[1] https://www.ugto.mx/

adaptarse a las necesidades del futuro y para ello el UNESCO-UNEVOC (Centro Internacional para la Educación y Formación Técnica y Profesional) define la Educación 4.0 como una técnica de aprendizaje que se centra en transformar la educación con el uso de tecnologías avanzadas, reconociendo la Inteligencia Artificial como una de ellas. Alineado con esta visión, este proyecto pretende servir como ejemplo de la integración de la inteligencia artificial en el ámbito educativo, para resolver una de las necesidades de la comunidad. Así mismo, ayudará a fomentar el uso responsable de las tecnologías como los chatbots inteligentes al emplearlos como herramientas de apoyo en la interpretación de documentos normativos.

Esta tendencia del uso de la tecnología en el ámbito educativo se intensificó con la introducción de ChatGPT en 2022 [3], y los denominados modelos de lenguaje con capacidades conversacionales o chatbots inteligentes. Estas herramientas se han integrado en diversos ámbitos de la vida cotidiana, siendo la educación una de las más influenciadas, motivando a los investigadores a analizar el impacto que tienen estas herramientas en la educación, como es el caso de Peláez-Sánchez et al. [4]. Así mismo, se introduce la incógnita qué usos se les puede dar a estas herramientas en beneficio de la comunidad académica, siendo ésta la motivación del presente proyecto.

Si bien en el mercado actual existen chatbots inteligentes capaces de responder preguntas relacionadas con documentos que les sean proporcionados, resulta difícil para los usuarios proporcionarles todo el contexto necesario para que sus respuestas sean correctas, provengan de fuentes autorizadas y estén debidamente referenciadas. Además, estas herramientas comerciales presentan desventajas marcadas, como el hecho de que suelen estar sujetas a políticas de uso o privacidad que pueden ser desfavorables para los usuarios; pueden presentar variaciones en su costo, su nivel de personalización es limitado y, generalmente, se desconoce el uso que se da a los datos confidenciales que se comparten en ellas. Lo anterior obliga a considerar opciones de código abierto, con las cuales es posible alcanzar un mayor nivel de personalización, contar con un mejor control sobre las políticas de privacidad y garantizar la continuidad del sistema mediante un plan adecuado de mantenimiento y actualización. Es bajo esta filosofía que este proyecto pretende

crear un sistema adaptado a las necesidades de los usuarios y que funcione con herramientas de código abierto.

1.2. Antecedentes

El Procesamiento de Lenguaje Natural (NLP, Natural Language Processing) es una rama de la inteligencia artificial y la lingüística, dedicada a dotar a las computadoras de entendimiento de frases o palabras escritas en lenguajes humanos [5]. En este proyecto, intervienen diferentes áreas del NLP, como el modelado del lenguaje (LM, Language Modeling), la respuesta a preguntas (QA, Question Answering) y la similitud semántica de texto (STS, Semantic Textual Similarity), por mencionar algunas.

Respecto al modelado de lenguaje, es el área del NLP que busca predecir la siguiente palabra o carácter en una secuencia de texto dada. En esta área se han presentado avances significativos en las últimas décadas, comenzando por el desarrollo del modelo neuronal probabilístico de Bengio et al. [6], que empleaba una red neuronal y una tabla de búsqueda para predecir el siguiente carácter. Posteriormente, Collobert & Weston [7] propusieron el uso de redes neuronales entrenadas de forma semisupervisada para aprendizaje multitarea. Más adelante, Mikolov et al. [8] introdujeron los *embeddings* para aprender representaciones vectoriales distribuidas de las palabras. Luego surgieron los modelos *secuencia a secuencia* que utilizaban memoria a largo corto-plazo (LSTM) para convertir una secuencia de texto en otra, desarrollados por Sutskever et al. [9]. A su vez, Bahdanau et al. [10] presentaron la mejora de los modelos *codificador-decodificador* empleando mecanismos de atención para detectar partes de la oración relevantes. Finalmente, Vaswani et al. [11] introdujeron los *transformers*, una arquitectura de red neuronal basada únicamente en mecanismos de atención, que alcanzó excelentes resultados en tareas de traducción de texto automática.

La creación de los *transformers* dio origen a los Modelos de Lenguaje de Gran Tamaño (LLM, Large Language Models), los cuales emplean técnicas

de aprendizaje profundo, particularmente arquitecturas basadas en *transformers*, para aprender y entender patrones complejos y estructuras presentes en los datos. A su vez, la mejora de los LLMs llevó a la creación del Transformer Generativamente Pre-entrenado (GPT) por Radford et al. [12], una arquitectura basada en *transformers*, acompañada de un método de entrenamiento que consiste en pre-entrenar un modelo de lenguaje para predecir la siguiente palabra, empleando un corpus de texto de gran tamaño sin etiquetar, seguido de un ajuste fino en tareas específicas. Con esta técnica, fue posible alcanzar resultados superiores al estado del arte en diversas tareas de NLP, como son la inferencia de lenguaje natural, dar respuesta a preguntas, búsquedas por similitud semántica, entre otras.

Versiones posteriores del modelo GPT incluyeron conjuntos de entrenamiento más extensos, cambios en las técnicas de ajuste fino, incremento en el tamaño del modelo, entre otras mejoras, dando como resultado el lanzamiento de forma comercial de ChatGPT-3 [3]. Este modelo se definió como un modelo que interactúa de forma conversacional, con capacidad de responder preguntas de seguimiento, admitir sus errores, objetar premisas incorrectas y rechazar peticiones inapropiadas. Tras la liberación de ChatGPT-3 al público, múltiples empresas y organizaciones comenzaron a hacer uso de metodologías de entrenamiento similares para desarrollar sus propios modelos conversacionales y competir en el mercado.

Estos modelos conversacionales se consideran chatbots, los cuales se definen como sistemas computacionales inteligentes con capacidades de conversación, que son diseñados para emular una conversación humana con el objetivo de proporcionar orientación o apoyo [13].

En la actualidad, los modelos generativos basados en transformers como Chat GPT-4 (OpenAI), Gemini (Google), Llama (Meta), DeepSeek (DeepSeek), entre otros, son el estado del arte en cuanto a chatbots multidominio, ya que pueden responder preguntas y realizar tareas en diferentes dominios de conocimiento, siendo algunos de ellos también multimodales. Estos modelos cuentan con varias versiones y tamaños, para ajustarse a las necesidades de sus usuarios, algunos siendo de paga y otros de código abierto.

En el ámbito educativo los LLMs han tenido un impacto significativo, gracias a sus capacidades de generar resúmenes, resaltar partes importantes de textos, apoyar en tareas de escritura y en general proveer información a los estudiantes de temas específicos; además de servir a los profesores a generar material de apoyo personalizado, ayudar en la planeación de las lecciones o para calificar pruebas de forma semiautomática [14].

En el ámbito legal se ha explorado el uso de LLMs para apoyar a profesionales en ley, impuestos y finanzas, como es el caso de la plataforma Harvey. Esta empresa, a través de una alianza con OpenAI, entrenó un LLM con conocimiento legal e historial de casos reales, para generar una herramienta tipo chatbot que puede contestar preguntas teóricas, citar eventos reales y, en general, funcionar como asistente en la integración y revisión de casos complejos [15].

En cuanto a otras áreas del NLP, la presentación del *transformer* derivó en la creación de arquitecturas como BERT (Bidirectional Encoder Representations from Transformers), propuesta por Devlin et al. [16]. Esta arquitectura tiene la capacidad de generar representaciones bidireccionales profundas de texto (*embeddings*) con el objetivo de emplearlas posteriormente en un proceso de ajuste fino (*fine-tuning*) para realizar tareas específicas, alcanzando resultados superiores al estado del arte en actividades como la similitud semántica de texto. Posteriormente, se presentaron diversas modificaciones y alternativas a esta arquitectura, entre las cuales destacan el modelo SBERT (Sentence BERT) de Riemers & Gurevych [17], MiniLM de Wang et al. [18] y Qwen3 de Zhang et al. [19], todas con el objetivo de desarrollarse adecuadamente en tareas de similitud semántica de texto, respuesta a preguntas, entre otras.

Estos modelos de *embeddings*, así como los modelos generativos basados en *transformers*, hacen uso de diferentes técnicas de entrenamiento y ajuste para obtener los resultados esperados en tareas específicas, siendo las técnicas más comunes la ingeniería de *prompts* y el ajuste fino.

La ingeniería de *prompts* tiene por objetivo modificar la forma en que un

modelo emite su respuesta, empleando instrucciones que se le proporcionan en forma de texto. Un ejemplo de aplicación de esta técnica se puede ver en el trabajo de Patil et al. [20] que usó instrucciones para guiar a los modelos a generar explicaciones amigables para los pacientes sobre conceptos médicos, enfermedades y opciones de tratamiento, con el fin de reducir la brecha de conocimientos entre médicos y pacientes.

Por otra parte, el ajuste fino es un proceso en el que un modelo preentrenado, como un LLM, es reentrenado en un conjunto de datos específico para adaptarlo a tareas o dominios especializados. Dentro de esta metodología existen dos técnicas ampliamente usadas: Supervised fine-tuning y Reinforcement Learning From Human Feedback [21]. Ambas metodologías emplean técnicas para ajustar el comportamiento del modelo a casos específicos de uso y dotarlo de comportamientos enfocados en tareas específicas.

A pesar de todas estas cualidades, los modelos de lenguaje usualmente enfrentan problemas como las alucinaciones, la desactualización del conocimiento y la falta de transparencia y trazabilidad de su proceso de razonamiento. Con el objetivo de aliviar estos problemas se desarrollaron las técnicas de recuperación-generación aumentada de información o generación aumentada por recuperación (RAG, Retrieval-Augmented Generation). Las técnicas RAG consisten en mejorar los LLMs al extraer fragmentos de información relevante de bases de conocimiento externas, a través de cálculos de similitud semántica. Por ejemplo, en una investigación realizada en el ámbito de la medicina, se encontró que el uso de RAGs en conjunto con LLMs puede mejorar hasta un 39.7 % la exactitud en las respuestas de preguntas relacionadas a las normas de la American Academy of Orthopaedic Surgeons (AAOS), para la atención a lesiones del ligamento cruzado anterior [22].

1.3. Objetivos

1.3.1. Objetivo General

Desarrollar un asistente virtual tipo chatbot, alojado en la nube, que responda preguntas sobre la normativa de la Universidad de Guanajuato, implementando cuatro flujos de trabajo: extracción de información con RAGs, reentrenamiento de LLMs, backend y frontend. Los cuatro flujos de trabajo emplearán técnicas y herramientas de CI/CD y MLOps o DevOps (según sea el caso) para el despliegue continuo y actualización del sistema.

1.3.2. Objetivos específicos

- Implementar un algoritmo de extracción de información de documentos normativos de la Universidad de Guanajuato basado en técnicas RAG, que pueda interactuar con un LLM.
- Reentrenar un LLM que se comunique con el algoritmo de extracción de información, para que sea capaz de interpretar la normativa de la Universidad de Guanajuato y contestar preguntas relacionadas de forma especializada y personalizada, proporcionando referencias de dónde se obtiene la información.
- Desarrollar una API, empleando FastAPI o una tecnología similar, para comunicar una aplicación cliente con el modelo.
- Desarrollar una aplicación web tipo chatbot, empleando React o una tecnología similar, para que un usuario pueda hacer consultas al modelo a través de internet.
- Implementar un flujo de trabajo para el algoritmo de extracción de información que permita la actualización automática o semisupervisada de la nueva normativa.
- Implementar un flujo de trabajo, utilizando herramientas de MLOps, para el reentrenamiento, actualización y despliegue del LLM de forma automática o semisupervisada.
- Implementar un flujo de trabajo para la API y uno para la aplicación

web, que emplee técnicas y herramientas de CI/CD y DevOps para desplegar actualizaciones de forma automática o semisupervisada.

1.4. Hipótesis del trabajo

Es posible emplear recuperación-generación aumentada de información con información de un dominio específico y limitado, así como reentrenar un modelo de lenguaje para que pueda proporcionar información y resolver dudas que sean entendibles por personas sin conocimiento previo. La información que se emplee puede actualizarse con documentos recientes y el modelo puede reentrenarse para mejorar la calidad de las respuestas basándose en la interacción con los usuarios. Además, esta metodología puede aplicarse con diferentes fuentes de información, siempre que se ajuste el modelo al ámbito deseado.

1.5. Organización de la tesis

Este documento se divide en 5 capítulos. Esta introducción tiene el propósito de enfatizar los beneficios y cualidades de los modelos de lenguaje como herramientas multidominio y, específicamente, para responder preguntas e interpretar documentos. En el capítulo dos se presentan los conceptos esenciales que se requiere entender para realizar el proyecto, así como las herramientas de las que se hará uso. En el capítulo tres se presenta la metodología a seguir para alcanzar el objetivo de generar y poner en marcha la herramienta que sea capaz de responder preguntas con las características deseadas. Posteriormente, en el capítulo cuatro se muestran los resultados de implementación de la metodología así como el análisis de los mismos. Finalmente, en el último capítulo se presentan las conclusiones finales del proyecto.

Capítulo 2

Marco teórico

En este capítulo se presentan, de forma teórica, los conceptos y técnicas necesarias para el desarrollo de este proyecto. Se comienza por delimitar la problemática en el marco del NLP, posteriormente se presentan los modelos de lenguaje y su aplicación a la problemática, así como la descripción de la técnica de recuperación-generación aumentada que se empleará. Por último se presentan conceptos adicionales necesarios para el desarrollo del proyecto, así como trabajos relacionados en el área.

2.1. Delimitación de la problemática

El objetivo de este proyecto es desarrollar un asistente virtual tipo chatbot que responda preguntas de documentos específicos. Si bien, un asistente virtual inteligente se desempeña en diferentes áreas del NLP, este proyecto se centra específicamente en el que se conoce como dar respuesta a preguntas (QA, Question Answering).

En el contexto del NLP, se entiende por QA al desarrollo de sistemas que permiten a los usuarios utilizar interfaces de lenguaje natural para for-

mular preguntas y recibir respuestas concisas [23]. Estos sistemas pueden presentarse de diferentes formas, siendo las aplicaciones basadas en modelos de lenguaje de gran tamaño (LLMs) las que predominan en el área. El uso de LLMs hace posible la creación de sistemas que respondan a preguntas de forma interactiva (IQA, Interactive Question Answering), en los cuales el sistema puede entablar conversaciones con el usuario y responder de forma dinámica [24].

Una parte importante del QA es que, generalmente, funciona con ternas de información: pregunta, contexto, respuesta. La pregunta y la respuesta son secuencias de texto en lenguaje natural, y el contexto se refiere a la información que empleará el modelo para contestar la pregunta, usualmente el contexto también se encuentra en forma de texto, aunque puede tener otras formas.

Dentro de los sistemas IQA y QA, existen diferentes elementos a evaluar, sin embargo la evaluación de la respuesta es el parámetro fundamental en la mayoría de ellos; para ello se emplean conjuntos de datos especializados que contienen preguntas, su contexto y las respectivas respuestas, por ejemplo: TREC (**T**ext **RE**trieval **C**onference) [25], Yahoo! (YH) [26], WikiQA [27], Standford Question Answering Dataset (SQuAD) [28], entre otras.

2.2. Modelos de lenguaje de gran tamaño (LLMs)

Para entender lo que es un LLM, primero es necesario presentar el concepto de red neuronal artificial (ANN, Artificial Neural Network). Una red neuronal artificial es un modelo dividido en capas de procesamiento, donde cada capa se compone de nodos o neuronas. Cada neurona recibe entradas ponderadas, les aplica una transformación y devuelve una salida. En el modelo tradicional de red neuronal, conocido como *feedforward*, las salidas de cada capa sirven como entrada de la siguiente, de esta forma la información se propaga desde la capa de entrada, pasando por una o más capas ocultas hasta llegar a la capa de salida, donde se obtiene el resultado final de la red.

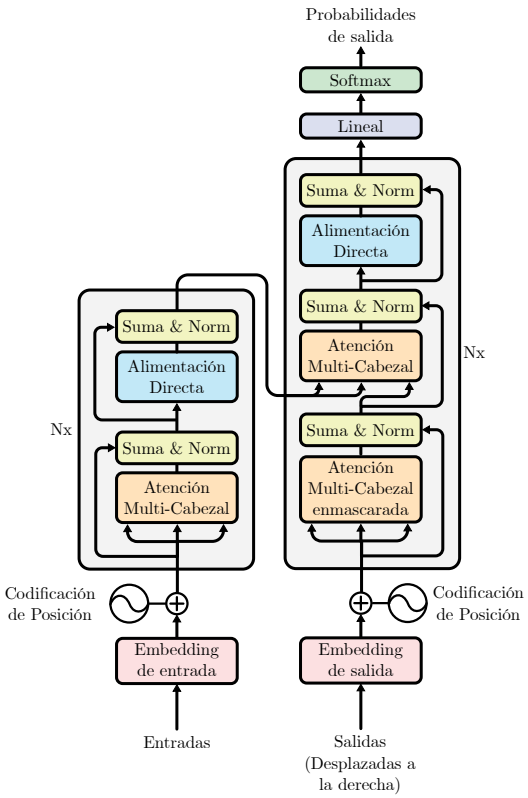


Figura 2.1: Arquitectura *transformer*.

Múltiples arquitecturas de ANNs han sido creadas con diferentes propósitos, y a su vez, se han creado mecanismos que funcionan en conjunto con estas redes para mejorar su desempeño en tareas específicas. En el contexto del NLP uno de estos mecanismos es la *atención*. Este mecanismo fue introducido por Bahdanau et al. [10] en el contexto de la traducción de texto automática con el propósito de que la red neuronal decida a qué partes de las oraciones prestar atención. Si bien Bahdanau no empleó el término *atención*, su propuesta fue tomada por Vashwani et al. [11] para crear la arquitectura del *transformer* para crear una arquitectura basada únicamente en mecanismos de *atención* la cual se muestra en la figura 2.1.

El *transformer* esta basado en una arquitectura del tipo codificador-decodificador, es decir, una red neuronal dividida en dos partes cuyo objetivo es convertir una secuencia de texto a otra. En su artículo, Vashwani et al. [11] definen una función de *atención* como un mapeo de una *query* y un conjunto de pares *key-value* a una salida, donde la *query*, las *keys* y los *values* son vectores. Esta salida de atención es una suma ponderada de los *values*, donde el peso asignado a cada *value* es calculado por una función de compatibilidad de la *query* con la *key* correspondiente. Para realizar este cálculo de forma eficiente se emplea el *Scaled Dot-Product Attention* definido en la Ecuación 2.1.

$$Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{2.1}$$

Esta operación puede replicarse de forma paralela, con el objetivo de que el modelo preste *atención* a información de diferentes representaciones de subespacios en diferentes posiciones, conformando lo que se denomina *Multi-Head Attention*, definido por la Ecuación 2.2.

$$\begin{aligned} MultiHead(Q, K, V) &= Concat(head_i, ..., head_h)W^O \\ \text{donde } head_i &= Attention(QW_i^Q, KW_i^K, VW_i^V) \end{aligned} \tag{2.2}$$

Estos bloques de atención son replicados y combinados en la estructura de *codificador-decodificador*, dando como resultado la arquitectura del *transformer*. Es de esta arquitectura, que derivan múltiples modelos con arquitecturas profundas cuyo propósito es realizar tareas relacionadas con el NLP, a estos modelos se les conoce como LLMs.

2.2.1. Modelos multidominio

Los modelos basados en *transformers* son entrenados de forma general en dos etapas: Un pre-entrenamiento no supervidado en un corpus muy

grande de texto para hacer modelado de lenguaje estándar, seguido de un ajuste fino supervisado en tareas específicas empleando conjuntos de datos especializados. Esta forma de entrenamiento fue propuesta por Radford et al. [12], y combinada con el uso de una arquitectura de *transformer decodificador* [29] es la base de los LLM actuales.

Los modelos se someten a ajuste fino con el objetivo de mejorar su desempeño en tareas de distintos dominios, moderar sus respuestas, y optimizar sus capacidades conversacionales. Ejemplos de ello son ChatGPT (OpenAI), Gemini (Google), Llama (Meta). Estos modelos, al combinarse con interfaces de usuario amigables y elementos de aplicaciones comerciales, son capaces de realizar una amplia variedad de tareas relacionadas con NLP, siendo una de ellas la respuesta a preguntas desde documentos.

Los modelos multidominio, al ser arquitecturas profundas, usualmente emplean el número de parámetros como métrica de su tamaño, estando por lo general en el orden de billones. Esta medida es importante pues sirve para determinar la cantidad aproximada de memoria que requieren para funcionar en su modo de inferencia de acuerdo a la Ecuación 2.3.

$$\begin{gathered} VRAM(bytes) \approx N_{params} \times \text{bytes-per-param} \times (1+\alpha) \\ \text{donde } \alpha \text{ es un factor de la carga adicional del framework} \end{gathered} \tag{2.3}$$

Un modelo de particular relevancia para este proyecto es el modelo Qwen3 [30], el cual es un modelo multi-idioma, de licencia abierta, disponible como un modelo denso de diferentes tamaños o como una arquitectura de mezcla de expertos (MoE, Mixture of Experts). Su arquitectura densa está basada en la arquitectura Llama (Meta), incluyendo bloques de atención de consultas agrupadas (GQA, Grouped Query Attention), bloques de alimentación directa (*feedforward*) con activación SwiGLU, codificación posicional rotatoria (RoPE, Rotary Positional Embeddings) y RMSNorm. Esta arquitectura optimiza el uso de memoria y el rendimiento general del modelo que se muestra en la figura 2.2 y está disponible en tres tamaños: 0.6B, 4B y 8B.

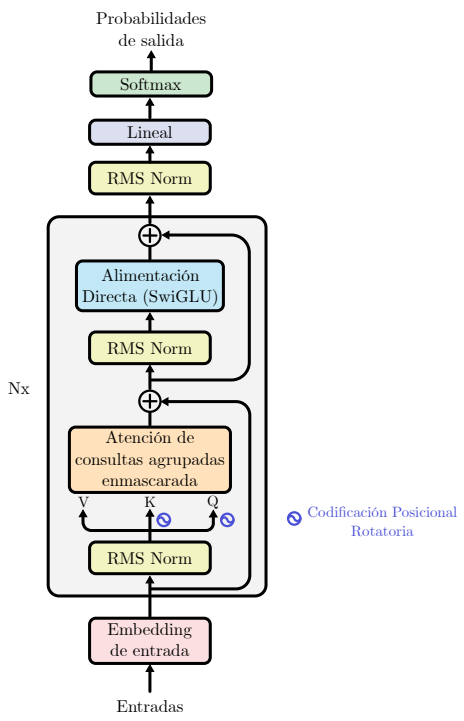


Figura 2.2: Arquitectura *Qwen3*.

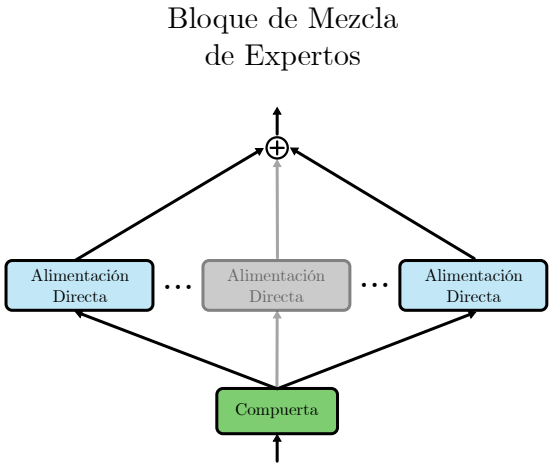


Figura 2.3: Arquitectura *Mezcla de Expertos*.

La arquitectura MoE fue inicialmente propuesta por Shazeer et al. [31] y empleada en *transformers* por Du et al. [32]. Esta arquitectura consiste en reemplazar las capas de alimentación directa (*feedforward*) del transformer por un bloque MoE. Este bloque se compone de múltipes capas de alimentación directa en paralelo, llamadas expertos, conectadas a un enrutador o compuerta que "escoge" los expertos más aptos para procesar el token de entrada, como se muestra en la figura 2.3. Esta arquitectura permite entrenar modelos con más parámetros pero a un menor costo computacional pues solo una parte de los parámetros se activa en cada paso.

Además del modelo Qwen3 en su versión MoE, otro modelo que se beneficia de esta arquitectura es el modelo GPT-OSS. Este modelo es un modelo de licencia abierta, basado en GPT-2 y GPT-3, liberado por OpenAI en dos versiones: 20B y 120B. Cuenta con bloques MoE de 32 y 128 expertos para cada versión, escogiendo los top-4 expertos para procesar cada token y emplea SwiGLU en las capas de alimentación directa. La arquitectura se presenta en la Figura 2.4.

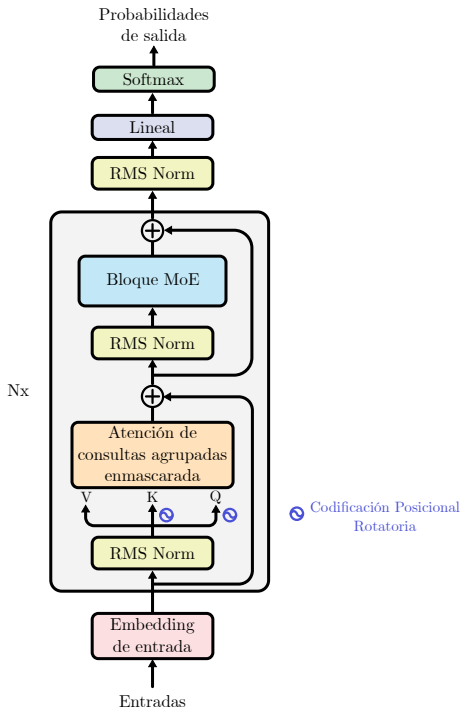


Figura 2.4: Arquitectura *GPT-OSS*.

2.2.2. Modelos de representaciones vectoriales *embeddings*

Se llama *embedding* a una representación numérica de una secuencia de caracteres, que puede ser una palabra o una oración completa. La característica principal de un *embedding* es que las palabras similares tienen representaciones similares, mientras que palabras u oraciones diferentes u opuestas tienen *embeddings* muy distintos. A los modelos con la capacidad de generar representaciones con estas características se les conocen como modelos de *embeddings*.

Existe una gran variedad de modelos para generación de *embeddings*, por ejemplo, el modelo SBERT tiene la capacidad de extraer *embeddings* de oraciones completas, considerando la posición y contexto de las palabras, permitiendo generar representaciones más significativas [17]. Otro modelo relevante es el modelo *Qwen3-Embedding* [19], el cual es una versión de Qwen3 que fue sometida a un proceso de ajuste fino en tareas como la similitud semántica o la respuesta a preguntas con conjuntos de datos generados sintéticamente y obtiene representaciones orientadas a la tarea que se desea realizar.

2.2.3. Cuantización de modelos

La cuantización consiste en reducir el uso de memoria de un LLM mediante el uso de pesos que utilicen representaciones de menor precisión, esto permite el despliegue de modelos grandes en hardware de menor tamaño [33]. Algunos de los métodos de cuantización relevantes para este proyecto son la cuantización a formato MPFX4 [34] que reduce el tamaño de los parámetros a 4.25 bits por parámetro, y las cuantizaciones Q4_K_M y Q8_0 que reducen el tamaño a 4 bits y 8 bits por parámetro respectivamente.

2.2.4. Modelos comerciales

Se entiende por modelos comerciales a aquellos que pertenecen a una entidad privada y su uso está restringido por el pago de una subscripción. Además, la arquitectura del modelo, métodos de entrenamiento, así como sus parámetros de funcionamiento no son públicos. Algunos ejemplos son ChatGPT (OpenAI), Gemini (Google), Claude (Anthropic), entre otros. Usualmente los proveedores de estos modelos lo hacen a través de aplicaciones web o APIs, y en ocasiones, cuentan con versiones gratuitas con capacidades limitadas.

2.2.5. Modelos de código abierto

Los modelos de código abierto son aquellos que se encuentran disponibles en sitios especializados, tanto su arquitectura como parámetros de funcionamiento son accesibles, usualmente a través de una licencia de uso que no es restrictiva. Algunos ejemplos son DeepSeek (DeepSeek), Llama (Meta), Qwen (Qwen), entre otros. Los creadores de estos modelos generalmente no proveen aplicaciones web o APIs, sino que permiten que los usuarios o desarrolladores los utilicen en sus propias infraestructuras, sin depender del creador.

2.3. QA con modelos multidominio

Un modelo multidominio tiene la capacidad de responder preguntas que se le hagan en lenguaje natural y emitir respuestas que usualmente son también en lenguaje natural, pero pueden ser de otra naturaleza.

2.3.1. Respuestas desde conocimiento previo

Durante el entrenamiento del modelo, éste se entrena con corpus de información muy grande y de diversas fuentes, además de que se le hace un ajuste fino para esta tarea, de tal forma que el modelo es capaz de responder una pregunta utilizando la información que se le proporcionó durante su entrenamiento.

Este tipo de respuestas es bueno cuando se trata de preguntas de dominio abierto donde la información es pública y no cambia con el tiempo, como es el caso de conocimientos generales o eventos históricos. Sin embargo, las respuestas suelen ser deficientes cuando se trata de eventos recientes, situaciones personales o conocimiento restringido, en estos casos se pueden presentar alucinaciones, que son respuestas cuya estructura es correcta, pero su información es errónea. Además, los modelos no cuentan con la capacidad de indicar la fuente de la información.

2.3.2. Respuestas desde un contexto

Es posible alimentar un modelo con información desde la cual pueda buscar o inferir la respuesta, a esta información se le denomina contexto y usualmente se trata de un texto del cual se puede extraer la respuesta. Un ejemplo es proporcionar la información biográfica de una persona como contexto y hacer preguntas sobre la persona. Generalmente, estas respuestas son más acertadas, pues los modelos son entrenados para responder preguntas de esta forma, siempre y cuando el contexto sea adecuado. Una desventaja de este método es que los modelos operan con ventanas de contexto limitadas, las cuales pueden ser de algunos miles de tokens hasta unos pocos millones y es posible que el contexto que se deba proporcionar sea mayor.

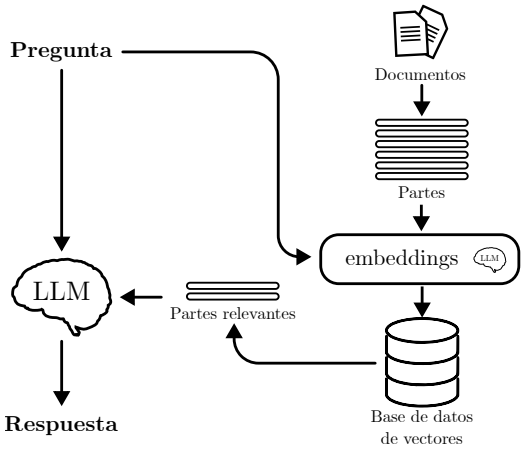


Figura 2.5: Diagrama de funcionamiento de un esquema de RAG simple.

2.4. Recuperación-Generación Aumentada (RAG)

Para superar las limitantes que presentan los LLMs, el uso de técnias de de Recuperación-Generación Aumentada (RAG, Retrieval-Augmented Generation) ha sido incorporado con frecuencia. Estas técnicas consisten en incorporar información o conocimiento desde fuentes de datos externas, que sirven como complemento a las preguntas [35]. Un esquema de RAG simple, como el que se presenta en la Figura 2.5, contempla tres etapas principales: indexado, recuperación y generación

2.4.1. Indexado

En su forma más simple, una base de conocimiento para RAG parte de la recolección de documentos relacionados con las preguntas que se harán a continuación. Una de las ventajas es que no hay limitante en la cantidad o tamaño de los documentos, aunque sí deben estar en formato de texto. Estos documentos se separan en fragmentos, comúnmente llamados *chunks*, el objetivo es obtener fragmentos pequeños pero que contengan suficiente información. Posteriormente se calcula la representación vectorial (*embedding*)

de cada fragmento, para finalmente almacenar cada uno con su respectivo *embedding* en una base de datos.

2.4.2. Recuperación

Cuando se le hace una solicitud al LLM, la recuperación consiste en buscar información relacionada, en forma de fragmentos de documentos, en la base de datos previamente construida. Para determinar cuáles son los fragmentos que más se relacionan con la pregunta, se calcula la similitud semántica entre la pregunta y cada fragmento. Una vez calculada la similitud con todos los fragmentos disponibles, se seleccionan los k fragmentos más similares a la pregunta o se establece un valor umbral y se consideran como relacionados a aquellos que estén por encima del umbral. Uno de los cálculos de distancia más comunes es la distancia coseno, en su forma de similitud coseno, como se muestra en la Ecuación 2.4.

$$d = 1.0 - \frac{\sum (A_i \times B_i)}{\sqrt{\sum (A_i^2)}\sqrt{\sum (B_i^2)}} \tag{2.4}$$

Otra métrica común es el cálculo de similitud empleando el producto interno, como se muestra en la Ecuación 2.5.

$$d = 1.0 - \sum (A_i \times B_i) \tag{2.5}$$

2.4.3. Generación

En esta etapa, la solicitud y los fragmentos marcados como relacionados son sintetizados en una instrucción coherente que se le proporciona al modelo. Usualmente esta instrucción incluye: indicación de responder la pregunta solamente con los fragmentos proporcionados, los fragmentos de texto y la pregunta.

2.5. Reentrenamiento de los modelos

Tanto los LLMs comerciales como los de código abierto pasan por una serie de pasos de entrenamiento y ajuste para funcionar adecuadamente en los diferentes dominios para los que son preparados, sin embargo, es posible mejorar su rendimiento en tareas más específicas aplicando diferentes técnicas de ajuste.

2.5.1. Ingeniería de instrucciones (*prompts*)

Cuando un modelo está entrenado para seguir instrucciones, es posible modificar su comportamiento dando instrucciones adicionales que le sirvan como guía al momento de dar la respuesta, a esto se le conoce como ingeniería de *prompts*. Con esta técnica se puede instruir al modelo a realizar una serie de pasos antes de emitir la respuesta, modificar el tono, longitud o intención de la respuesta, además de proporcionar información sobre la situación en que se debe desempeñar.

2.5.2. Ajuste fino supervisado

Consiste en tomar un conjunto de datos que contenga ejemplos del comportamiento que se desea que aprenda el modelo. En este proceso suelen usarse ejemplos con pares instrucción-respuesta, donde la respuesta tiene las características que se desea que el modelo aprenda. Existen diferentes ténicas para llevar a cabo este proceso, que van desde reajustar todos los pesos del modelo, hasta emplear técnicas como LoRA (Low-Rank Adaptation) que congela el modelo pre-entrenado y solo entrena un pequeño número de parámetros adicionales.

2.5.3. Aprendizaje por refuerzo

El aprendizaje por refuerzo consiste en entrenar al modelo para tomar decisiones que maximicen una recompensa. En el caso de los modelos de lenguaje, uno de los enfoques más utilizados es el aprendizaje por refuerzo con retroalimentación humana (RLHF, Reinforcement Learning from Human Feedback), el cual utiliza retroalimentación humana para optimizar modelos. Esta técnica de ajuste es particularmente útil porque es posible hacer que el modelo adquiera comportamientos que no son fáciles de modelar matemáticamente, sin embargo, tiene la desventaja de que requiere intervención humana y es por ello más lento.

2.6. Documentos normativos

Se entiende como documentos normativos a aquellos que contienen reglas o normas que rigen la operación de una organización o un organismo dentro de la misma, también pueden ser reglamentos de eventos o actividades específicas. En la actualidad, la mayoría de los documentos normativos se encuentran digitalizados en formato PDF y, en ocasiones, disponibles en servidores web para que los miembros de las organizaciones puedan consultarlos en cualquier momento.

En el caso de la normativa de la Universidad de Guanajuato, y de muchas otras normativas, la estructura de los documentos se divide principalmente en: Título, Sección, Capítulo y Artículos. Este trabajo pretende aprovechar esta estructura predefinida para obtener mejores fragmentos de documentos para la metodología RAG y para referenciar las respuestas del sistema.

2.6.1. Formato PDF

El formato PDF (Portable Document Format) es un formato creado con el objetivo de ofrecer una forma sencilla y segura de presentar e intercambiar documentos con independencia del software, hardware o sistema operativo que utilice quien los consulte. Con este fin, el formato PDF se encuentra estandarizado por la ISO (Organización Internacional de Normalización) bajo la ISO 32000-1:2008 (PDF 1.7) y más recientemente la ISO 32000-2:2020 (PDF 2.0). Sin embargo, este formato fue pensado como una herramienta para presentar documentos en una forma entendible por humanos, es decir, no está diseñado para que una máquina interprete su contenido de forma sencilla, lo cual dificulta el procesamiento automático del mismo.

2.6.2. Herramientas de extracción de texto

Los LLMs operan con entradas de texto, si los documentos a ser empleados se encuentran en formato PDF es necesario extraer la información textual que contienen. Existen diferentes herramientas para extraer el texto de documentos PDF, sin embargo, el resultado suele presentar errores o limitaciones propias del formato, como son la presencia de encabezados y pies de página entre el contenido del documento, la falta de distinción entre títulos, subtítulos o divisiones de secciones y la dificultad para leer tablas.

En el Apéndice A se encuentra una lista de las herramientas más comunes y sus limitantes, la cual se emplea para seleccionar la herramienta más apta. Dependiendo de la herramienta a emplear se debe hacer un procesamiento del archivo PDF para eliminar los defectos e información que no sea relevante, este proceso se presenta como parte de la metodología del proyecto.

2.7. Trabajos relacionados

En un esfuerzo por implementar una arquitectura robusta de QA desde diferentes fuentes de información, Christmann & Weikum [36] presentaron el sistema QUASAR, el cual emplea una arquitectura basada en RAG para responder preguntas desde texto sin estructura, tablas y grafos de conocimiento. Este sistema procesa las preguntas en diferentes etapas, que denomina: entendimiento de la pregunta, recuperación de evidencia y reclasificación y filtrado. Con esta metología alcanzó resultados superiores a modelos grandes como GPT-4 y Llama 3 en los benchmarks CompMix [37] y TimeQuestions [38] con un modelo Llama 3.1-8B-instruct[1].

Al ser sistemas basados en RAG, este proyecto comparte elementos comunes con el sistema QUASAR, sin embargo, se diferencía en que pretende aprovechar la estructura semi-estandarizada de los documentos normativos para realizar una fragmentación más eficiente de los documentos. Además, la restricción del dominio de las preguntas a documentos específicos, permite construir una base de datos de conocimiento más simple, tanto en tamaño como en pasos de procesamiento, lo que da como resultado un sistema más rápido y compacto.

Uno de los retos principales del RAG es la recuperación adecuada de fragmentos relacionados con las preguntas. Trabajos como el de Shao et al. [39] se centran en proponer mejores técnicas en la identificación de fragmentos relacionados en grandes volúmenes de datos, aplicando una sinergia entre el recuperador y el generador con el fin de mejorar las respuestas de manera iterativa. Por su parte, Shi et al. [40] proponen el *framework* *REPLUG*, que consiste en tratar al modelo de lenguaje que responde la pregunta como una caja negra y reentrenar el modelo responsable de la recuperación de información para optimizar su desempeño. Dadas estas consideraciones, el sistema propuesto en este proyecto incluye una etapa de ajuste fino para el modelo que obtiene los *embeddings* de los fragmentos de los documentos normativos.

[1] https://huggingface.co/meta-llama/Llama-3.1-8B-Instruct

Por último, existen herramientas comerciales y de código abierto con la capacidad de ejecutar metodologías RAG para responder a preguntas de documentos, siendo el más usado ChatGPT[2], el cual permite generar chats personalizados y proporcionar una serie de documentos como contexto. Al habilitar la opción de "Recuperación de conocimiento" la herramienta aplica RAG y responde las preguntas correspondientes. Otra heramienta con características similares es LMStudio[3], la cual es de código abierto y permite el uso de diferentes modelos, así como su ejecución en entornos locales.

A pesar de las bondades de las herramientas existentes, éstas ponen sobre el usuario la responsabilidad de encontrar los documentos requeridos y proporcionarlos al sistema. Además, tienen un límite a la cantidad de documentos de consulta y, cuando el historial de conversación crece demasiado, se debe reiniciar el chat y repetir el proceso. Otras desventajas incluyen la imposibilidad de referenciar sus respuestas a los artículos específicos, así como la presencia de texto no deseado, como encabezados, pies de página o malas interpretaciones de la secuencia del texto. Son estos problemas los que se busca resolver en este proyecto, al proporcionar una herramienta que esté lista para usarse, sea de código abierto, proporcione respuestas referenciadas correctamente y permita un nivel de personalización completo para la institución.

[2]https://help.openai.com/en/articles/8868588-retrieval-augmented-generation-rag-and-semantic-search-for-gpts

[3]https://lmstudio.ai/docs/app/basics/rag

Capítulo 3

Metodología

En este capítulo se presentan las herramientas y pasos a seguir para la implementación del sistema. Se comienza dando un panorama general del sistema, para posteriormente describir la fuente de información normativa, así como los pasos de implementación del asistente, que se divide en cuatro componentes: Extactor de información, modelador de lenguaje, API de comunicación y aplicación web. Posteriormente, se presenta la metodología de evaluación de los modelos que componen el sistema, para finalmente explicar el método de reentrenamiento del modelo de *embeddings*.

3.1. Panorama general

El desarrollo del asistente requiere el uso de múltiples lenguajes de programación, *frameworks* y librerías. Se emplean las librerías *transformers* y *llama_cpp* de *Python* para ejecutar los modelos LLM, así como *FastAPI* para programar la API de comunicación. Además, se usa *Javascript* (*NextJS* y *React*) para desarrollar la aplicación web.

En la Figura 3.1 se muestran los cuatro módulos que componen el siste-

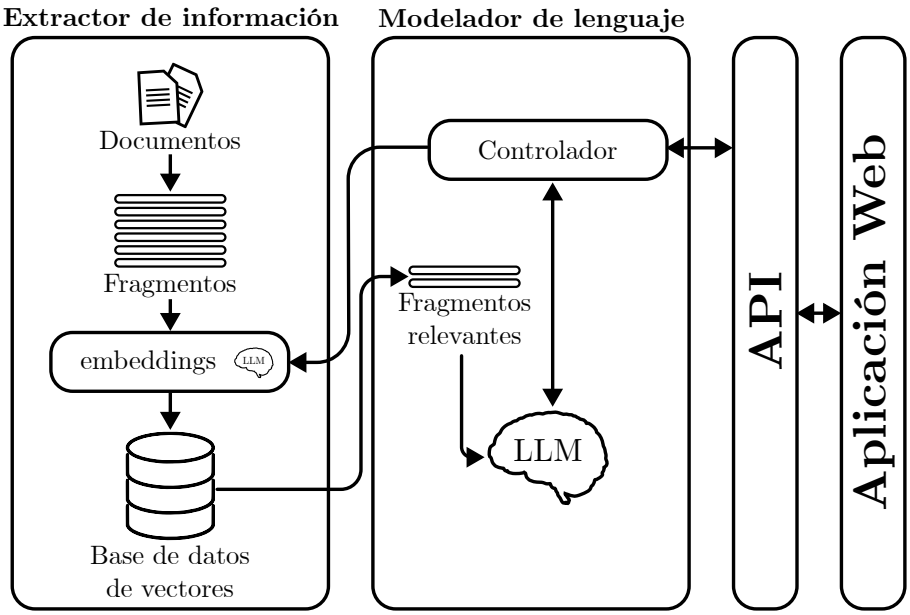


Figura 3.1: Diagrama de componentes que conforman el sistema.

ma, en donde el procesamiento de la información comienza previo a la interacción de un usuario, en el extractor de información. El extractor obtiene el contenido de los documentos y lo fragmenta, transforma los fragmentos en *embeddings* y, finalmente, almacena los *embeddings* en una base de datos. Una vez lista la base de datos, el usuario accede a la aplicación web, desde donde realiza una pregunta que se envía al modelador de lenguaje a través de la API de comunicación. Cuando el modelador de lenguaje recibe la pregunta, el controlador busca en la base de datos los fragmentos de información relacionados con la pregunta y se los proporciona al LLM como contexto, para que genere la respuesta. Una vez obtenida la respuesta, el controlador la envía de vuelta a través de la API para que el usuario pueda visualizarla en la aplicación web.

Para el desarrollo y puesta en producción de este proyecto se cuenta con una estación de trabajo *Dell Precision 7920 Tower* con un procesador *Intel®Xeon®Silver 4214R @2.40Ghz*, 256 GB de RAM DDR5, un disco SSD *Samsung PM881 SATA 52 GB*, un disco SSD *ADATA SU800 1TB*,

dos GPUs *Nvidia Quadro P620* de 2 GB de VRAM, una GPU *Nvidia RTX A4000* de 16 BG de VRAM y con sistema operativo Windows 11. Además, se obtuvo acceso durante cuatro meses al centro de supercómputo del CIMAT, a través de la convocatoria *Supercómputo como motor de colaboraciones académia-industria 2025*, para usar un nodo de cómputo GPU con dos tarjetas *Nvidia TITAN RTX* de 24 GB de VRAM cada una, con Ubuntu 22.04.5. Adicionalmente, se cuenta con una laptop de desarrollo *Dell G3 15* con una GPU *Nvidia GTX 1050Ti* de 4 GB de VRAM y con Ubuntu 24.04.02. Por último, para la puesta en producción del sistema se tiene una licencia de estudiante de Azure con $100 USD de crédito.

3.2. Fuente de información

El sistema está diseñado para responder preguntas de la normativa de la Universidad de Guanajuato, esta normativa se encuentra disponible en la página oficial de la universidad[1] en forma de 22 documentos PDF individuales. Cada documento tiene un número diferente de páginas, las cuales suman 511, representan 6.8 MB de información y, convertidas a texto, 168,011 palabras. Para alimentar la base de datos, los documentos son descargados manualmente y almacenados en un directorio, donde se utiliza el nombre del archivo como identificador. En la Tabla 3.1 se muestra un resumen de los documentos que conforman la normativa, así como el número de artículos que contiene cada uno.

3.3. Extractor de información

La labor del módulo de extracción de información comienza con un archivo en formato PDF y termina con la creación de una base de datos de *embeddings* de los fragmentos del documento. En la Figura 3.2 se observa la

[1] https://www.ugto.mx/gacetauniversitaria/normatividad/normatividad-vigente

N	Nombre del documento	Artículos	Transitorios
1	Código de Conducta e Integridad ...	13	2
2	Código de Ética de las Personas ...	7	1
3	Código de Ética de la Universidad ...	NA	1
4	Estatuto Orgánico de la ...	91	8
5	Ley Orgánica de la Universidad ...	75	8
6	Reglamento Académico de la ...	101	11
7	Reglamento de Becas, Apoyos y ...	39	5
8	Reglamento de Bienes del ...	28	6
9	Reglamento de Distinciones ...	21	4
10	Reglamento de la Defensoría de ...	42	8
11	Reglamento de la Junta Directiva ...	38	0
12	Reglamento del Personal Académico ...	98	7
13	Reglamento del Programa de ...	38	4
14	Reglamento de Mecanismos Alternos ...	37	5
15	Reglamento de Quienes Integran ...	31	9
16	Reglamento de Responsabilidades ...	53	7
17	Reglamento de Transparencia y ...	52	3
18	Reglamento Editorial de la ...	35	1
19	Reglamento Interno del Patronato ...	17	2
20	Reglamento para la Incorporación ...	49	5
21	Reglamento de Responsabilidades ...	30	11
22	Modelo Educativo de la Universidad ...	NA	NA

Tabla 3.1: Resumen de documentos normativos de la Universidad de Guanajuato

secuencia de procesamiento para un archivo y la salida de cada una de las etapas.

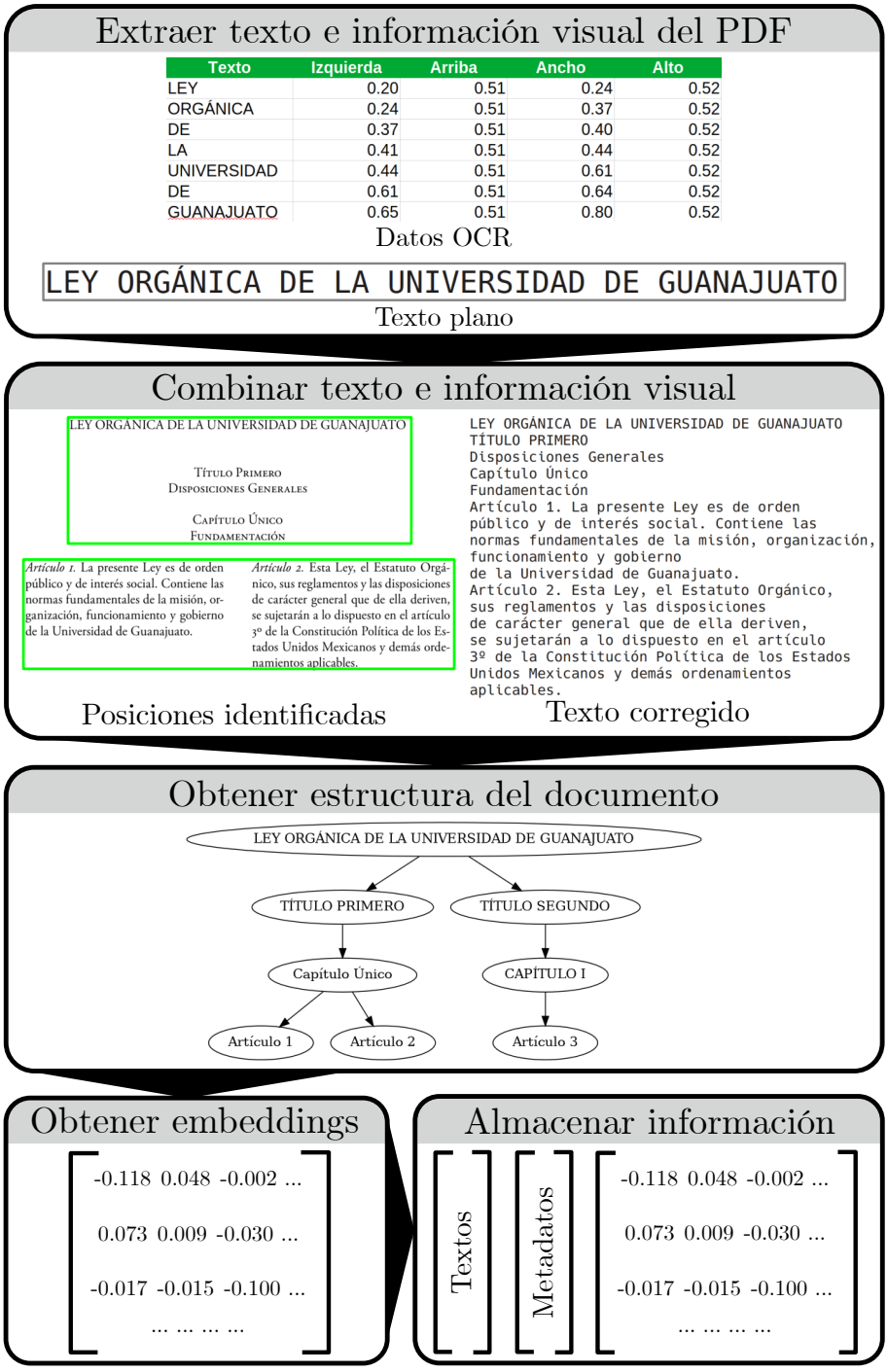


Figura 3.2: Diagrama de extracción de información de un archivo PDF.

3.3.1. Extraer texto e información visual del PDF

Cada documento PDF debe convertirse a texto para su procesamiento, conservando la información del formato y ubicación de los bloques de texto con el fin de poder extraer la estructura de títulos y secciones del documento. Por lo anteror, se extraen dos tipos de datos del archivo: texto plano e información visual relacionada con el formato.

Para seleccionar las herramientas de extracción de texto plano se hizo un análisis de aquellas que fueran de código abierto, considerando sus deficiencias en la extracción de texto (ver Apéndice A). Se seleccionó *PyPDF*, porque es la herramienta más usada para esta tarea con Python, y *PdfPlumber*, porque su extracción de texto es la más limpia y provee mecanismos para extraer la información visual.

El primer paso del procesamiento es extraer el texto plano con *PyPDF* o *PdfPlumber*. En el caso de PyPDF no se puede obtiener información visual que ayude a identificar los títulos, secciones y encabezados, además en ocasiones modifica el orden del texto. Para solucionar estos problemas se utiliza la herramienta *Tesseract*, la cual es un software de reconocimiento óptico de caracteres (OCR, Optical Character Recognition) que permite obtener la posición, ancho, alto y texto de cada palabra dentro del documento (Figura 3.3). Para emplear Tesseract, primero se convierte el documento a imágenes con una densidad de pixeles de 1000 DPI con *PDF2Image*, para posteriormente procesar cada imagen con la librería *PyTesseract*. Por otra parte, cuando se usa *PdfPlumber*, el texto va acompañado de la información visual necesaria, por lo que no es necesario emplear otras herramientas.

Tesseract es una herramienta, desarrollada por *Google*, que puede reconstruir el texto del documento, así como detectar las líneas, párrafos y secciones, sin embargo, con frecuencia comete errores en la detección del texto y el ordenamiento. Para corregir estos errores se propone combinar el texto plano obtenido con *PyPDF*, con la información de posición y tamaño devuelta por *Tesseract*, de esta forma se obtiene el texto correcto asociado a su ubicación y tamaño. Este proceso se describe en el diagrama de la Figura

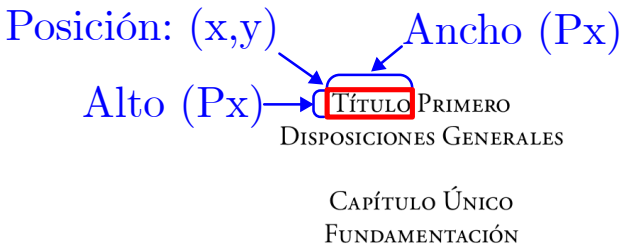


Figura 3.3: Ejemplo de información visual obtenida por *Tesseract* y *Pdfplumber*.

3.4.

Para el caso de PdfPlumber, la librería devuelve las mismas características que PyTesseract, sin embargo, comete también errores en la agrupación y el ordenamiento del texto, por ello se requiere realizar los pasos de "Crear grupos para reconstruir el texto" y "Reconstruir texto" del diagrama de la Figura 3.4.

La tarea de agrupar el texto en secciones para reconstruirlo en el orden correcto no es trivial, y es altamente dependiente del tipo de documento a procesar. Este proceso se explica en los diagramas de las Figuras 3.5 y 3.6, y funciona adecuadamente para documentos donde predomina el texto en una o dos columnas con títulos centrados, pero podría fallar cuando la estructura del documento difiera considerablemente.

Una vez realizada la agrupación de los elementos de la página, es posible reconstruir el texto respetando el orden normal de lectura, eliminando los errores de posicionamiento de *PyPDF* o *PdfPlumber* según sea el caso. Además, cuando se utiliza *PyPDF+Tesseract*, el texto contiene errores de detección de texto que serán corregidos al combinar la información visual del texto reconstruido, con el texto plano extraído con *PyPDF*.

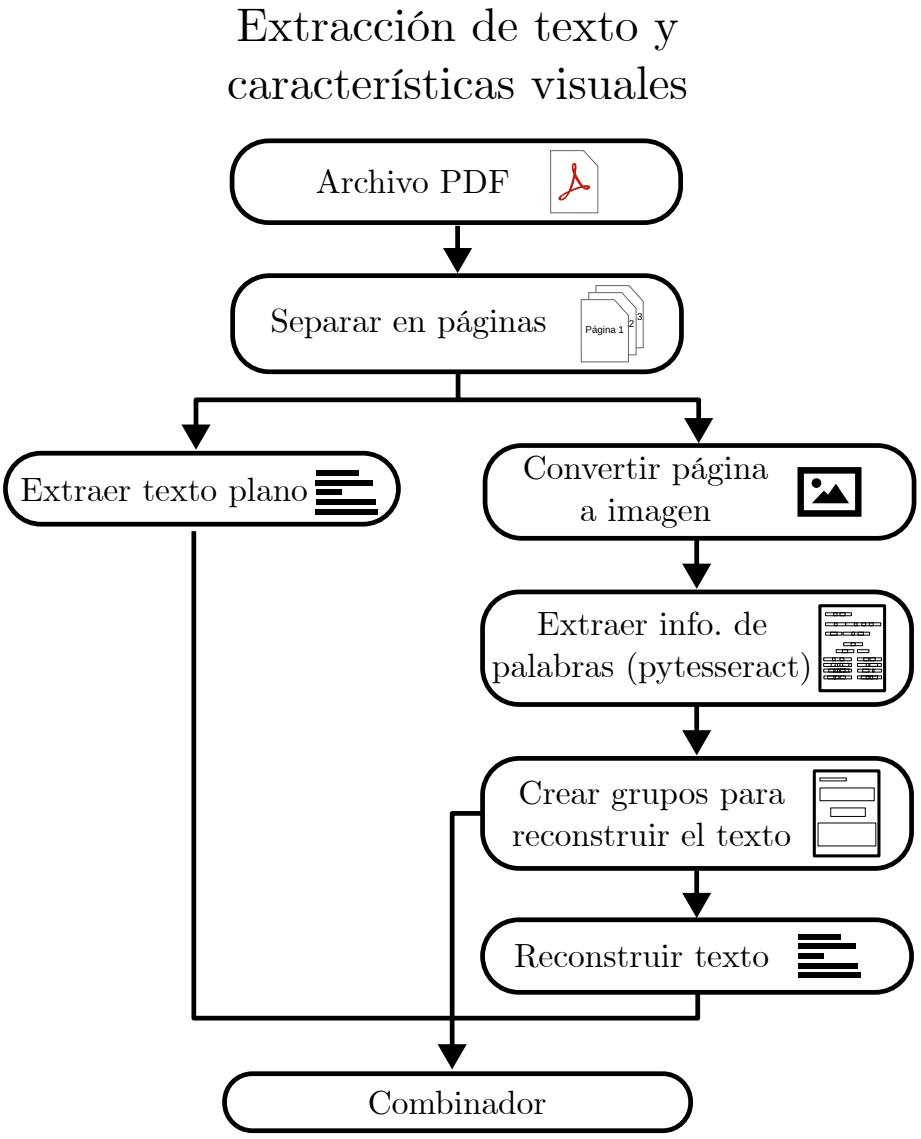


Figura 3.4: Detalle extracción de texto y características de OCR.

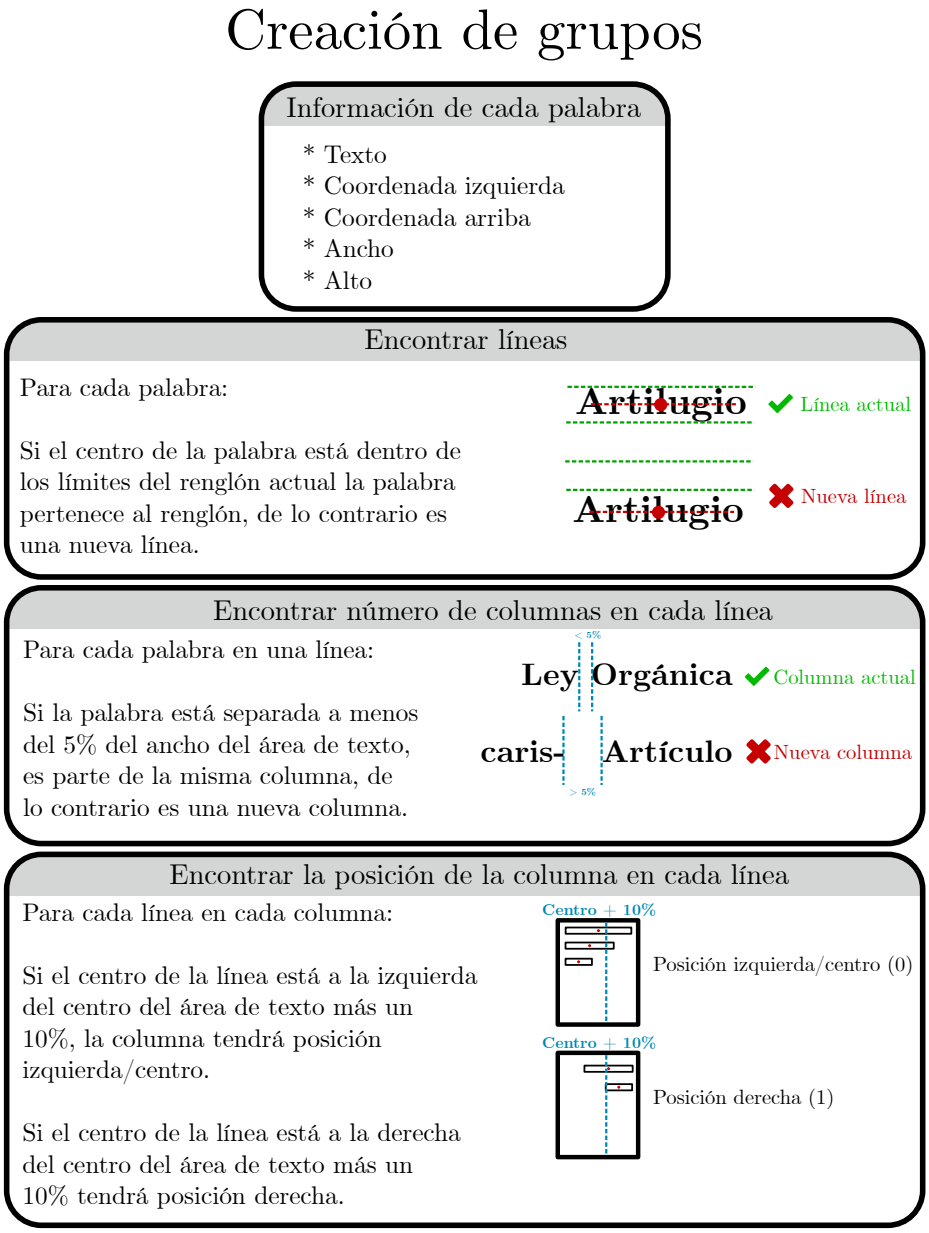


Figura 3.5: Detalle de reconstrucción de texto del documento (Pt. 1).

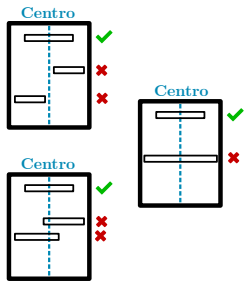
Ver si la línea tiene una columna centrada

Se considera centrado cuando:

El centro del área de texto se encuentra dentro de la columna.

La razón de la porción de texto de la columna a la izquierda del centro del área de texto y la porción de texto de la columna a la derecha del centro del área de texto es 1.0 más/menos 0.1.

La columna abarca máximo un 90% del área de texto.



Asignar un grupo

Para cada línea se asigna un nuevo grupo si:

El número de columnas de la línea y del grupo es diferente y la línea actual o la anterior tienen una columna centrada.

El número de columnas de la línea y del grupo es igual, pero la línea actual tiene una columna centrada y la anterior no (o viceversa), además la primera columna de la línea está en posición izquierda/centro.

Si la posición de las columnas de la línea coinciden con las del grupo, pero las columnas de la línea no se envuelven entre sí con las del grupo.

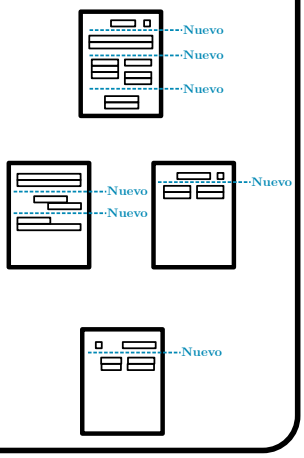


Figura 3.6: Detalle de reconstrucción de texto del documento (Pt 2).

3.3.2. Combinar texto e información visual

Cuando se usa *PdfPlumber*, este proceso no es necesario, pues la librería ya devuelve el texto correcto asociado a su información visual de posición y tamaño, sin embargo, cuando se usa *PyPDF+PyTesseract*, es necesario combinar estas dos fuentes de información para obtener el texto y la información visual correctas. El objetivo es tener la posición de cada palabra asociada con su texto correcto, de esta forma se podrá distinguir entre diferentes elementos, como títulos, párrafos, entre otros.

El proceso de combinación requiere las cadenas de texto extraídas por los dos métodos: la cadena proporcionada por *PyPDF* y la cadena reconstruida con la información visual, como se muestra en la Figura 3.6. Ambas cadenas se separan por palabra y se pasan a la librería *Difflib*, la cual aplica el algoritmo Ratcliff-Obershelp, también conocido como Coincidencia de patrones Gestalt, para comparar dos cadenas y encontrar sus diferencias.

Si se analizan los patrones de salida de la librería *Difflib*, es posible corregir las palabas detectadas con OCR utilizando las palabras extraídas directamente del texto, los detalles de esta implementación se explican en la Figura 3.7. En escencia, se considera la palabra obtenida con *PyPDF* como la correcta, se compara contra la palabra obtenida con *Tesseract* y si es diferente se sobreescribe. Al terminar el proceso se tiene un arreglo de datos donde se conoce la posición y dimensiones de cada palabra, así como su texto correcto, además se cuenta con datos adicionales de línea, columna, alineación o grupo que serán empleadas para dividir las secciones del documento más adelante.

Las ventajas de usar *PdfPlumber* sobre *PyPDF+PyTesseract* son que *PdfPlumber* es más rápido y la identificación del texto es certera, pues proviene directamente del archivo, sin embargo, no funciona si el PDF es un escaneo o si contiene texto en forma de imagen. Por su parte, al generar el método de *PyPDF+PyTesseract*, se desarrollaron elementos que fueron reutilizados al usar *PdfPlumber*, como la resconstrucción de texto. Además, si se confía plenamente en la predicción de *PyTesseract*, se puede dar soporte

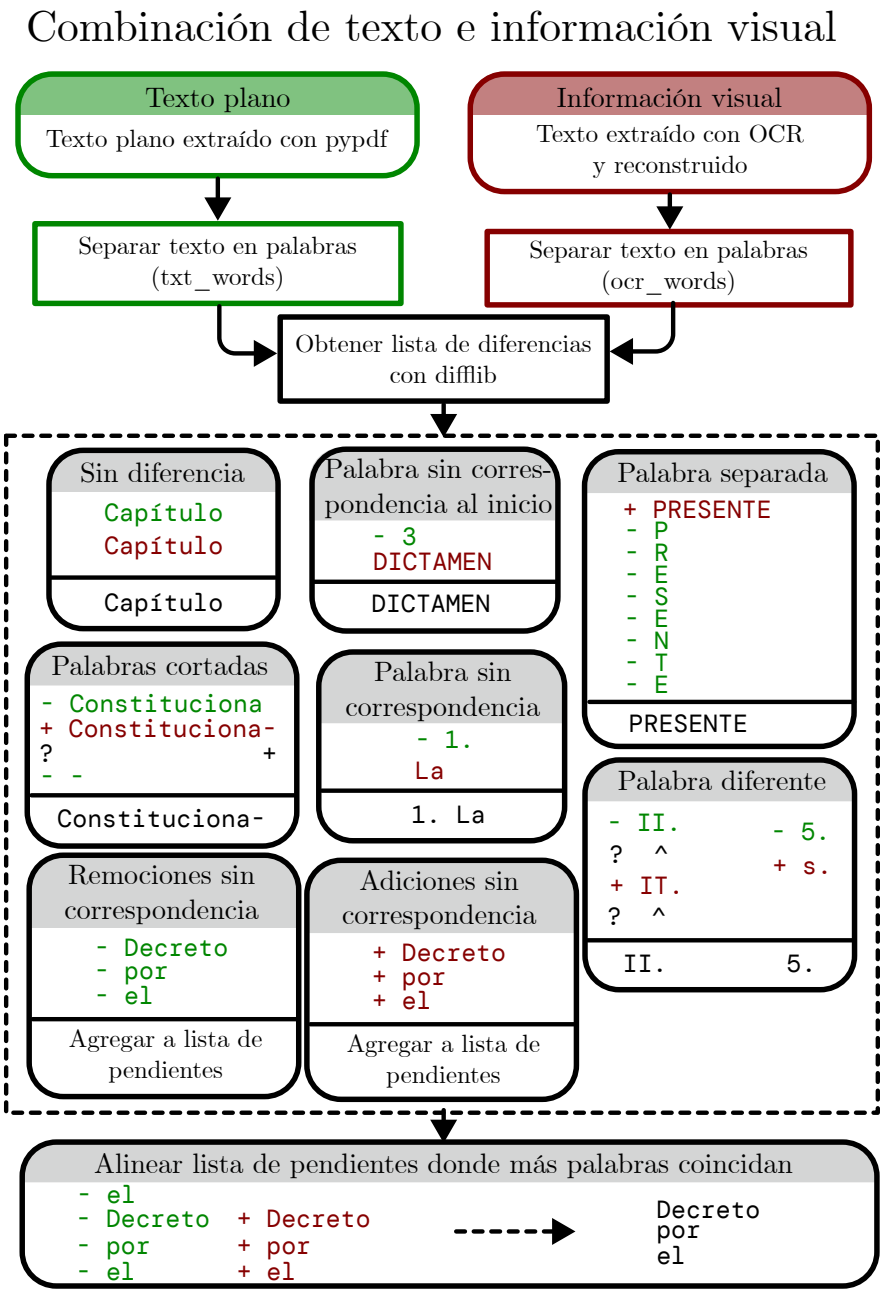


Figura 3.7: Detalle de combinación de texto plano con información de OCR.

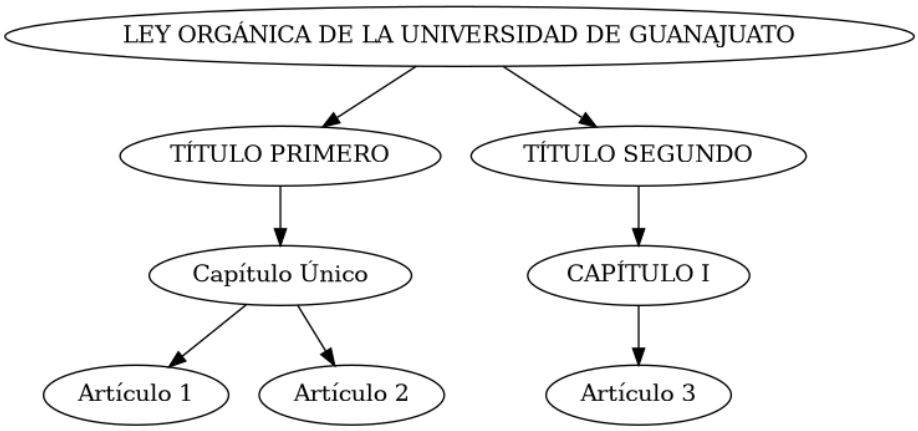


Figura 3.8: Porción del árbol del documento "Ley Orgánica de la Universidad de Guanajuato".

a documentos escaneados o con imágenes con texto.

3.3.3. Obtener estructura del documento

El objetivo de este paso es generar una estructua de datos en la que cada parte del documento esté referenciada a su sección y subsecciones correspondientes, por ejemplo, para la Ley Orgánica se desea conocer en qué título, capítulo y artículo se encuentra un texto específico. Para crear dicha estructura, se optó por generar un árbol, donde cada nodo corresponde a una sección del documento, además, las hojas y los nodos intermedios almacenan el texto de cada artículo o sección según sea el caso. En la Figura 3.8 se muestra una porción del árbol correspondiente a las primeras secciones de la Ley Orgánica.

Para generar el árbol es necesario analizar el documento en varios pasos. Primero, se divide el documento en bloques de texto que estén separados verticalmente, es decir, que haya al menos una línea en blanco entre ellos. Posteriormente se detectan los bloques que corresponden a títulos, estos típicamente se encuentran centrados en la página o en la columna. Para

encontrar las secciones o subsecciones se consideran los títulos centrados en la página y se les asigna un nivel. Para asignar el nivel se debe conocer de antemano la estructura del documento y generar expresiones regulares que ayuden a diferenciar un título, de un subtítulo, y así sucesivamente. Para los documentos normativos de la Universidad de Guanajuato la estructura es la siguiente:

- **Nivel 1 - Encabezado general:** Son textos abiertos que no tienen palabras o estructura específica y se consideran dentro del primer nivel.
  - Sin expresión regular
- **Nivel 2 - Título:** Los documentos se dividen en títulos. Cada título comienza con la palabra *Título*. También pueden ser numerales romanos.
  - ^(título|[xiv]+\.) .*
- **Nivel 3 - Sección (Opcional):** Algunos documentos dividen los títulos en secciones. Cada sección comienza con la palabra *Sección*. También pueden ser secciones numeradas.
  - ^([0-9]+[0-9]—sección) .*
- **Nivel 4 - Capítulo:** Los títulos o secciones se dividen en capítulos y cada capítulo comienza con la palabra *Capítulo*
  - ^capítulo .*

Además, se debe tener en cuenta que hay divisiones del documento que no se encuentran centrados, sino que están contenidos en el grueso del texto, como lo son los Artículos. Para identificar estas separaciones en el contenido, también se crean expresiones regulares que se verifican contra el inicio de cada bloque mientras se va construyendo el árbol.

1. **Artículo:** Los capítulos tienen uno o más artículos.
   - ^artículo ([0-9]+|[a-zé]+(ro|do|ro|to|mo|vo|no|único) (bis|ter|quáter|quinquies)?\.

Una vez identificados los títulos y teniendo la forma para encontrar las

divisiones dentro del contenido, se recorre el documento. Recorrer el documento es el equivalente a recorrer el árbol por profundidad, por lo que se va creando el árbol de la misma forma, es decir, cuando se encuentra un título se crea un nuevo nodo en el nivel correspondiente, siempre teniendo la referencia de cual será su padre, cuando se llega al nivel más bajo se guarda el contenido de texto en el nodo correspondiente. El proceso completo se presenta en la Figura 3.9.

3.3.4. Obtener *embeddings*

Para el cálculo de *embeddings* se emplean modelos LLM de extracción de *embeddings*. El sistema propuesto permite el uso y evaluación de varios modelos, los cuales serán: MiniLM (33M de parámetros) [18], MPNET [41] (110M de parámetros) y Qwen 3 [19] (600M, 4B y 8B de parámetros). Estos modelos se encuentran disponibles en HuggingFace y se pueden usar con las librearías *SentenceTransformers* o *Transformers*.

Los modelos MiniLM y MPNET fueron seleccionados por su tamaño reducido, ya que pueden ejecutarse en CPU de forma eficiente. Los modelos Qwen se seleccionaron ya que ocupan los primeros puestos en el MTEB Ladderboard de HuggingFace[2], que clasifica modelos de extracción de *embeddings* en diferentes idiomas y con múltiples métricas. Además, con el objetivo de reducir el uso de memoria, también se evalúan los modelos cuantizados de Qwen 3: f16 (16 bits), Q8_0 (8 bits), Q6_K (6 bits), Q5_K_M (5 bits) y Q4_K_M (4 bits). Estos modelos cuantizados se emplean con la librería *llama_cpp*, la cual es una implementación de la arquitectura *Llama* de Meta en C/C++ para hacer inferencia de forma eficiente. Un resumen de las características de los modelos de extracción de *embeddings* se encuentra en la Tabla 3.2.

El proceso para convertir la información del documento a *embeddings* consiste en recorrer el árbol en profundidad, y en cada nodo hacer lo si-

[2]https://huggingface.co/spaces/mteb/leaderboard

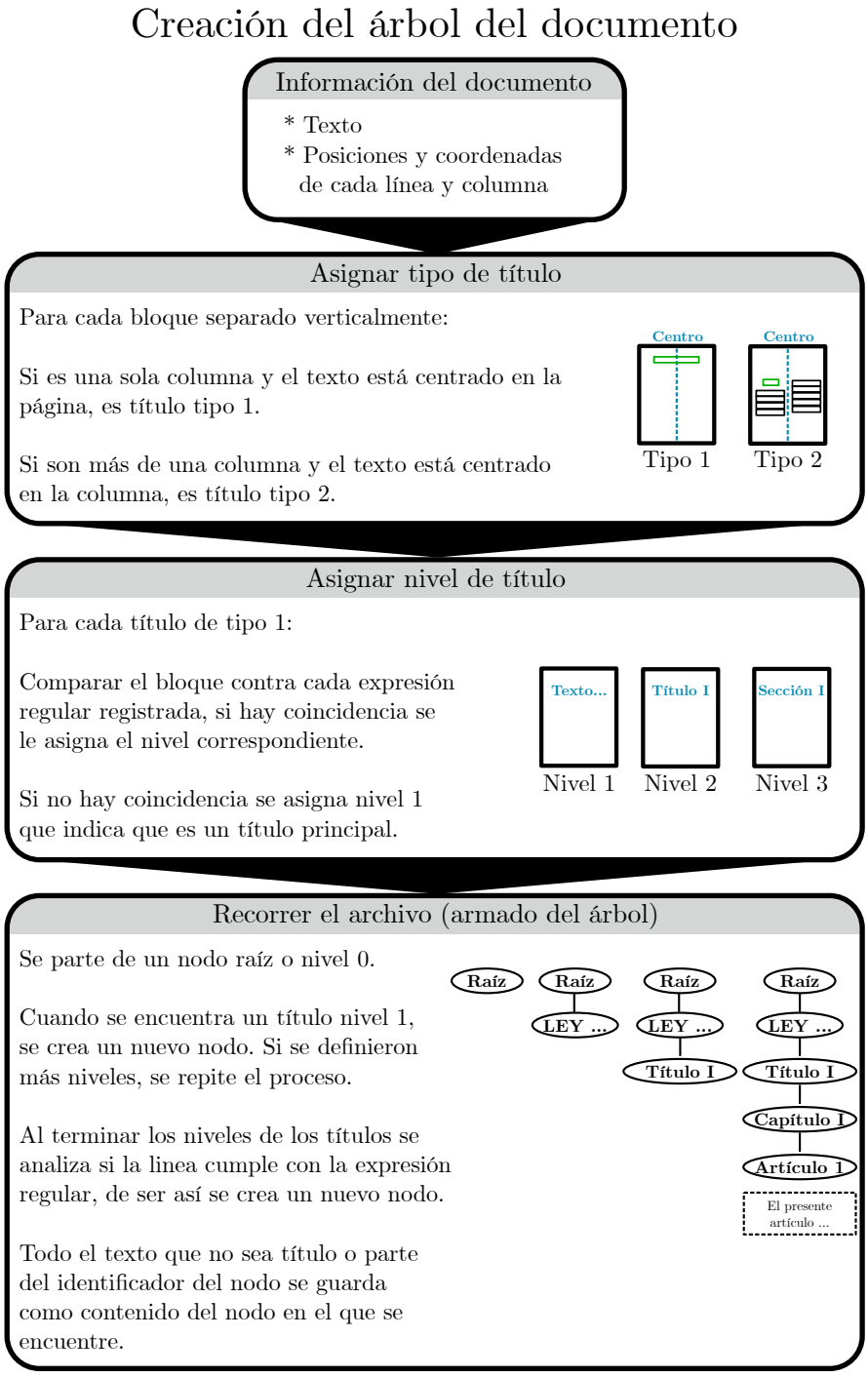


Figura 3.9: Proceso de creación del árbol de secciones de un documento.

Modelo	Tipo de dato	Memoria	Tamaño Embed.
Qwen3-Embedding-0.6B	BF16	3.65 GB	1024
Qwen3-Embedding-4B	BF16	15.31 GB	2560
Qwen3-Embedding-8B	BF16	28.5 GB	4096
Qwen3-Embedding-0.6B-GGUF	Q8_0	2.51 GB	1024
Qwen3-Embedding-0.6B-GGUF	F16	3.03 GB	1024
Qwen3-Embedding-4B-GGUF	Q4_K_M	5.01 GB	2560
Qwen3-Embedding-4B-GGUF	F16	10.18 GB	2560
Qwen3-Embedding-8B-GGUF	Q4_K_M	7.05 GB	4096
Qwen3-Embedding-8B-GGUF	F16	16.82 GB	4096
all-MiniLM-L6-v2	I64/F32	144 MB	384
multi-qa-mpnet-base-dot-v1	I64/F32	478 MB	768

Tabla 3.2: Resumen de modelos de extracción de *embeddings*

guiente:

1. Tomar el contenido textual del nodo, separarlo por párrafos y agrupar los párrafos en fragmentos de aproximadamente C caracteres. Cada fragmento será un registro independiente.
2. Utilizar *SentenceTransformers*/*Transformers*/*Llama_cpp* para calcular el *embedding* de cada fragmento y guardarlo como un vector de números.
3. Obtener la ruta del nodo dentro del árbol. Ej: Ley Orgánica → Título Primero → Capítulo segundo → Artículo 30.
4. Guardar la ruta y el nombre del nodo como metadatos, así como el nombre del documento y cualquier información relevante.

Al final, por cada fragmento se tendrá la siguiente información:

- Texto del fragmento.

- *Embedding* como vector de N valores numéricos.
- Metadatos:
  - Nombre del documento.
  - Ruta en el árbol.
  - Nombre del nodo.
  - Número de fragmento dentro del nodo.
  - Nombre del nodo padre

3.3.5. Almacenar información

Los *embeddings* pueden almacenarse de varias formas en el disco siendo, las más convenientes el formato CSV y las bases de datos con soporte para vectores. El formato CSV tiene la ventaja de ser portable y fácil de leer, basta con convertir los metadatos a formato JSON, para que puedan ser almacenados como texto, mientras que el vector de *embedding* se puede expandir y crear una columna para cada valor. En la Figura 3.10 se muestra un ejemplo de la información almacenada como archivo CSV. Este método funciona bien para hacer la evaluación y análisis de los modelos, sin embargo, no es apto para entornos productivos, ya que no se tienen optimizados para la búsqueda en los vectores de datos.

Otra forma de almacenar los *embeddings* es empleando una base de datos que soporte vectores. El mayor beneficio de estas herramientas es que permiten almacenar los datos de forma óptima, ya que la información se puede agrupar en tablas, se puede agregar información adicional y cuentan con funciones optimizadas de búsqueda por similitud de vectores. Algunos ejemplos de estas bases de datos vectoriales son: Chroma, Marco o PostgreSQL.

Este proyecto emplea ChromaDB, la cual funciona con SQLite y está diseñada para funcionar en entornos productivos, y permite el uso de otras librerías para calcular los *embeddings*, como *SentenceTransformers* y *llama_cpp* u otro método personalizado, además, permite el uso de diferentes parámetros de búsqueda por similitud semántica como son la distancia coseno y el producto interno, los cuales también se encuentran optimizados. En

sentences	metadatas	emb0	emb1	emb2	emb3	emb4	emb5	emb6	eml
	{'title': 'root', 'path': '"/root"'}	-0.119	0.048	-0.003	-0.011	0.052	0.010	0.115	0.
DICTAMEN DE LA COMISIÓN	{'title': 'LEY ORGÁNICA DE LA	0.073	0.009	-0.030	-0.051	-0.065	0.088	-0.011	0.
Analizada la iniciativa de refere	{'title': 'LEY ORGÁNICA DE LA	-0.017	-0.015	-0.100	-0.062	-0.002	0.008	0.010	0.
	{'title': 'DICTAMEN', 'path': '"/roo	-0.119	0.048	-0.003	-0.011	0.052	0.010	0.115	0.
Antecedentes 1. En el ejercicio	{'title': 'I. Del proceso legislativo	-0.024	0.057	-0.041	-0.023	-0.110	0.099	0.073	0.
2. En términos de lo dispuesto	{'title': 'I. Del proceso legislativo	-0.030	0.044	-0.009	-0.065	-0.115	0.038	-0.041	0.
3. En Sesión Ordinaria del 29 d	{'title': 'I. Del proceso legislativo	0.024	-0.005	-0.011	-0.078	-0.015	0.052	-0.011	0.
4. La Comisión de Gobernación	{'title': 'I. Del proceso legislativo	0.092	0.047	0.010	-0.061	-0.041	0.042	-0.027	0.
5. Como parte del proceso de e	{'title': 'I. Del proceso legislativo	-0.005	-0.006	-0.047	-0.057	-0.021	0.030	0.050	0.
	{'title': 'I. Del proceso legislativo	-0.119	0.048	-0.003	-0.011	0.052	0.010	0.115	0.
Quienes integramos esta Comi	{'title': 'II. Trabajo de la comisió	0.122	0.032	-0.083	-0.114	-0.045	-0.002	0.006	0.
Por lo anterior, desde el inicio	{'title': 'II. Trabajo de la comisió	0.018	0.062	-0.024	-0.045	-0.043	0.030	-0.021	0.
Al seno de la Comisión de Gob	{'title': 'II. Trabajo de la comisió	0.077	0.057	-0.061	-0.068	-0.081	-0.002	0.007	0.
Es así que, en una primera eta	{'title': 'II. Trabajo de la comisió	0.037	-0.014	-0.014	0.008	0.003	-0.016	-0.065	0.
1. Proyecto de Ley Orgánica El	{'title': 'II. Trabajo de la comisió	0.027	0.013	0.011	-0.031	0.044	0.008	0.053	0.
3. Dictámenes internos de la UI	{'title': 'II. Trabajo de la comisió	0.020	-0.062	-0.026	-0.081	0.057	-0.022	0.009	0.
4. Diversos Acuerdos del H. Co	{'title': 'II. Trabajo de la comisió	0.058	0.017	-0.005	-0.050	-0.002	0.058	0.044	0.
5. Plan de Desarrollo Institucion	{'title': 'II. Trabajo de la comisió	-0.037	0.058	-0.009	-0.035	-0.012	0.010	-0.049	0.
6. Normatividad vigente de la U	{'title': 'II. Trabajo de la comisió	0.092	0.039	0.019	-0.048	-0.041	0.010	0.029	0.
7. Documentos varios de la Aso	{'title': 'II. Trabajo de la comisió	0.015	-0.015	0.037	0.010	-0.078	-0.013	-0.065	0.
8. Documentos varios de la Org	{'title': 'II. Trabajo de la comisió	0.028	0.036	-0.024	0.016	-0.085	-0.008	-0.027	0.
10. Foros, encuentros y confere	{'title': 'II. Trabajo de la comisió	0.009	-0.063	-0.041	-0.057	0.048	-0.013	0.025	0.
11. Impacto de la reestructurac	{'title': 'II. Trabajo de la comisió	-0.012	0.023	-0.005	0.017	-0.025	-0.010	-0.061	-0.
Dicha documentación nos perm	{'title': 'II. Trabajo de la comisió	0.057	-0.008	-0.020	-0.022	-0.059	-0.038	-0.023	0.

Figura 3.10: Ejemplo almacenamiento de metadatos y *embeddings* en formato csv.

la Figura 3.11 se muestra un ejemplo de datos almacenados en ChromaDB.

3.4. Modelador de lenguaje

El modelador de lenguaje tiene dos funciones fundamentales: manejar la interacción con la base de datos de vectores y hacer uso del LLM de inferencia principal, en términos de RAG, se encarga de la recuperación y la generación. El proceso general que ejecuta el modelador de lenguaje se presenta en el diagrama de la Figura 3.12 y se explica en las siguientes subsecciones.

3.4.1. Recuperación de fragmentos relacionados

Dada una pregunta, el modelador la convierte a *embedding* empleando el mismo modelo usado en la base de datos de vectores. Este *embedding* se le proporciona a ChromaDB para que realice la búsqueda semántica, el parámetro de búsqueda puede ser por producto interno o por similitud

Table: embedding_metadata

	id	key	string_value	int_value	float_value	bool_value
	Filter	Filter	Filter	Filter	Filter	Filter
1	1	title	root	NULL	NULL	NULL
2	1	chroma:document		NULL	NULL	NULL
3	1	path	'/root'	NULL	NULL	NULL
4	2	chroma:document	DICTAMEN DE LA COMISIÓN DE ...	NULL	NULL	NULL
5	2	path	'/root/LEY ORGÁNICA DE LA UNIVERSIDA...	NULL	NULL	NULL
6	2	title	LEY ORGÁNICA DE LA UNIVERSIDAD DE ...	NULL	NULL	NULL
7	3	path	'/root/LEY ORGÁNICA DE LA UNIVERSIDA...	NULL	NULL	NULL
8	3	title	LEY ORGÁNICA DE LA UNIVERSIDAD DE ...	NULL	NULL	NULL
9	3	chroma:document	Analizada la iniciativa de referencia, esta ...	NULL	NULL	NULL
10	4	chroma:document		NULL	NULL	NULL
11	4	path	'/root/DICTAMEN'	NULL	NULL	NULL

Figura 3.11: Ejemplo de cómo se almacenan metadatos en Chroma DB. Los vectores se almacenan de forma óptima en un archivo separado.

coseno, dependiendo del modelo de *embedding*. Para hacer una búsqueda eficiente, ChromaDB emplea un índice llamado HNSW (Hierarchical Navigable Small World), esto le permite encontrar los fragmentos con mayor similitud a la pregunta sin tener que comparar con toda la base de datos.

Con ChromaDB se obtiene el top k de *embeddings* similares al *embedding* de la pregunta, por defecto k se establece en 5. De estos k *embeddings* se obtiene su texto original, su posición dentro del árbol del documento, el documento al que pertenece y los demás metadatos del fragmento.

3.4.2. Generación de respuesta

Para la generación de la respuesta, se emplea un modelo multidominio de código abierto. Para seleccionar el modelo óptimo para el sistema se seleccionaron distintos modelos cuya restricción principal es que pudieran ejecutarse en el hardware del sistema, es decir, que funcionen correctamente con 16 GB de VRAM. Se evaluarán los modelos Qwen 3 [19] (600M y 8B de parámetros) en sus versiones completas y cuantizadas, Llama 3.1 (8B

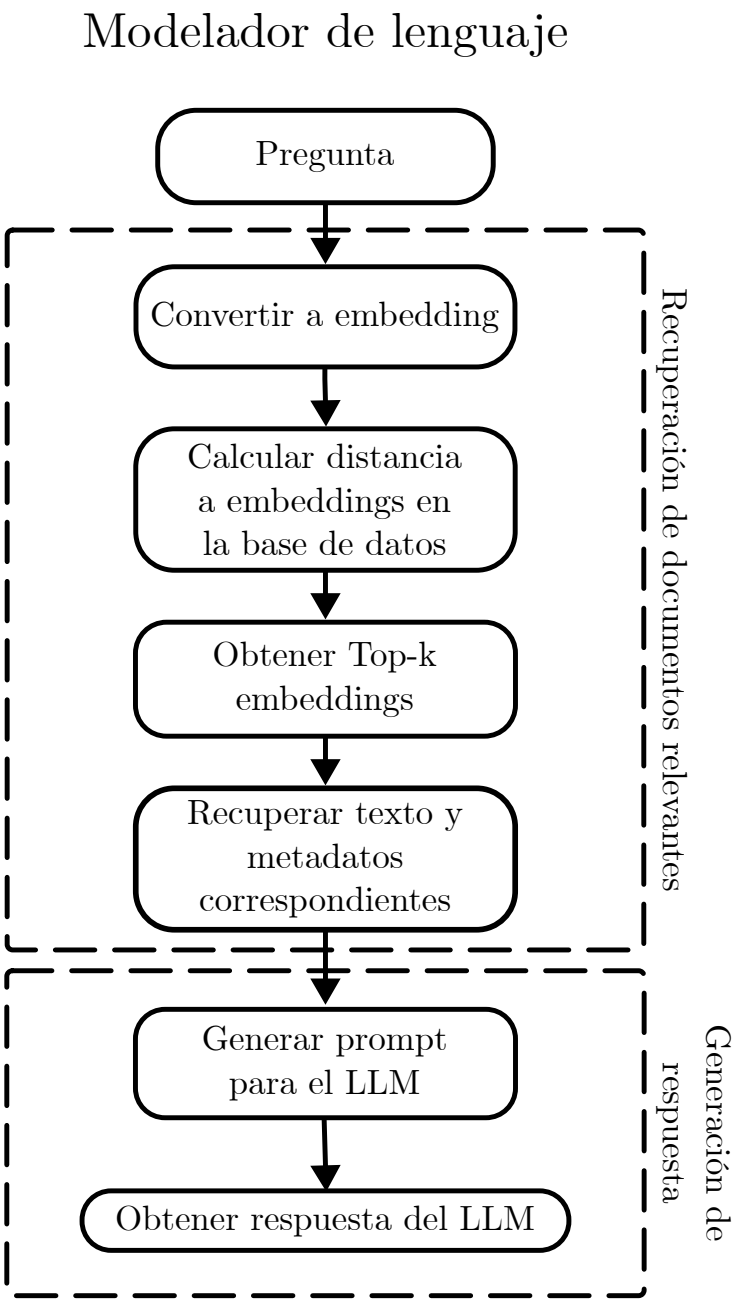


Figura 3.12: Pasos seguidos por el modelador de lenguaje para procesar una pregunta.

de parámetros) y el modelo GPT-OSS (20B de parámetros) en su versión cuantizada.

Modelo	Tipo de dato	Memoria	Herramienta
Qwen3-0.6B	BF16	1.74 GB	Transformers
Qwen3-8B	BF16	15.59 GB	Transformers
Qwen3-8B-Q4_K_M	Q4_K_M	7.07 GB	Llama_cpp
gpt-oss-20b-MXFP4	MXFP4	13.76 GB	Llama_cpp
Llama-3.1-8B-Instruct	BF16	15.27 GB	Transformers

Tabla 3.3: Resumen de modelos de generación de respuesta.

Para la evaluación inicial, a todos los modelos se les configura una instrucción de sistema simple que le indica que su propósito es ser un asistente, esto con el objetivo de evaluarlos en su forma básica.

```
Eres un asistente que ayuda a responder preguntas.
```

En una segunda evaluación, se comparan únicamente los mejores modelos de la primera etapa, pero esta vez con una instrucción de sistema más compleja, donde se incluyen instrucciones de cómo procesar las preguntas, el tono en que debe contestar, entre otros elementos. Esta instrucción es más cercana a la que se usará en la aplicación final y se puede consultar en el Apéndice B.

Teniendo la pregunta y los fragmentos relacionados con la misma, con sus metadatos, se transmite la información al LLM a través de una instrucción con la plantilla mostrada a continuación, que contiene la pregunta, y los fragmentos relacionados en formato JSON con toda su información.

```
<pregunta>
{pregunta}
```

```
</pregunta>

<contexto>
{documentos_en_JSON}
</contexto>
```

Un ejemplo de una instrucción completa se muestra a continuación:

```
<pregunta>
¿Qué contiene la Ley Orgánica de la Universidad de Guanajuato?
</pregunta>

<contexto>
[
    {
        "documento": "ley-organica-de-la-universidad
                     -de-guanajuato",
        "nombre": "Artículo 1",
        "contenido": "La presente Ley es de orden
                     público y de interés social. Contiene
                     las normas fundamentales de la misión,
                     organización, funcionamiento y
                     gobierno de la Universidad de
                     Guanajuato."
    },
    {...}
]
</contexto>
```

La instrucción generada se le proporciona al modelo junto con el historial anterior del chat para obtener la respuesta. Por lo general, la respuesta a la pregunta del usuario se encuentra únicamente en uno o dos fragmentos, pero al modelo se le proporcionan k fragmentos como contexto. Para lograr que en la aplicación web se le muestre al usuario únicamente los fragmentos de los que se extrajo la respuesta, se aprovecha la capacidad pre-entrenada del modelo de referenciar su respuesta, pero se mejora al agregar al comando del sistema una línea adicional para indicar que debe referenciar los fragmentos como una lista en formato JSON al final de la respuesta. Se teoriza que al proporcionar al modelo los fragmentos de forma estructurada en un JSON, se puede solicitar que haga referencia a campos específicos de los mismos también en formato JSON, el cual es un formato que la aplicación puede interpretar fácilmente. El comando adicional de sistema se muestra a continuación y sólo se agrega en el sistema final y no para las evaluaciones de los modelos.

```
Si la respuesta se encuentra en uno o varios artículos o
secciones específicas, cítalos al final de tu respuesta en
formato JSON de la siguiente manera:
<documents>[{"documento": [Nombre del Documento],
"nombre": [Artículo/Sección]}]</documents>.
Si el nombre del documento o artículo no está disponible en
el fragmento, omite la cita.
```

Un ejemplo de una respuesta del modelo con esta instrucción se muestra a continuación:

```
La Ley Orgánica de la Universidad de Guanajuato contiene
las normas fundamentales que establecen la misión, la
organización, el funcionamiento y el gobierno de la
Universidad, y se caracteriza por ser de orden público
```

```
y de interés social.


<documents>
[
    {
        "documento": "ley-organica-de-la-universidad
                      -de-guanajuato",
        "nombre": "Artículo 1",
    }
]
</documents>
```

3.5. API de comunicación

La tarea de la API es conectar la aplicación web con el modelador de lenguaje. La API funciona con peticiones HTTP de tipo POST y expone un solo endpoint que recibe parámetros en formato JSON, además esta protegida con un API-KEY que se debe enviar en el encabezado de cada petición. Un ejemplo de petición se muestra en la Figura 3.13. La estructura de una petición es la siguiente:

- **model:** Identificador del modelo a emplear.
- **num_related_questions:** (Opcional) Número de preguntas previas a tomar en cuenta para buscar fragmentos relacionados a la pregunta actual. Por defecto 1.
- **messages:** Lista de mensajes.
  - **role:** Rol del mensaje. “system”, “user” o “assistant”.
  - **content:** Texto de la pregunta, respuesta o instrucción.

La respuesta de la API también es en formato JSON y contiene el texto

```
{
    "model": "gpt-oss-20b-MXFP4",
    "num_related_questions": 1,
    "messages": [
        {
            "role": "user",
            "content": "La Universidad de Guanajuato ¿Es autónoma?"
        }
    ]
}
```

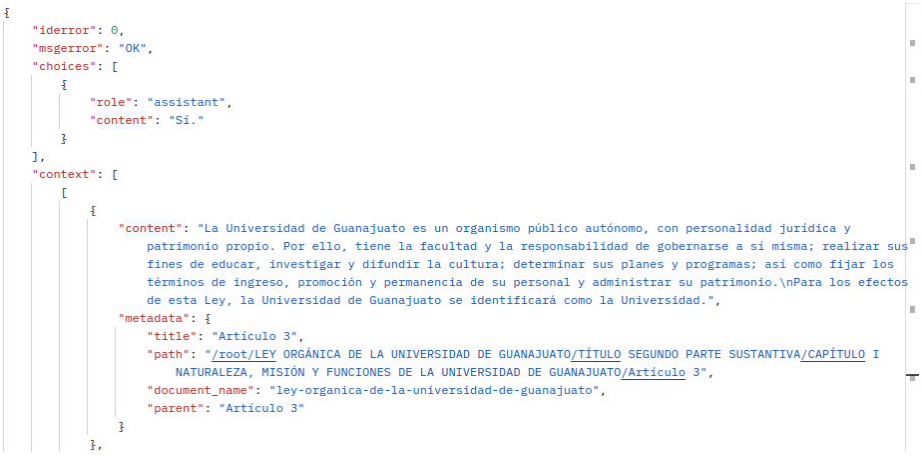
Figura 3.13: Ejemplo de petición a API.

de la respuesta, así como los fragmentos de los documentos que fueron tomados como contexto para emitirla, como se muestra en la Figura 3.14. En estos documentos, se incluyen los metadatos para poder desplegarlos en la aplicación web. La estructura de la respuesta es la siguiente:

- **iderror:** Identificador del error. 0 en caso de que no haya error.
- **msgerror:** Mensaje de error, en caso de haberlo.
- **choices:** Respuesta del modelo, como arreglo de respuestas, aunque solo sea una.
  - **content:** Contenido de texto de la respuesta.
  - **metadata:** Metadatos de los fragmentos de referencia.
    - **title:** Título de la sección.
    - **path:** Ruta del fragmento en la estructura del documento.
    - **document_name:** Nombre del documento de referencia.
    - **parent:** Nombre de la sección donde se encuentra el fragmento.

3.6. Aplicación web

La aplicación web es la única forma de interacción de los usuarios con el sistema, es por ello que debe contener todas las funcionalidades necesarias que en este caso son: ingresar una pregunta, recibir una respuesta y consultar



```
{
    "iderror": 0,
    "msgerror": "OK",
    "choices": [
        {
            "role": "assistant",
            "content": "Si."
        }
    ],
    "context": [
        [
            {
                "content": "La Universidad de Guanajuato es un organismo público autónomo, con personalidad jurídica y
                    patrimonio propio. Por ello, tiene la facultad y la responsabilidad de gobernarse a sí misma; realizar sus
                    fines de educar, investigar y difundir la cultura; determinar sus planes y programas; así como fijar los
                    términos de ingreso, promoción y permanencia de su personal y administrar su patrimonio.\nPara los efectos
                    de esta Ley, la Universidad de Guanajuato se identificará como la Universidad.",
                "metadata": {
                    "title": "Artículo 3",
                    "path": "/root/LEY ORGÁNICA DE LA UNIVERSIDAD DE GUANAJUATO/TÍTULO SEGUNDO PARTE SUSTANTIVA/CAPÍTULO I
                        NATURALEZA, MISIÓN Y FUNCIONES DE LA UNIVERSIDAD DE GUANAJUATO/Artículo 3",
                    "document_name": "ley-organica-de-la-universidad-de-guanajuato",
                    "parent": "Artículo 3"
                }
            },
```


Figura 3.14: Ejemplo de respuesta de API.

los documentos que dan fundamento a dicha respuesta. Con este fin, la aplicación web debe contar con una caja de texto para hacer la pregunta y una lista de mensajes enviados y recibidos como un chat convencional. Adicionalmente, se debe destinar un espacio para colocar los fragmentos de los documentos relacionados con la pregunta, como se muestra en el *wireframe* de la Figura 3.15.

Con el fin facilitar el uso de la aplicación, no será obligatorio crear una cuenta de usuario para hacer uso de la misma, sin embargo, se puede agregar la funcionalidad como opcional, para que los usuarios registrados puedan conservar el historial de conversaciones a largo plazo, así como acceder desde distintos dispositivos. En caso de no crear una cuenta, los chats solo estarán disponibles por 7 días, siempre y cuando el usuario no elimine las cookies de su navegador. Adicional a esto, otras funcionalidades se deben agregar como el visualizar el historial de chat o eliminar chats. Para cubrir todas las funcionalidades se diseña la estructura de base de datos de la Figura 3.16 para que sea implementada.

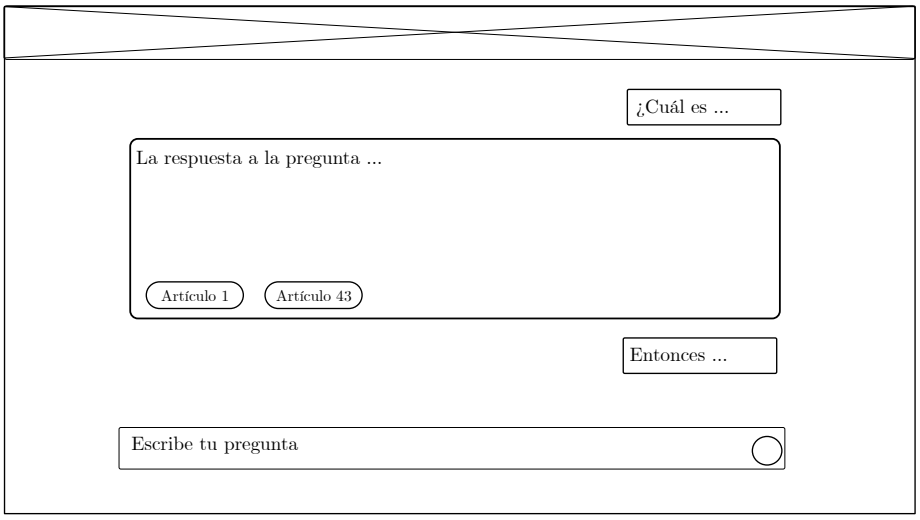


Figura 3.15: Esqueleto del chat de la aplicación.

3.7. Evaluación de los modelos

Para seleccionar los modelos de extracción de *embeddings* y de inferencia más aptos, se realiza una evaluación en varios pasos, donde de cada paso se seleccionan los mejores modelos, hasta llegar a la mejor combinación considerando rendimiento y recursos.

3.7.1. Conjunto de datos de evaluación

Es necesario generar un conjunto de datos de preguntas y respuestas de los documentos normativos, con el fin de conocer el rendimiento de los modelos en este dominio específico. Tres personas fungen como los generadores de preguntas, a cada una se le proporcionan 7 u 8 documentos, para completar el total de 21, se excluye el documento denominado “Modelo Educativo de la Universidad de Guanajuato y su Modelo Académico” pues no se encuentra divido en artículos y dificultaría la referenciación de las respuestas. Se instruye a cada participante para que genere de 1 a 4 preguntas por cada artículo de cada documento, ignorando los textos introductorios.

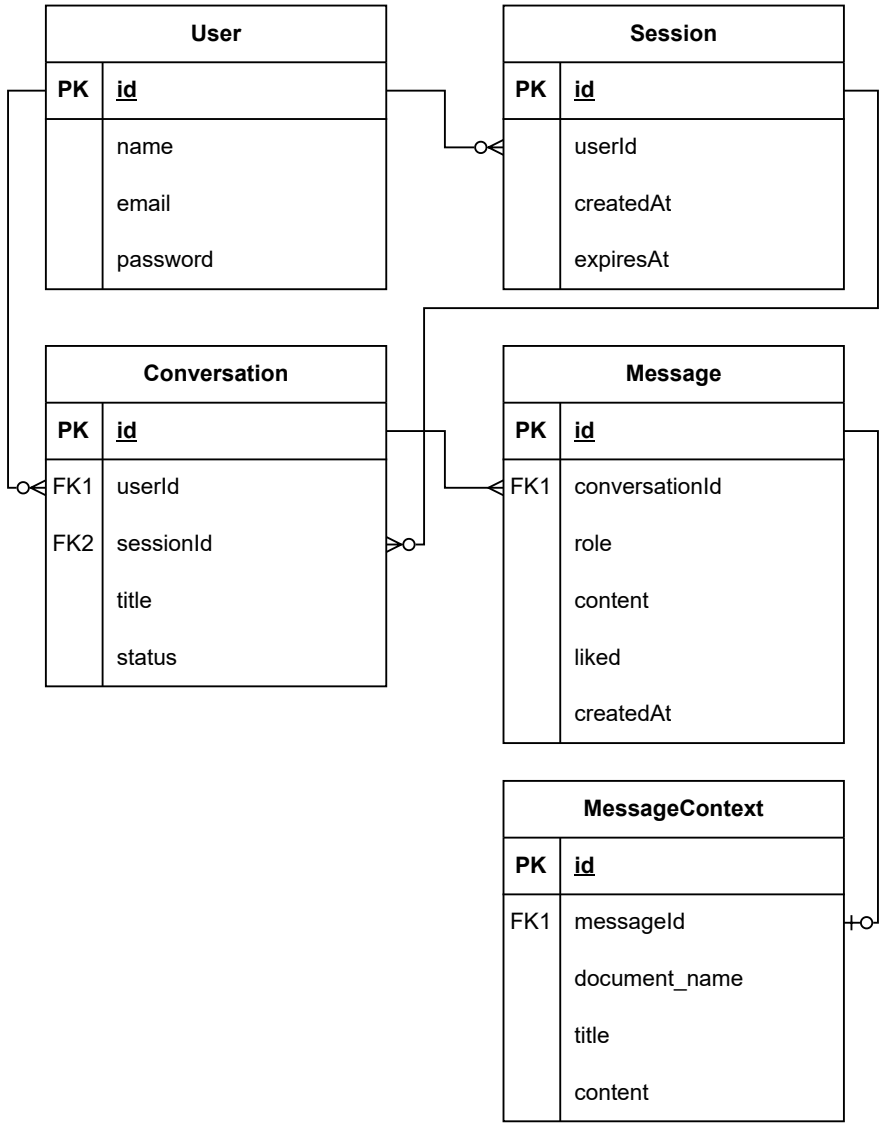


Figura 3.16: Esquema de tablas de la base de datos de la aplicación.

Cada pregunta debe contener la siguiente información: pregunta, respuesta y contexto. La pregunta debe redactarse de forma natural, mientras que la respuesta debe encontrarse directamente en el artículo correspondiente del documento y debe ser copiada directamente del mismo, por último, el contexto corresponde al nombre del artículo de donde fue extraída la pregunta.

La base de datos se genera en formato CSV, por conveniencia de los participantes, pero después se convierte a JSON y complementa con otros campos pertinentes para que sea similar a la estructura del conjunto de datos SQuAD [28]. Posteriormente, el conjunto de datos se carga en la plataforma HuggingFace como un conjunto de datos de QA por conveniencia. Una entrada del conjunto de datos final contiene los siguientes elementos:

- **id:** Identificador único de la pregunta. Es un MD5.
- **title:** Nombre del documento.
- **context:** Nombre del artículo donde se encuentra la respuesta.
- **context_text:** Texto plano del artículo donde se encuentra la respuesta.
- **additional_context:** En caso de que el artículo haga referencia a otro documento o artículo se incluye aquí.
- **question:** Texto de la pregunta.
- **anwers:**
  - **text:** Arreglo con las cadenas de respuesta a la pregunta. Usualmente solo es una.

3.7.2. Evaluación del modelo de *embedding*

El objetivo de esta evaluación es conocer la calidad de los fragmentos identificados como relacionados con cada pregunta. Se emplearán tres métricas: *precision@k*, *recall@k* y *f1@k*. Cada una de estas métricas se definen con las ecuaciones 3.1, 3.2, 3.3 respectivamente. Para aplicar estas fórmulas se necesita calcular la similitud entre la pregunta y todos los fragmentos en la base de datos, compararlos y obtener el top k con mayor similitud. Además,

se debe considerar como fragmento relevante a aquel que realmente contenga la respuesta a la pregunta, lo cual se determina al comparar el nombre del nodo del fragmento con el nombre del artículo marcado como contexto en el conjunto de datos de evaluación.

$$\text{precision@k} = \frac{\text{N fragmentos relevantes en top k}}{k} \tag{3.1}$$

$$\text{recall@k} = \frac{\text{N fragmentos relevantes en top k}}{\text{N fragmentos relevantes totales}} \tag{3.2}$$

$$\text{f1@k} = \frac{2 \times \text{precision} \times \text{recall}}{\text{precision} + \text{recall}} \tag{3.3}$$

El proceso de evaluación de los *embeddings* se realiza siguiendo los siguientes pasos:

1. Para cada pregunta, se identifican los fragmentos relevantes. Es decir, los fragmentos de los artículos donde se encuentra la respuesta.
2. Se calcula la similitud coseno entre la pregunta y cada fragmento de la base de datos de *embeddings*, incluyendo todos los documentos.
3. Se obtienen los k fragmentos con mayor similitud coseno.
4. Se calculan las métricas de evaluación con los k fragmentos obtenidos.
5. El proceso se repite con todas las preguntas del conjunto de datos y al final se obtienen las medias de cada métrica.

3.7.3. Evaluación del modelo de inferencia

El objetivo de esta evaluación es conocer la similitud de la respuesta candidata con la respuesta de referencia, y si la respuesta responde correctamente la pregunta. Se emplearán tres métricas principales: puntuación ROUGE, puntuación BERT y similitud coseno. ROUGE (Recall-Oriented Understudy for Gisting Evaluation) es una métrica que evalúa la superposición entre una

oración candidata y otra de referencia, en específico, ROUGE-L emplea la subsecuencia de texto en común más larga (LCS, Longest Common Subsequence) para calcular la precisión, recall y la puntuación f1, como se muestra en las fórmulas 3.4, 3.5, 3.6.

$$P_{LCS} = \frac{\text{N palabras en LCS}}{\text{N palabras en respuesta inferida}} \tag{3.4}$$

$$R_{LCS} = \frac{\text{N palabas en LCS}}{\text{N palabras en referencia}} \tag{3.5}$$

$$F1_{LCS} = \frac{2 \times P_{LCS} \times R_{LCS}}{P_{LCS} + R_{LCS}} \tag{3.6}$$

La puntuación BERT consiste en tomar la respuesta candidata y la respuesta de referencia, tokenizarla y obtener el *embedding* de cada token con un modelo BERT. Una vez obtenidos los *embeddings*, se calcula la similitud coseno entre cada token de la secuencia candidata $(\hat{x}_1, ..., \hat{x}_k)$ y cada token de la referencia $(x_1, ..., x_m)$, para calcular la precisión y recall empleando los tokens más similares, como se muestra en las ecuaciones 3.7, 3.8 y 3.9.

$$P_{BERT} = \frac{1}{|\hat{x}|} \sum_{\hat{x}_j \in \hat{x}} \max_{x_i \in x} x_i^T \hat{x}_j \tag{3.7}$$

$$R_{BERT} = \frac{1}{|x|} \sum_{x_i \in x} \max_{\hat{x}_j \in \hat{x}} x_i^T \hat{x}_j \tag{3.8}$$

$$F1_{BERT} = \frac{2 \times P_{BERT} \times R_{BERT}}{P_{BERT} + R_{BERT}} \tag{3.9}$$

Por último, la similitud coseno se evalua calculando el *embedding* de la respuesta candidata y el de la respuesta referencia, para ello se emplea un modelo simple de extracción de *embeddings* como el all-MiniLM-L6-v2 y se obtiene la similitud coseno entre ambos vectores.

3.8. Reentrenamiento del modelo de *embeddings*

En este proyecto se emplea la forma más directa de reentrenamiento o ajuste fino, que es el reentrenamiento completo del modelo sobre un conjunto de datos especializados. El conjunto de datos será el descrito anteriormente para la evaluación de los modelos, haciendo una división aleatoria de 80 % de muestras para el entrenamiento y un 20 % para la evaluación. Esta separación se hace a nivel archivo, por lo que se garantiza que exista al menos una pregunta de cada documento en ambos conjuntos de datos.

Para reentrenar los modelos se emplea la librería *SentenceTransformers*, la cual permite cargar un modelo y reentrenarlo con una función de pérdida específica, dependiendo de la tarea. En este caso, la función de pérdida es *CachedMultipleNegativesRankingLoss*, que requiere pares de oraciones (ancla, positivo) o tríos (ancla, positivo, negativo) para maximizar la similitud entre el ancla y la secuencia positiva. Para generar estos pares de oraciones, se considera cada muestra del conjunto de datos original y se emplea la pregunta como el ancla, el texto del contexto como el par positivo, y texto del contexto de otra pregunta como la parte negativa. Se harán pruebas de entrenamiento con solamente ancla y positivo, así como pruebas con el texto negativo también. Sin embargo, la librearía *SentenceTransformers* requiere que tanto el modelo como los ejemplos de entrenamiento sean cargados en la misma GPU para el reentrenamiento, por lo que solo se hará reentrenamiento sobre los modelos all-MiniLM-L6-v2 y el multi-qa-mpnet-base-dot-v1, ya que son los únicos que cumplen este requisito en las GPU de 24 GB de memoria del nodo de supercómputo del CIMAT.

Capítulo 4

Resultados y Análisis

En este capítulo se presentan los resultados obtenidos de la implementación del sistema. Se comienza presentando los resultados obtenidos por el extractor de información y presentando la base de datos de *embeddings* generada. A continuación, se muestra el conjunto de datos de preguntas y respuestas generado, así como los resultados de las evaluaciones de los diferentes modelos de extracción de *embeddings* y de inferencia de respuestas. Posteriormente, se presentan los resultados del reentrenamiento de los modelos de *embeddings*, para finalmente describir el proceso de puesta en producción del sistema. En cada paso se discutirán los resultados mostrados, así como los retos y criterios considerados en la toma de decisiones que llevaron a la configuración final del sistema en producción. Finalmente, se presenta un análisis general del sistema.

4.1. Extractor de información

En la metodología se mencionan dos formas de extraer la información de los documentos normativos, usando *PyPdf+PyTesseract* y empleando *Pdf-*

plumber. Ambos métodos cumplieron con la finalidad de generar una base de datos de *embeddings* con toda la información de los documentos normativos, incluyendo la estrutura de los documentos con sus títulos y secciones. Sin embargo, *Pdfplumber* mostró mayores ventajas, por lo que se seleccionó como el método de extracción de información para el sistema final. En las siguientes subsecciones se presentan algunas de las consideraciones adicionales que se tuvieron que implementar para hacer la correcta extracción de la información así como el razonamiento para seleccionar *Pdfplumber* sobre *PyPdf+PyTesseract*.

4.1.1. Eliminación de encabezados y pies de página

Todos los documentos normativos de la universidad tienen un encabezado con el nombre del documento y un pie de página con el número de página, esta es información irrelevante que se desea eliminar. Para eliminar estos elementos se normalizó a 1.0 el ancho y alto de la página, para posteriormente establecer los márgenes superior, inferior, izquierdo y derecho. Todo el texto que se encuentre fuera de dichos márgenes es ignorado como se muestra en la Figura 4.1. El ancho y alto totales del documento se obtienen directamente de *PyPdf*, *PyTesseract* o *Pdfplumber* según sea el caso. La mayoría de los documentos normativos comparten un márgen de: (izquierda: 0.5, arriba: 0.1, derecha: 0.95, abajo: 0.95), sin embargo, este parámetro es configurable mediante un archivo YAML para dar flexibilidad al procesar documentos con márgenes diferentes.

4.1.2. Detección de títulos centrados

Inicialmente, la detección de títulos centrados se realizó línea por línea, para no tener que detectar los bloques separados verticalmente, sin embargo, hay ocasiones en las que la línea parece estar centrada (recuadro rojo de Figura 4.2), cuando en realidad es parte de una viñeta o de un texto con sangría, es por ello que es necesario hacer el análisis por bloques separados

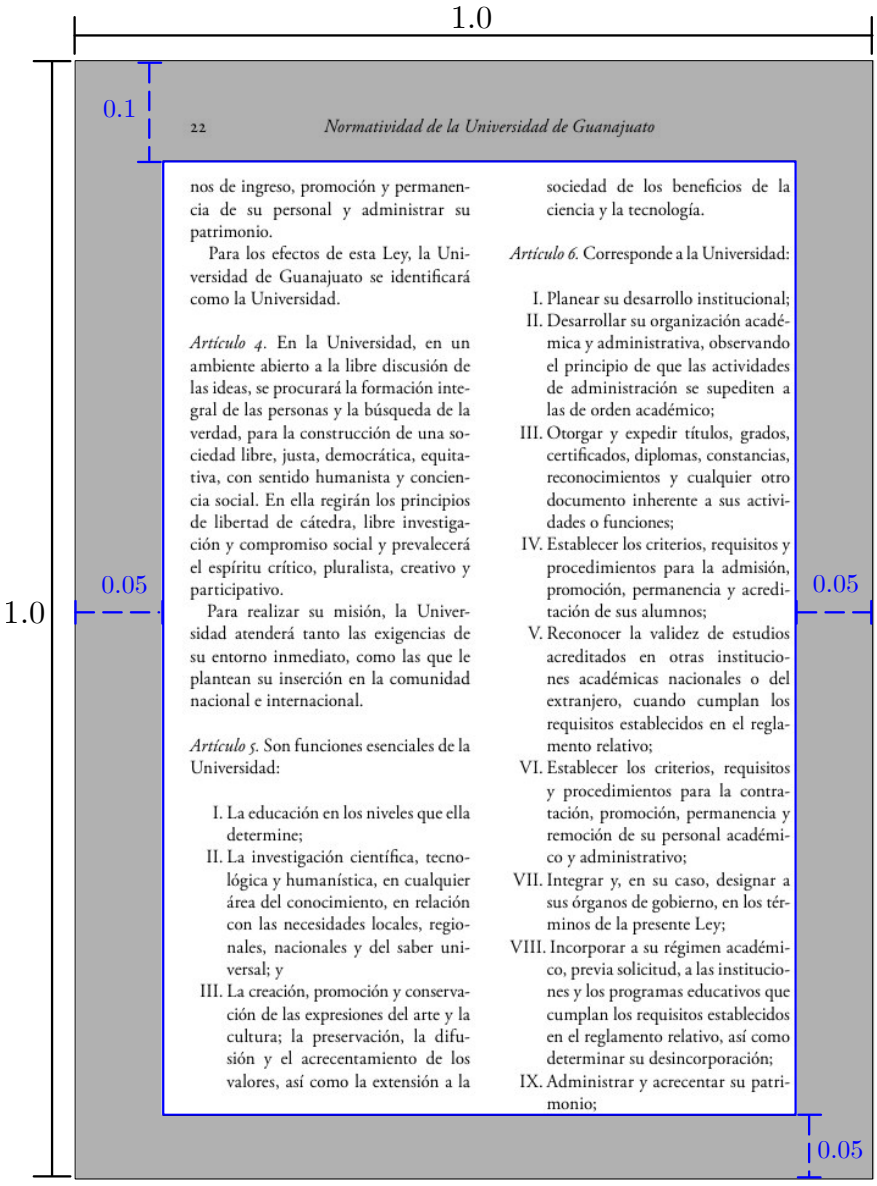


Figura 4.1: Elminación de encabezados y pies de página con márgenes.

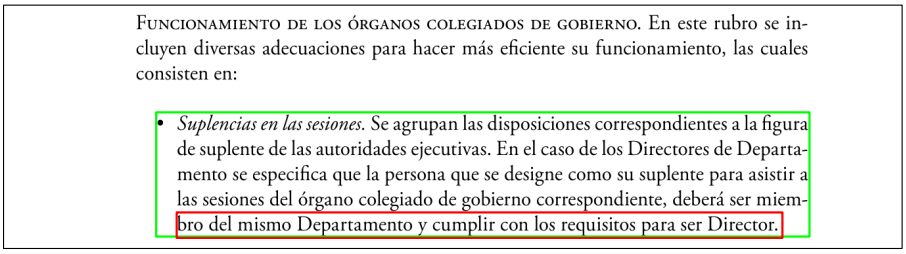


Figura 4.2: Línea de texto aparentemente centrada (recuadro rojo) que al separar el documento en bloques (recuadro verde) ya no se detecta como título.

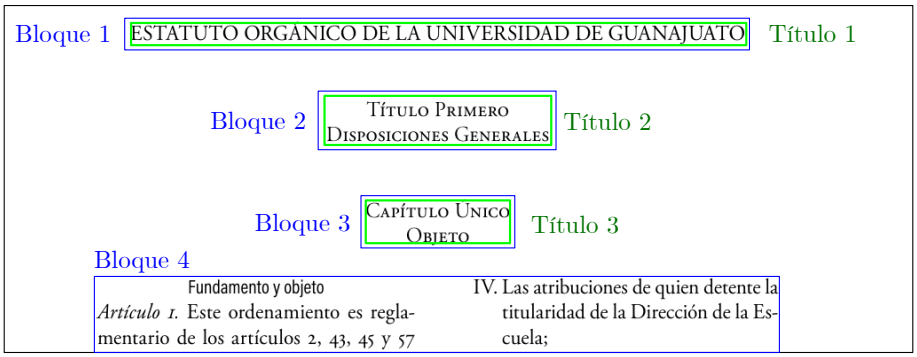


Figura 4.3: Detección correcta de títulos centrados y agrupación en bloques de texto por separación vertical.

verticalmente (recuadro verde de Figura 4.2). Se determina que un bloque está separado verticalmente cuando hay una distancia mayor a un valor configurable, proporcional al alto de la línea como se muestra en la Figura 4.3.

De la misma forma, inicialmente se planteó identificar los títulos centrados en la columna analizando línea por línea, sin embargo, cuando el texto es muy extenso, ya no parece estar centrado, como en la Figura 4.4. Por lo anterior, se recurrió al mismo enfoque donde se detecta el bloque de texto y se da una tolerancia de L líneas para detectar el inicio de un artículo con la expresión regular, de ser así, las l líneas previas son consideradas como un

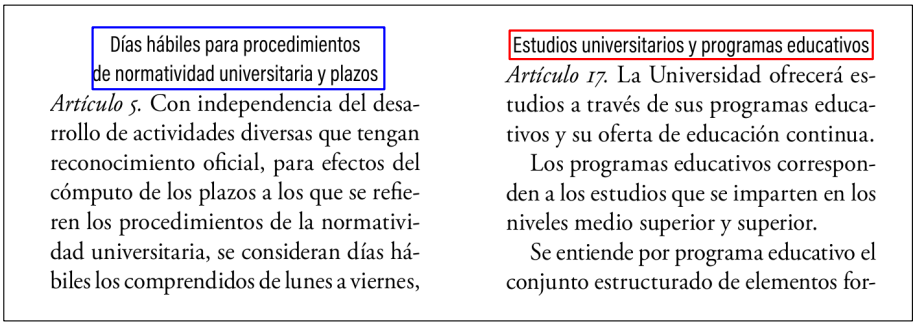


Figura 4.4: Título centrado en columna ideal (izquierda) y título centrado en columna que no puede ser detectado como texto centrado (derecha).

título del artículo.

4.1.3. Detección de títulos no centrados

El documento correspondiente al modelo educativo de la universidad, rompe con la estructura de los demás documentos, pues no es un documento con artículos, sino más bien una guía de la estructura educativa de la institución. Esto no impide que sea dividido en secciones y fragmentos, pero difiere en que el texto está justificado a una columna y sus secciones son numéricas y sin centrar. Para detectar estas secciones no centradas, el texto se divide en bloques verticales y cada bloque se valida con una expresión regular como se muestra en la Figura 4.5. Esta validación no interfiere con la detección de títulos centrados pues ambos procesos se realizan para todos los documentos.

4.1.4. Generación del árbol

La generación del árbol con la estructura del documento se realizó adecuadamente para todos los documentos normativos, sin embargo, se detectó que al generar la estructura del preámbulo de los documentos, ésta puede

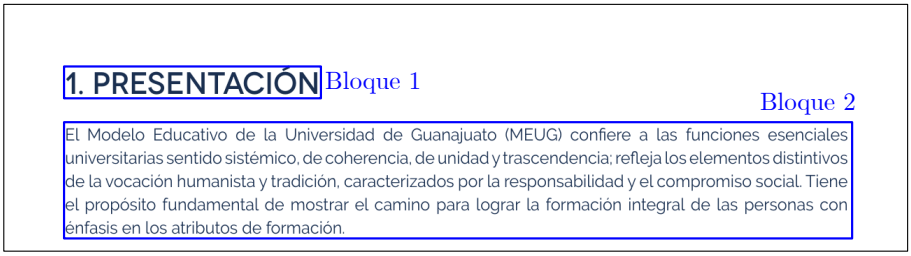


Figura 4.5: Documento con títulos a la izquierda son detectados al separalos en bloques verticales y validar contra una expresión regular.

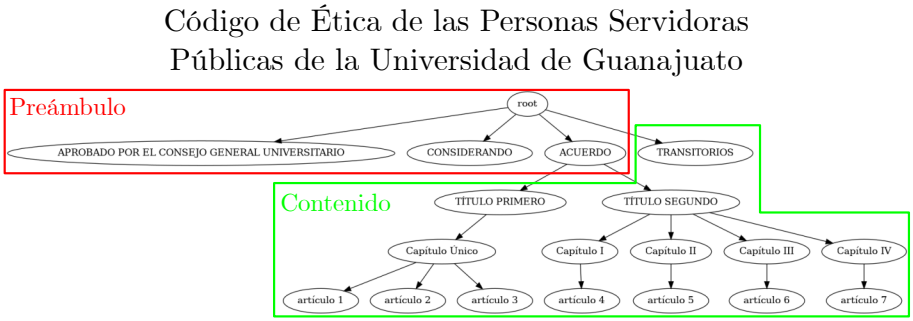


Figura 4.6: Árbol de documento generado automáticamente. Se observa el preámbulo con algunos títulos erroneos (recuadro rojo) y el contenido del documento detectado correctamente (recuadro verde).

presentar algunas estructuras poco congruentes, como se observa en el recuadro rojo de la Figura 4.6. Sin embargo, estas estructuras no afectan el sistema pues la estructura de capítulos y artículos se genera correctamente, como se observa en el recuadro verde de la Figura 4.6, además el preámbulo puede ignorarse si así se deseara pues no contiene información relevante para la normativa en la mayoría de los casos. Adicionalmente, en la Figura 4.7 se muestra parte de la estructura generada para el documento "Modelo Educativo de la Universidad de guanajuato y su Modelo Académico" que, a pesar de que el documento tiene una estructura diferente, puede procesarse adecuadamente.

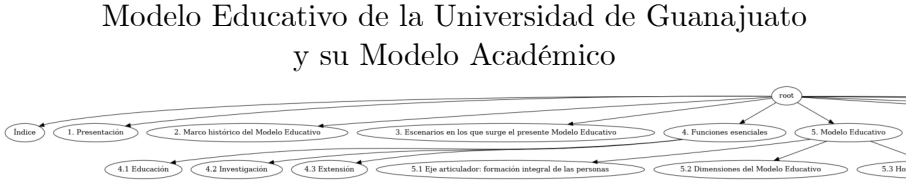


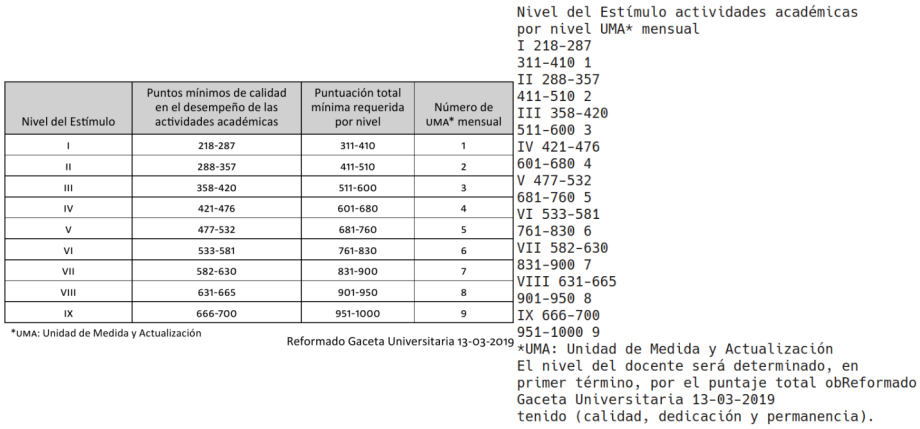
Figura 4.7: Árbol de documento "Modelo Educativo de la Universidad de guanajuato y su Modelo Académico" que es un documento a una columna, justificado y con secciones numéricas.

4.1.5. Separación en fragmentos

Al generar el árbol del documento, colocando un artículo en cada nodo, se encontró que hay artículos cuya longitud superan los 10,000 caracteres, lo cual provoca que la cantidad de recursos necesarios para procesarlos enteros sea muy grande. Es por ello que se hicieron múltiples pruebas para seleccionar el número máximo de caracteres por fragmento en que se debía dividir cada nodo. Estas pruebas consideraron la VRAM disponible, la longitud media y máxima del contenido de los nodos, entre otros factores. Al final, se encontró que se pueden usar adecuadamente los modelos de *embeddings* con 2,048 tokens de contexto, lo que equivale a aproximadamente 8,200 caracteres si se usa el tokenizador de Qwen 3. Por lo anterior, el límite de caracteres por fragmento se estableció en 8,000, ya que al dividirse por párrafos el fragmento más grande resultó de 1,964 tokens.

4.1.6. Detección de tablas

Un problema que no se pudo resolver durante la implementación de esta herramienta es la detección de tablas en los documentos. Una tabla es un elemento que puede ser complejo, especialmente cuando cada celda es de más de un renglón o se mezclan alineaciones de las columnas, todo esto hace que los extractores de texto, tanto *PyPDF*, *PyTesseract* y *Pdfplumber* desorganicen el texto de las tablas, generando secuencias ilegibles como se



Nivel del Estímulo	Puntos mínimos de calidad en el desempeño de las actividades académicas	Puntuación total mínima requerida por nivel	Número de UMA* mensual
I	218-287	311-410	1
II	288-357	411-510	2
III	358-420	511-600	3
IV	421-476	601-680	4
V	477-532	681-760	5
VI	533-581	761-830	6
VII	582-630	831-900	7
VIII	631-665	901-950	8
IX	666-700	951-1000	9

*UMA: Unidad de Medida y Actualización

Reformado Gaceta Universitaria 13-03-2019

```
Nivel del Estímulo actividades académicas
por nivel UMA* mensual
I 218-287
311-410 1
II 288-357
411-510 2
III 358-420
511-600 3
IV 421-476
601-680 4
V 477-532
681-760 5
VI 533-581
761-830 6
VII 582-630
831-900 7
VIII 631-665
901-950 8
IX 666-700
951-1000 9
*UMA: Unidad de Medida y Actualización
El nivel del docente será determinado, en
primer término, por el puntaje total obReformado
Gaceta Universitaria 13-03-2019
tenido (calidad, dedicación y permanencia).
```

Figura 4.8: Extracción de tabla como texto. A la izquierda se muestra la tabla original y a la derecha la secuencia de texto desorganizado.

muestra en la Figura 4.8. Este proceso se acentúa más cuando las tablas están orientadas verticalmente, en lugar de horizontalmente, como es el caso de uno de los documentos de la normativa. Dada esta circunstancia, las partes del documento con tablas verticales fueron eliminadas completamente y las otras tablas que aparecen en los documentos fueron dejadas como están.

4.1.7. *PyPDF+PyTesseract* y Pdfplumber

En general, ambas metodologías fueron capaces de extraer los árboles con la estructura de los documentos PDF, así como su contenido, sin embargo, el enfoque de *PyPDF+PyTesseract* es mucho más lento, pues tarda 5 minutos en procesar un documento de 48 páginas, mientras que *Pdfplumber* demora 45 segundos. En un intento de reducir el tiempo de procesamiento se paralelizó el procesamiento de las páginas, logrando reducir a 2.5 minutos al procesar 48 páginas en paralelo, pudiendo disminuir más con más núcleos de procesamiento. Otra desventaja de *PyPDF+PyTesseract* es que se encontró texto no visible en los documentos PDF de la normativa, especialmente palabras que al obtener el texto plano con *PyPDF* se duplican o triplican como se muestra en la Figura 4.9, lo cual altera considerablemente el algoritmo

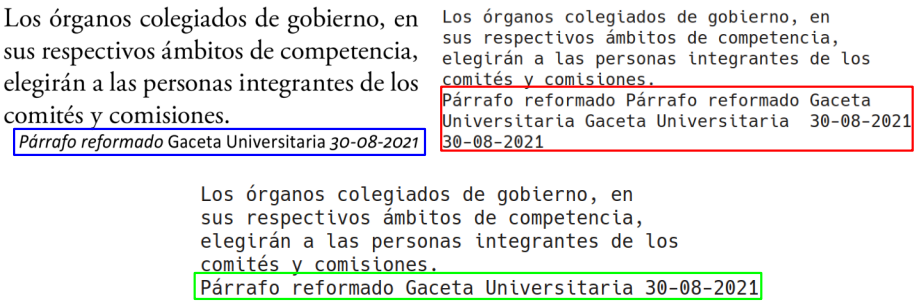


Figura 4.9: Ejemplo de texto oculto. (Arriba a la izquierda) Párrafo como aparece en archivo PDF. (Arriba a la derecha) Texto extraído con *PyPDF*. (Abajo al centro) Texto corregido sin texto duplicado.

y requiere de un proceso adicional para eliminarlas, este proceso se puede hacer directamente con *Pdfplumber* con una bandera de configuración. Sin embargo, una ventaja de *PyPDF+PyTesseract* es que permite al sistema detectar texto escrito en imágenes, aunque en toda la normativa solamente hay un documento que lo tiene y es en su portada, la cual puede simplemente ser ignorada. En la Tabla 4.1 se muestra una comparación entre ambos métodos, donde se observa que emplear únicamente *Pdfplumber* es más beneficioso para el tipo de documentos de la normativa de la universidad.

Característica	*PyPDF+PyTesseract*	*Pdfplumber*
Tiempo de procesamiento	3.12 s/pag	**0.93 s/pag**
Texto en imágenes	**SÍ**	NO
Almacenamiento adicional	SÍ	**NO**

Tabla 4.1: Tabla comparativa de métodos de extracción de información. Pruebas realizadas en computadora con un procesador Intel®Core®i5-8300H CPU @ 2.30GHz

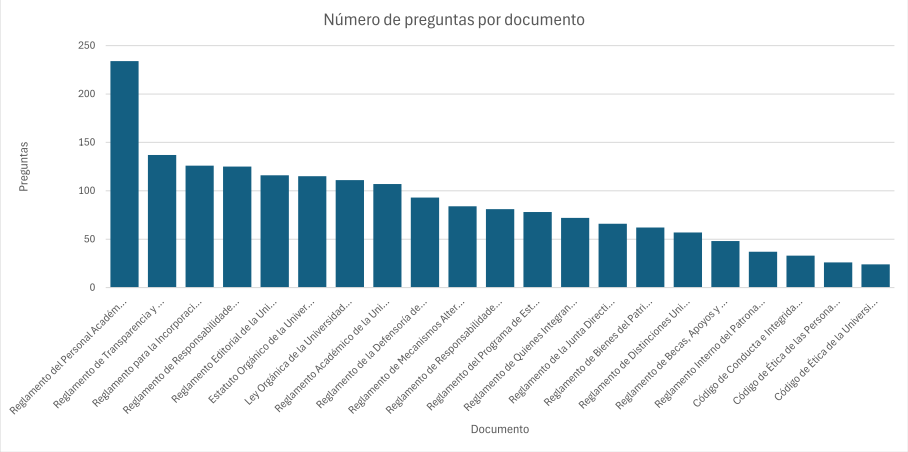


Figura 4.10: Distribución de preguntas por documento.

4.2. Conjunto de datos generado

El conjunto de datos que se generó recopila preguntas y respuestas de 21 documentos diferentes, con un total de 1,082 preguntas y respuestas, la distribución de preguntas por cada documento se muestra en la Figura 4.10. Un ejemplo de un registro del conjunto de datos generado se ve como el siguiente:

- **id:** 81e729afa369916be4d4db99f9c1817c
- **title:** ley-organica-de-la-universidad-de-guanajuato
- **context:** Artículo 1
- **context_text:** La presente Ley es de orden público y de interés social. Contiene las normas fundamentales de la misión, organización, funcionamiento y gobierno de la Universidad de Guanajuato.
- **question:** ¿Qué contiene la Ley Orgánica de la Universidad de Guanajuato?
- **answers:** {‘text’: [‘Contiene las normas fundamentales de la misión, organización, funcionamiento y gobierno de la Universidad de Guanajuato’]}

Durante la generación del conjunto de datos se observaron varias particularidades de los documentos que se tomaron en cuenta para generar las preguntas y se deberán tener en consideración durante la evaluación de los modelos. No fue posible generar preguntas concretas del preámbulo de los documentos, por lo que estas partes fueron omitidas. Otro elemento particular es la presencia de artículos transitorios, los cuales, como su nombre lo indica, son de caracter temporal y solo tienen relevancia cuando recién se publica el documento o la reforma al documento, por lo que incluirlos podría ser innecesario o contradictorio, pues muchos hacen referencia a fechas o tiempos de los cuales el sistema no tiene conocimiento. A pesar de esto, todas las pruebas se realizaron incluyendo estos artículos.

Adicionalmente, en el conjunto de datos se observa cierta tendencia a que las preguntas contengan las palabras clave tal cual están redactadas en los artículos, usando el nombre completo y apropiado para cada elemento, ej: ¿Cuáles son las responsabilidades de la persona titular de la Rectoría General?, cuando sería más natural preguntar ¿Cuáles son las responsabilidades del Rector General?. Esto podría generar preguntas más fáciles de responder y con lenguaje un poco alejado del lenguaje común del usuario final.

Una vez recopiladas las preguntas y respuestas, el conjunto de datos se dividió en dos subconjuntos: entrenamiento (80 %) y prueba (20 %). El objetivo de esta división es emplear el conjunto de prueba durante la evaluación de los modelos de *embeddings* reentrenados, sin embargo, para las demás evaluaciones se empleó el conjunto de datos completo para evaluar. El conjunto de datos se colocó en un repositorio de HuggingFace con el objetivo de facilitar su uso en los diferentes equipos de trabajo, así como su publicación. Usar esta estrategia demostró ser efectiva pues se pueden conservar diferentes versiones del conjunto de datos de forma organizada, así como los subconjuntos de entrenamiento y prueba.

4.3. Evaluación y selección del modelo de *embeddings*

Se evaluaron tres modelos diferentes: MiniLM, MPNET y Qwen 3. Del modelo Qwen 3 se evaluaron sus tres variantes de tamaño: 0.6B, 4B y 8B, para cada variante se evaluó además su versión cuantizada disponible más pequeña y la más grande. Adicionalmente, se evaluaron los modelos con k=3 y k=5 fragmentos similares, dando como resultados los mostrados en las Tablas 4.2 y 4.3 para top-3 y top-5 de fragmentos similares respectivamente. Se debe considerar que la precisión y la puntuación f1 son naturalmente bajos puesto que cada pregunta del conjunto de datos fue formulada para que tenga un solo artículo relevante, es decir, el máximo de fragmentos relevantes por pregunta es 1 o 2 (dado que unos pocos artículos son de más de un fragmento), de ahí que la mejor métrica para comparar los modelos sea únicamente el recall.

Si se compara el recall (Figuras 4.11, 4.12), se puede observar que el mejor modelo es el Qwen3-Embedding-8B, seguido de sus versiones cuantizadas. Este resultado nos permite seleccionar al modelo Qwen-Embedding-8B para las siguientes pruebas, así como a su modelo cuantizado Q4_K_M de 4 bits, que con mucha menos memoria alcanza un rendimiento similar. Por otra parte, se observa la considerable diferencia de 0.63 y 0.5 puntos de recall entre el mejor modelo y los modelos all-MiniLM-L6-v2 y multi-qa-mpnet-base-dot-v1 respectivamente. Además, observamos que existe una diferencia de aproximadamente 0.8 puntos entre las pruebas con el top-5 de fragmentos similares respecto al top-3, esto es indicativo de que no siempre el fragmento relevante real está dentro de los tres fragmentos más similares, por lo que alimentar el modelo de inferencia con 5 fragmentos de contexto entregará mejores resultados sacrificando un poco de memoria para el contexto.

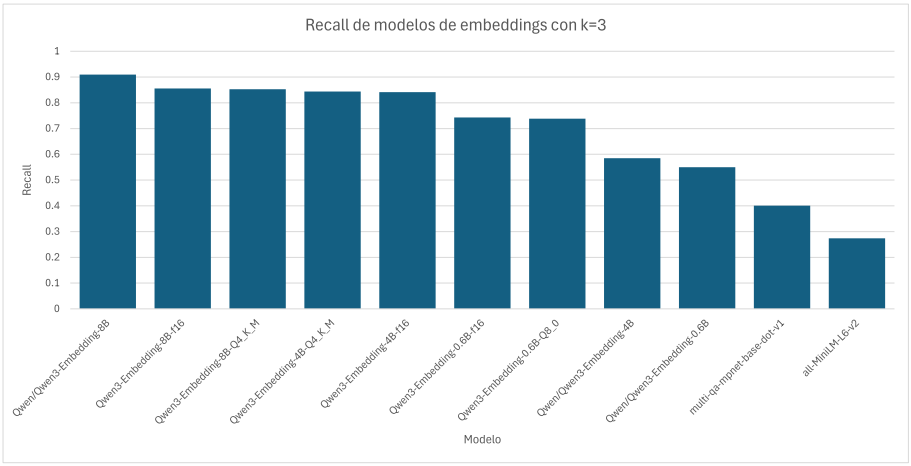


Figura 4.11: Recall para *embeddings* con el top-3 de fragmentos similares.

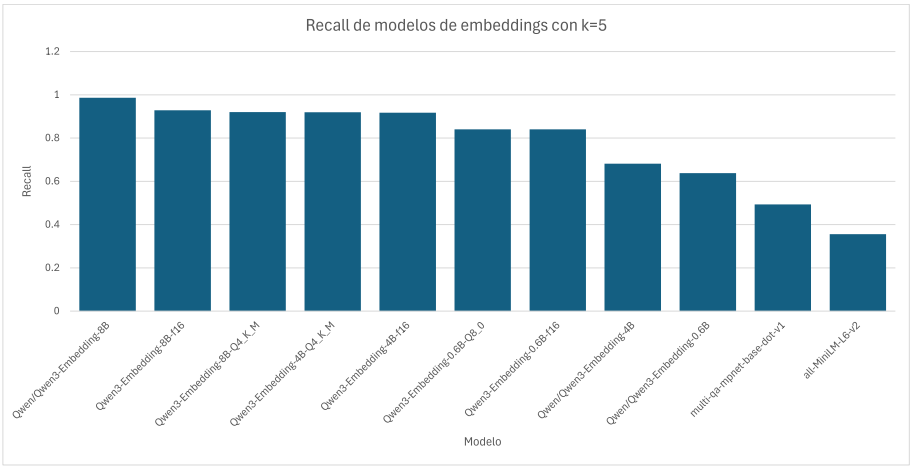


Figura 4.12: Recall para *embeddings* con el top-5 de fragmentos similares.

Modelo de *embeddings*	precision@3	recall@3	f1@3
Qwen3-Embedding-0.6B	0.184	0.549	0.274
Qwen3-Embedding-0.6B-f16	0.248	0.743	0.371
Qwen3-Embedding-0.6B-Q8_0	0.247	0.738	0.369
Qwen3-Embedding-4B	0.195	0.584	0.292
Qwen3-Embedding-4B-Q4_K_M	0.283	0.843	0.422
Qwen3-Embedding-4B-f16	0.282	0.841	0.420
Qwen3-Embedding-8B	**0.304**	**0.909**	**0.454**
Qwen3-Embedding-8B-Q4_K_M	0.285	0.852	0.426
Qwen3-Embedding-8B-f16	0.286	0.855	0.427
all-MiniLM-L6-v2	0.091	0.273	0.136
multi-qa-mpnet-base-dot-v1	0.133	0.400	0.200

Tabla 4.2: Resultados de evaluación para modelos de *embeddings* con el top-3 de fragmentos similares.

4.4. Selección de modelo de inferencia

Para la inferencia se evaluaron tres modelos diferentes: Qwen3 en sus versiones 0.6B y 8B, tanto su versión completa como su versión cuantizada de Q4_K_M, el modelo GPT-OSS 20B y el modelo Llama 3.1 8B. Adicionalmente se evaluó el modelo de Qwen3 8B con la opción de razonamiento activada. Estas evaluaciones fueron ejecutadas en el centro de supercómputo del CIMAT, con 48 GB de memoria de video disponibles, además, todos los modelos fueron evaluados con el top-5 de fragmentos similares, así como con el mejor modelo de *embeddings*, es decir, el modelo Qwen3-Embedding-8B. Los resultados de esta evaluación se muestran en la Tabla 4.4 donde se observa que los mejores modelos son Qwen3-8B y Qwen3-0.6B, en sus versiones completas. Nótese el rendimiento es casi igual entre estos dos modelos y la versión cuantizada de Qwen3-8B-Q4_K_M, aunque esta última requiere más memoria que Qwen3-0.6B, pero menos que Qwen3-8B.

Modelo de *embeddings*	precision@5	recall@5	f1@5
Qwen3-Embedding-0.6B	0.128	0.637	0.212
Qwen3-Embedding-0.6B-Q8_0	0.169	0.840	0.280
Qwen3-Embedding-0.6B-f16	0.169	0.840	0.280
Qwen3-Embedding-4B	0.137	0.681	0.227
Qwen3-Embedding-4B-Q4_K_M	0.186	0.919	0.306
Qwen3-Embedding-4B-f16	0.185	0.917	0.305
Qwen3-Embedding-8B	**0.198**	**0.986**	0.329
Qwen3-Embedding-8B-Q4_K_M	0.185	0.919	0.306
Qwen3-Embedding-8B-f16	0.187	0.928	0.309
all-MiniLM-L6-v2	0.071	0.356	0.118
multi-qa-mpnet-base-dot-v1	0.099	0.493	0.164

Tabla 4.3: Resultados de evaluación para modelos de *embeddings* con el top-5 de fragmentos similares.

Modelo de inferencia	RougeL	Punaje BERT	Similitud coseno
Qwen3-0.6B	0.412	0.774	0.677
Qwen3-8B	**0.424**	**0.772**	**0.690**
Qwen3-8B-Q4_K_M-think	0.213	0.676	0.610
Qwen3-8B-Q4_K_M	0.410	0.766	0.686
gpt-oss-20b-MXFP4	0.322	0.707	0.668
Llama-3.1-8B-Instruct	0.351	0.750	0.645

Tabla 4.4: Resultados de evaluación de modelos de inferencia con Qwen3-Embedding-8B y top-5 de fragmentos similares.

Una vez obtenidos los resultados de la Tabla 4.4, se debe considerar la cantidad de memoria disponible en el servidor *Dell Precision 7920 Tower*, que es de 16GB+2GB+2BG, y se determina que el modelo Qwen3-8B no

puede ejecutarse ahí, por lo que para el siguiente paso de evaluación se escogieron los tres mejores modelos siguientes, que son los modelos Qwen3-0.6B, Qwen3-8B-Q4_K_M y GPT-OSS-20B. La siguiente prueba fue mejorar el comando de sistema que se le proporciona al modelo para hacer ajuste por ingeniería de comandos. Los resultados de esta evaluación se presentan en la Tabla 4.5, donde también se evaluó el modelo Qwen3-8B-Q4_K_M con su opción de razonamiento activada para analizar si reacciona mejor al haber un comando de sistema más complejo.

Modelo	RougeL	Puntaje BERT	Similitud coseno
Qwen3-0.6B	0.319	0.745	0.626
Qwen3-8B-Q4_K_M	0.368	0.747	0.670
Qwen3-8B-Q4_K_M-think	0.241	0.691	0.622
gpt-oss-20b-MXFP4	**0.420**	**0.753**	**0.707**

Tabla 4.5: Resultados de evaluación de mejores modelos de inferencia con comando de sistema más complejo.

En la Tabla 4.5 se observa que el modelo GPT-OSS-20B es el que tiene mejor rendimiento, mostrando una mejora contra la evaluación sin instrucción de sistema. Por su parte, los modelos Qwen3 empeoraron ligeramente su puntaje al introducir el comando del sistema. Por último, el modelo Qwen3 con la opción de razonamiento mejoró ligeramente comparado con su evaluación sin instrucción, pero no significativamente. Todo lo anterior nos inclina a pensar que el modelo GPT-OSS-20B es el que mejor se adapta a las instrucciones de sistema, para verificarlo se procedió a hacer una evaluación manual de las respuestas. Para la evaluación manual una persona debe, leer la pregunta y comparar la respuesta del modelo contra la respuesta de referencia, para decidir si la respuesta responde correctamente a la pregunta, según su criterio. Para esta evaluación se verificaron 25 respuestas aleatorias de los modelos, evaluando la misma pregunta para todos los modelos con el fin de comparar otros factores como su legibilidad o estilo.

En la Tabla 4.6 se muestra el porcentaje de respuestas correctas para

cada uno de los cuatro modelos, y se observa que el modelo Qwen3-0.6B es inferior al resto por un margen considerable, mientras que el resto se desempeña de una forma similar.

Modelo	Aciertos
Qwen3-0.6B	60 %
gpt-oss-20b-MXFP4	**88** %
Qwen3-8B-Q4_K_M	84 %
Qwen3-8B-Q4_K_M-think	**88** %

Tabla 4.6: Porcentaje de respuesta correctas por modelo en una muestra de 25 preguntas aleatorias.

Durante el análisis de las respuestas se encontró que el modelo Qwen3-0.6B tiende a cometer ligeras imprecisiones en las respuestas, como omitir una fracción, agregar texto que no tiene que ver con la pregunta o escribir texto incoherente, lo que provoca que el evaluador califique la respuesta como incorrecta, además de cometer errores de escritura de las palabras con acentos o formateo incorrecto de las respuestas, esto justificaría la discrepancia entre la evaluación cuantitativa del modelo y la evaluación cualitativa. Por su parte, la mejora de rendimiento del modelo Qwen3-8B-Q4_K_M-think se debe a que el modelo genera respuestas muy extensas, donde el texto adicional abona contexto a la respuesta incluyendo detalles adicionales, por lo que un evaluador puede dar por buena la respuesta aunque textualmente no sea similar a la respuesta de referencia. Sin embargo, se observó que en muchas ocasiones las cadenas de razonamiento son innecesariamente largas.

Con las consideraciones antriores, se decide descartar el modelo Qwen3-0.6B por sus imprecisiones y errores, así como el modelo Qwen3-8B-Q4_K_M porque, si bien las respuestas son correctas, el texto adicional agregado es una distracción innecesaria. Se hicieron pruebas adicionales, menos rigurosas sobre los modelos restantes y se determinó que el modelo GPT-OSS reacciona mejor a los comandos de sistema, es mejor siguiendo instrucciones y en general su formato de salida es más amigable, es por ello que se seleccionó

este modelo como el modelo de inferencia final para el sistema.

4.5. Reentrenamiento de modelo de *embeddings*

Dos modelos de embeddings se sometieron a un proceso de reentrenamiento: all-MiniLM-L6-v2 y multi-qa-mpnet-base-dot-v1. Ambos modelos fueron reentrenados con el subconjunto de datos de entrenamiento, del cual se extrajo un 10 % adicional para validación. El reentrenamiento se llevó a cabo en el centro de supercómputo del CIMAT en uno de los nodos con dos GPUs utilizando *torchrun* para distribuir la carga entre ambos, aquí se buscaron los mejores parámetros de entrenamiento para cada modelo, dando como resultado los parámetros de la Tabla 4.7.

Modelo	Épocas	Batch	LR	WR	ES
all-MiniLM-L6-v2	20	32	2.00E-05	0.1	3
multi-qa-mpnet-base-dot-v1	15	32	1.00E-05	0.05	3

Tabla 4.7: Configuraciones de entranamiento de los modelos. Tasa de aprendizaje (LR), razón de calentamiento (WR), detención temprana (ES).

Con el fin de analizar la calidad del entrenamiento, se generaron las gráficas de la evolución de la función de pérdida para el entrenamiento de ambos modelos. Para el entrenamiento con pares ancla-positivo, las gráficas se muestran en las Figuras 4.13 y 4.14 para los modelos all-MiniLM-L6-v2 y multi-qa-mpnet-base-dot-v1 respectivmente. Para el entrenamiento con tríos ancla-positivo-negativo, las gráficas se muestran en las Figuras 4.15 y 4.16. En todas las gráficas se observa que el mejor modelo se alcanza tras pocas épocas.

Para hacer la evaluación de los modelos, se siguió la misma metodología de evaluación de extracción de fragmentos, obteniendo el *precision*, *recall* y *f1* con k=3 y k=5. Con el fin de comparar los modelos base y los entrenados, se realizó una evaluación con los modelos base sobre el conjunto de datos de

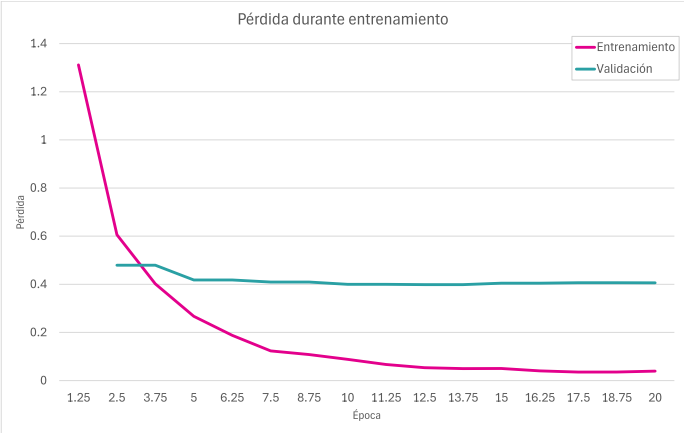


Figura 4.13: Función de pérdida para entrenamiento de pares ancla-positivo con modelo all-MiniLM-L6-v2.

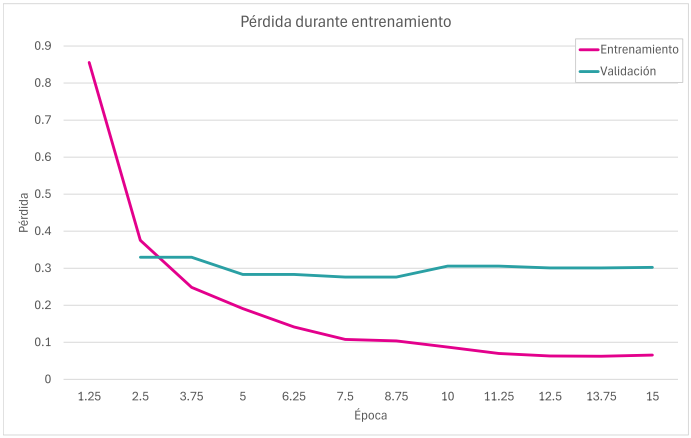


Figura 4.14: Función de pérdida para entrenamiento de pares ancla-positivo con modelo multi-qa-mpnet-base-dot-v1.

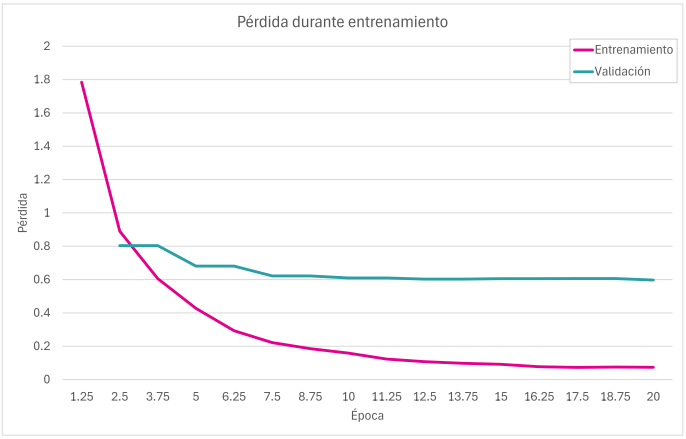


Figura 4.15: Función de pérdida para entrenamiento de tríos ancla-positivo-negativo con modelo all-MiniLM-L6-v2.

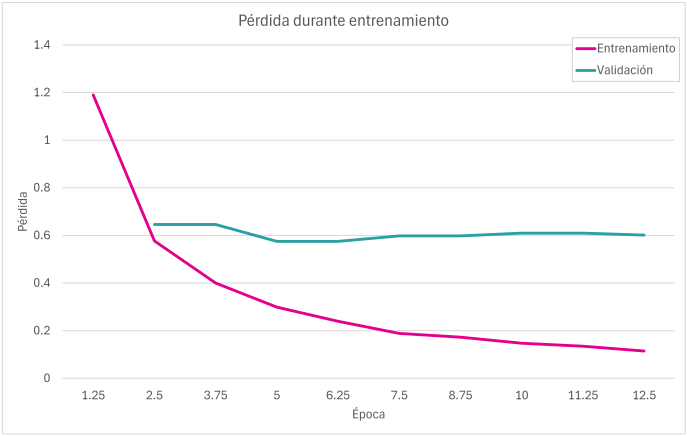


Figura 4.16: Función de pérdida para entrenamiento de tríos ancla-positivo-negativo con modelo multi-qa-mpnet-base-dot-v1.

prueba, así como una evaluación contra el modelo Qwen3-Embedding-8B-Q4_K_M. La evaluación se muestra en la Tabla 4.8 para k=3 en la Tabla 4.9 para k=5, donde se observa que ambos modelos reentrenados tienen una mejora de 0.32 y 0.25 puntos de recall respecto a sus modelos base, además, se observa que no hay una diferencia significativa entre el entrenamiento por tríos y el de pares, aunque esto puede deberse a la mala calidad de los ejemplos negativos, pues fueron escogidos al azar. A pesar de la mejora de los modelos reentrenados, el modelo Qwen3-Embedding-8B-Q4_K_M sigue siendo superior por un margen de 0.08 puntos de recall, por lo que será el que se use en el sistema final.

Modelo	precision@3	recall@3	f1@3
all-MiniLM-L6-v2	0.143	0.430	0.215
multi-qa-mpnet-base-dot-v1	0.175	0.522	0.261
all-MiniLM-L6-v2 (pares)	0.255	0.751	0.376
all-MiniLM-L6-v2 (tríos)	0.253	0.744	0.372
multi-qa-mpnet-base-dot-v1 (pares)	0.259	0.768	0.384
multi-qa-mpnet-base-dot-v1 (tríos)	0.262	0.776	0.389
Qwen3-Embedding-8B-Q4_K_M	**0.290**	**0.854**	**0.428**

Tabla 4.8: Resultados de evaluación de modelos para k=3.

4.6. Puesta en producción

La puesta en producción del sistema consta de tres componentes: un servidor con GPUs donde se ejecuten los modelos LLM, un servidor en la nube donde se ejecute la aplicación web y una VPN (Virtual Private Network) para comunicar ambos componentes. Para el servidor con GPUs, se emplea la *Dell Precision 7920 Tower* que se encuentra dentro de las instalaciones de la División de Ingenierías del Campus Irapuato-Salamanca (DICIS) de la Universidad de Guanajuato, en este servidor se coloca el extractor de in-

Modelo	precision@5	recall@5	f1@5
all-MiniLM-L6-v2	0.102	0.509	0.170
multi-qa-mpnet-base-dot-v1	0.121	0.601	0.200
all-MiniLM-L6-v2 (pares)	0.177	0.868	0.290
all-MiniLM-L6-v2 (tríos)	0.177	0.863	0.288
multi-qa-mpnet-base-dot-v1 (pares)	0.180	0.888	0.296
multi-qa-mpnet-base-dot-v1 (tríos)	0.174	0.857	0.286
Qwen3-Embedding-8B-Q4_K_M	**0.192**	**0.938**	**0.313**

Tabla 4.9: Resultados de evaluación de modelos para k=5.

formación, el modelador del lenguaje y la API de comunicación. El servidor en la nube corresponde a una máquina virtual de Azure del tipo *Standard B2pts v2 (2vcpus, 1GiB memory)*, con Ubuntu 24.04, donde se ejecuta la aplicación web. Este servidor debe ser visible desde internet, por lo se le asigna una IP pública. Por último, la VPN se configura en Azure para conectar directamente con la máquina virtual, mientras que la *Dell Precision 7920 Tower* se conecta a través de una puerta de enlace VPN. El esquema completo de conexiones se encuentra en la Figura 4.17.

4.6.1. Servidor GPUs

Este servidor contiene la API de comunicación, el modelador de lenguaje y el extractor de información. El servidor se configuró con Docker para hacer el despliegue de los contenedores necesarios, los cuales se muestran en la Figura 4.18. Se creó una imagen de contenedor llamada *normativity_rag*, la cual contiene el extractor de información. Este contenedor tiene la capacidad de ejecutar todos los scripts relacionados con la extracción de información, incluido un script que toma todos los documentos PDF de un directorio y realiza el proceso de extracción para generar la base de datos en ChromaDB con los *embeddings*. Hacerlo de esta forma permite usar llama_cpp en Win-

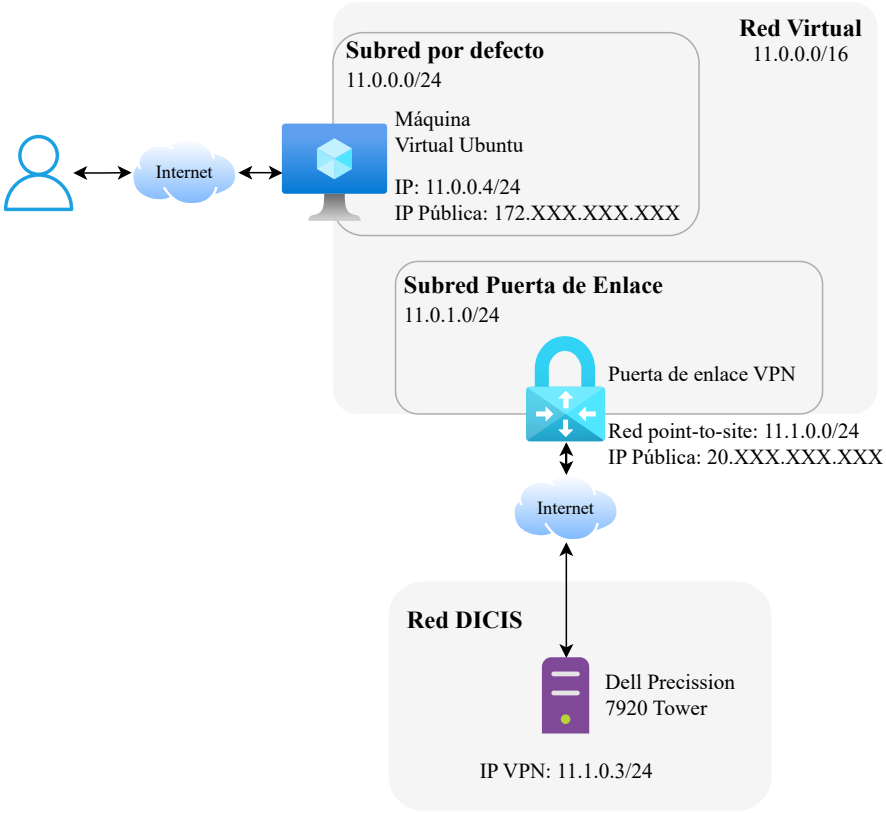


Figura 4.17: Diagrama de conexión de sistema.

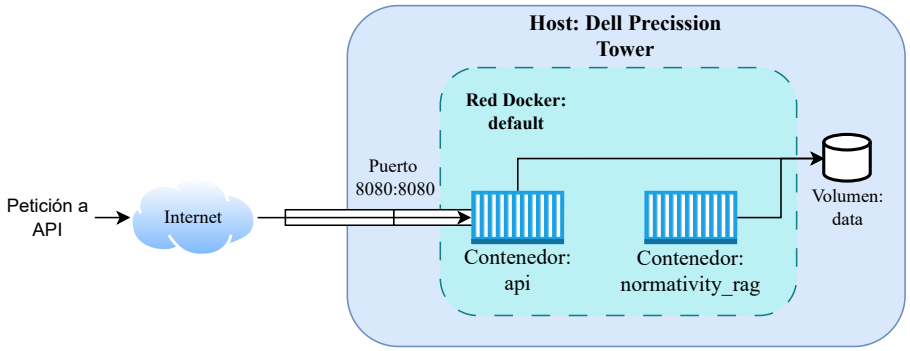


Figura 4.18: Diagrama de contenedores en servidor *Dell Precision Tower*.

dows, además proporciona las ventajas propias de los contenedores Docker, como son la posibilidad de ejecutar la misma imagen en equipos diferentes sin instalar dependencias. En el Apéndice C se muestra el archivo *Dockerfile* con la configuración del contenedor.

En cuanto al modelador del lenguaje y la API de comunicación, ambos se incluyen en otro contenedor bajo el nombre *llm_api*. A este contenedor se le mapea el puerto 8080 del servidor para poder acceder a la API desde la VPN. Para una administración más sencilla del contenedor se crea un archivo *docker-compose.yml* en el que se indica el contenedor a usar, el puerto a mapear y el volumen a cargar donde se encuentra la base de datos de la API y los modelos descargados. Este contenedor requiere la inicialización de la base de datos, así como la creación de al menos una API-KEY para acceder, para ello se ejecuta un script en python dentro del contenedor. En el Apéndice C se muestra el *Dockerfile* del contenedor, así como el archivo *docker-compose.yml* con las instrucciones de creación del servicio. En la Figura 4.19 se muestra una captura de pantalla del administrador de contenedores donde se observa el servicio de la API en funcionamiento.

El contenedor *llm_api* carga los modelos de *embeddings* e inferencia en la tarjeta gráfica principal cuando se realiza la primera petición a la API y utiliza los mismos modelos para todas las peticiones, por lo que es un posible cuello de botella cuando se tienen multiples usuarios conectados. El modelo seleccionado para el cálculo de *embeddings* es el Qwen3-Embedding-8B-

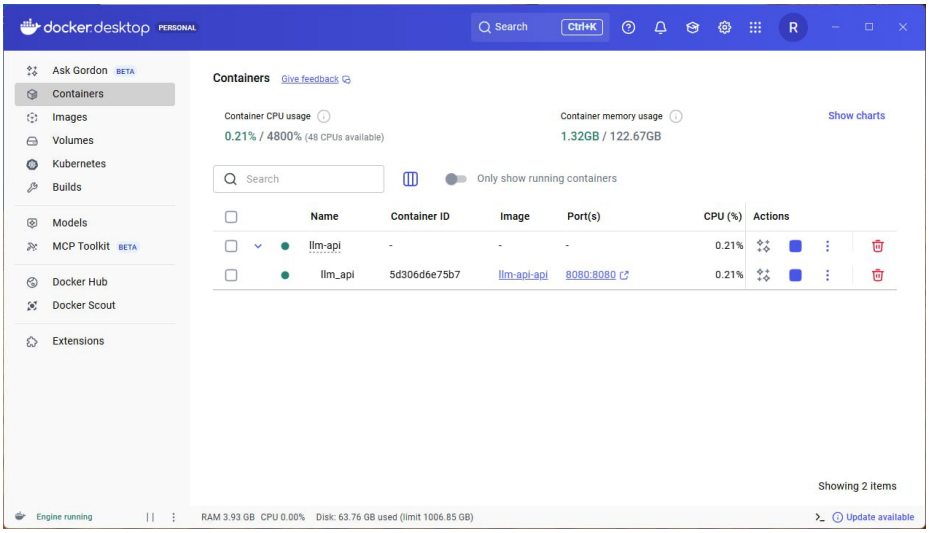


Figura 4.19: Captura de pantalla de administrador de contenedores en servidor con API en ejecución.

Q4_K_M, mientras que el modelo de inferencia es el GPT-OSS-20B-MXFP4, ambos modelos ocupan 17.6 GB de VRAM, por lo que usan la tarjeta RTX A4000 del servidor en su totalidad y además hacen uso de 2 GB de memoria compartida de la RAM como se muestra en la captura de la Figura 4.20. No se realizaron pruebas para medir el impacto del uso de la memoria compartida pues no se notó lentitud en el sistema, además de que el modelo que es cargado parcialmente en RAM es el modelo de *embeddings* que es mucho más rápido. Se intentó usar las otras dos GPUs del sistema pero no se logró establecer una configuración satisfactoria con la librería *llama_cpp*. Adicionalmente, se debe habilitar la conexión al puerto 8080 con una regla del firewall de Windows, como se muestra en la captura de la Figura 4.21.

La administración de la conexión a la VPN se hace a través de la aplicación “Azure VPN”, la cual emplea una llave privada, que hay que configurar en el sistema, y un archivo de configuración para conectarse a la puerta de enlace VPN configurada en Azure, en la Figura 4.22 se muestra una captura de pantalla de la interfaz de conexión. Una desventaja de esta estructura es que la conexión a la VPN tiene que hacerse manualmente al encender el servidor, además, cuando existen problemas de conexión de internet, es posible

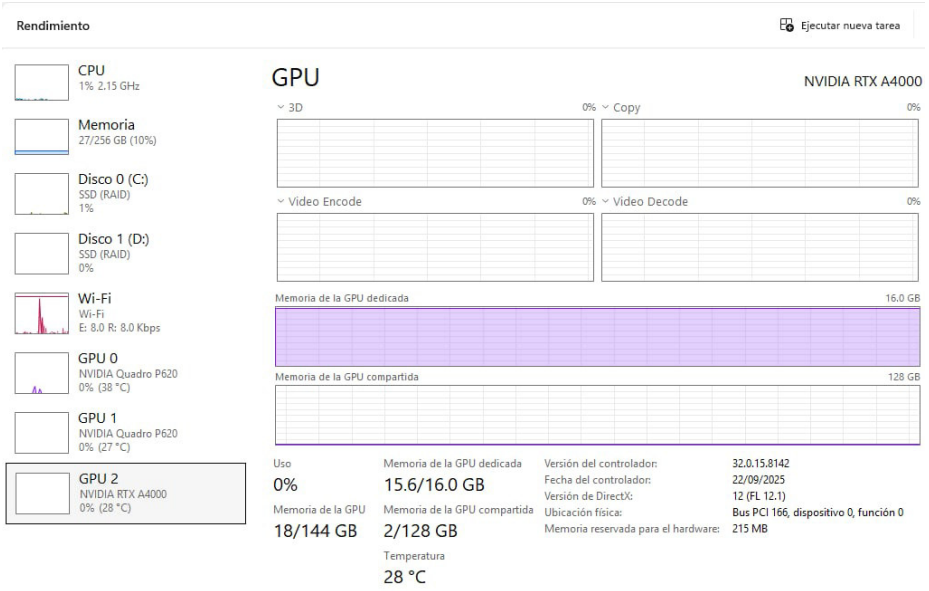


Figura 4.20: Captura de pantalla de administrador de recursos de Windows donde se muestra el uso de la GPU al cargar los modelos.

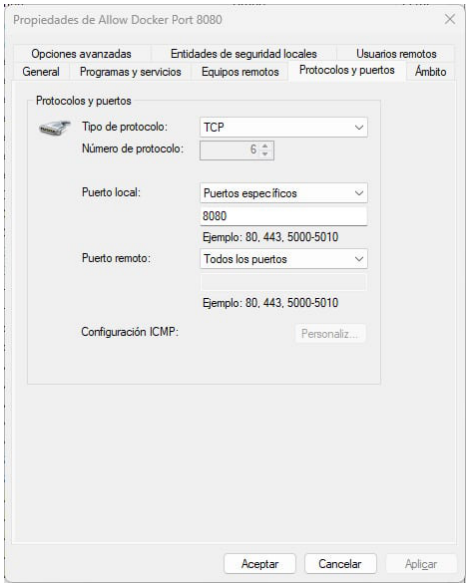


Figura 4.21: Captura de pantalla de regla de Firewall de Windows para permitir el acceso al puerto 8080.

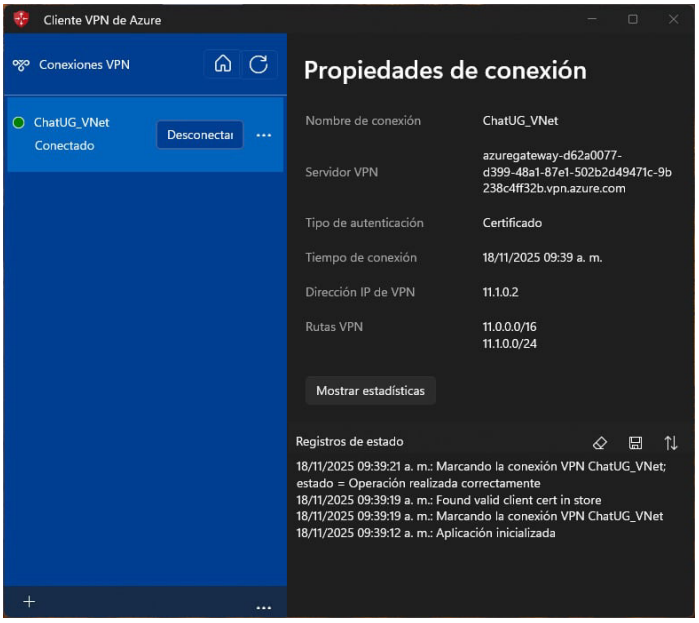


Figura 4.22: Captura de pantalla de administrador conexión a VPN de Azure desde servidor en una red local.

que la VPN se desconecte y deba reconectarse manualmente. Otro problema es que Azure no ofrece una forma directa de configurar una IP estática para un equipo específico, por lo que cada vez que se reconecta la VPN puede asignar una IP diferente al servidor y se debe actualizar esta configuración en la aplicación. Adicionalmente, al emplear un servidor local sin medidas para garantizar su disponibilidad, éste puede presentar intermitencia de conexión de red o fallas en el sumministro de luz.

4.6.2. Máquina virtual en la nube

En la máquina virtual de la nube también se instala Docker para ejecutar los contenedores requeridos por la aplicación, como se muestra en la Figura 4.23. La aplicación web se coloca en un contenedor con el puerto 3000 expuesto, este contenedor se crea con la aplicación construida para producción con *npm*. La base de datos de la aplicación se coloca en otro contenedor con

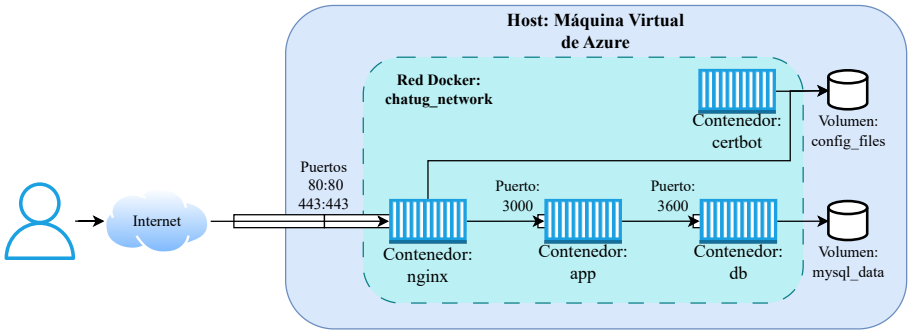


Figura 4.23: Diagrama de contenedores en máquina virtual en la nube.

```
[redacted]@VMChatUG:~$ docker ps
CONTAINER ID   IMAGE             COMMAND                  CREATED       STATUS
PORTS                                                                    NAMES
2f06219d9dc9   rivert97/chatug   "docker-entrypoint.s…"   2 weeks ago   Up 2 days
3000/tcp                                                                 chatug
432a1f53ec49   nginx:latest      "/docker-entrypoint.…"   3 weeks ago   Up 2 days
0.0.0.0:80->80/tcp, [::]:80->80/tcp, 0.0.0.0:443->443/tcp, [::]:443->443/tcp   nginx_proxy
08b401244f7e   mysql:8.0         "docker-entrypoint.s…"   3 weeks ago   Up About an hour
3306/tcp, 33060/tcp                                                      mysql_db
```

Figura 4.24: Captura de pantalla de contenedores de aplicación web en ejecución en servidor en la nube.

el puerto 3306 expuesto, para este contenedor se utiliza la imagen base de MySQL 8.0 y se configura con un usuario adicional que solo tenga acceso a la base de datos de la aplicación. Un tercer contenedor con Nginx se crea para fungir como proxy inverso y controlar el acceso a la aplicación, para esto se usa la imagen base de Nginx 1.29.3. Por último, para configurar los certificados SSL de Let's Encrypt se usa un contenedor temporal que ejecuta la herramienta Certbot. Para facilitar el lanzamiento de los contenedores se creó un archivo *docker-compose.yml* con la configuración para obtener la arquitectura deseada, el cual se encuentra en el Apéndice C, junto con el *Dockerfile* del contenedor con la aplicación. En la captura de la Figura 4.24 se observan los tres contenedores en ejecución en la máquina virtual en la nube.

Además, se deben configurar reglas en el firewall para solo permitir el acceso por el puerto 22 (SSH), 80 (http) y 433 (https), para ello se utiliza el firewall de la máquina virtual (UFW) y se configura como se muestra en

```
[redacted]@VMChatUG:~$ sudo ufw status
Status: active

To                         Action      From
--                         ------      ----
22/tcp                     ALLOW       Anywhere
80/tcp                     ALLOW       Anywhere
443/tcp                    ALLOW       Anywhere
22/tcp (v6)                ALLOW       Anywhere (v6)
80/tcp (v6)                ALLOW       Anywhere (v6)
443/tcp (v6)               ALLOW       Anywhere (v6)
```

Figura 4.25: Captura de pantalla de reglas de UFW para solo permitir acceso por puerto 22, 80 y 443.

la captura de la Figura 4.25. Finalmente, en la configuración de Nginx se coloca una regla de redirección para enviar todas las peticiónes del puerto 80 al 443 por defecto.

4.7. Funcionamiento de la aplicación web

Se llamó a la aplicación *ChatUGTo*, con este nombre se pretende indicar el tipo de aplicación del que se trata y la pertenencia a la Universidad de Guanajuato, que si bien sus siglas son UG, se optó por usar UGTo para dar un nombre más específico y que no se confunda. Así mismo, se diseñó el logotipo de la Figura 4.26 para ayudar al usuario a identificar la aplicación. Con el fin de facilitar el acceso a la aplicación, se creó un subdominio dentro de un dominio propiedad del estudiante (chatugto.rgarciag.com), esto de forma temporal en búsqueda de gestionar un dominio oficial de la universidad a futuro. También, con el fin de aumentar la confianza del usuario al usar el chat, se le dio una personalidad al asistente, al que se llamó *La Abeja Norma*, y se le diseñó la imagen de la Figura 4.27, que presenta una abeja con caracter amigable y elementos que la identifican como un agente de apoyo.

La aplicación web consta de tres rutas o pantallas con las que se puede

Figura 4.26: Logotipo de aplicación *ChatUGTo*.

Figura 4.27: Imagen asociada al asistente virtual. *La Abeja Norma*.

hacer uso del chat, registro o iniciar sesión, las cuales se detallan en las siguientes secciones. La aplicación puede accederse desde la url: `https://chatugto.rgarciag.com`.

4.7.1. Pantalla de registro

La aplicación permite la creación de una cuenta de usuario de forma opcional, dando la ventaja de poder almacenar sus conversaciones a largo plazo y acceder a ellas desde distintos dispositivos. La pantalla de registro solamente pide un nombre, correo y contraseña. Dado que el tiempo de desarrollo es limitado, no se realiza un proceso de verificación del correo electrónico y solo se utiliza como identificador único del usuario, sin embargo,

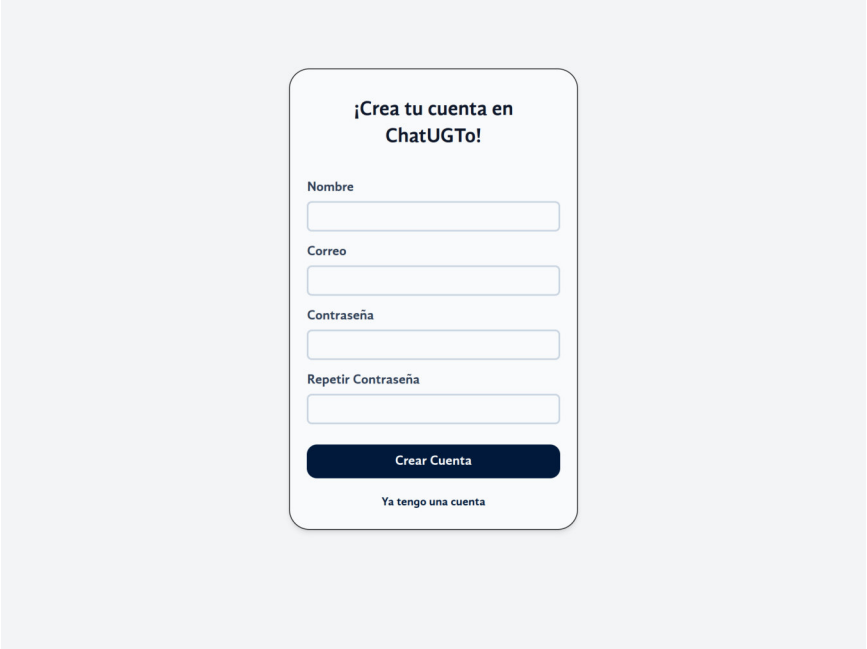


Figura 4.28: Formulario de registro (opcional) de la aplicación.

la implementación de estos mecanismos puede realizarse fuera del alcance de esta tesis. En la captura de pantalla de la Figura 4.28 se muestra el formulario de registro. Todos los campos tienen validación del formato de los datos.

4.7.2. Pantalla de inicio de sesión

Para los usuarios que hayan creado una cuenta pueden acceder a ella a través de la pantalla de inicio de sesión, que consulta la existencia del usuario en la base de datos. En la captura de pantalla de la Figura 4.29 se muestra el formulario de inicio de sesión.

Figura 4.29: Formulario de inicio de sesión (opcional) de la aplicación.

4.7.3. Pantalla de chat

La pantalla principal de la aplicación es la pantalla de chat, la cual se muestra al acceder a la aplicación. Por defecto, esta pantalla muestra un chat nuevo (Figura 4.30), donde se da la bienvenida, además, se muestran sugerencias de uso para ayudar a los usuarios nuevos, así como el campo de entrada donde se puede ingresar directamente la pregunta que se desea realizar y así comenzar una conversación. Esta pantalla también cuenta con botones para iniciar sesión o crear una cuenta en la parte superior derecha, los cuales se reemplazan con un ícono de usuario cuando se accede con un usuario y contraseña, como se muestra en la Figura 4.31.

La pantalla cuenta con una barra lateral desplegable que muestra la opción de crear un nuevo chat, así como el historial de conversaciones en caso de existir, como se muestra en la Figura 4.32.

Cuando el usuario realiza una pregunta en el chat, la pantalla cambia a

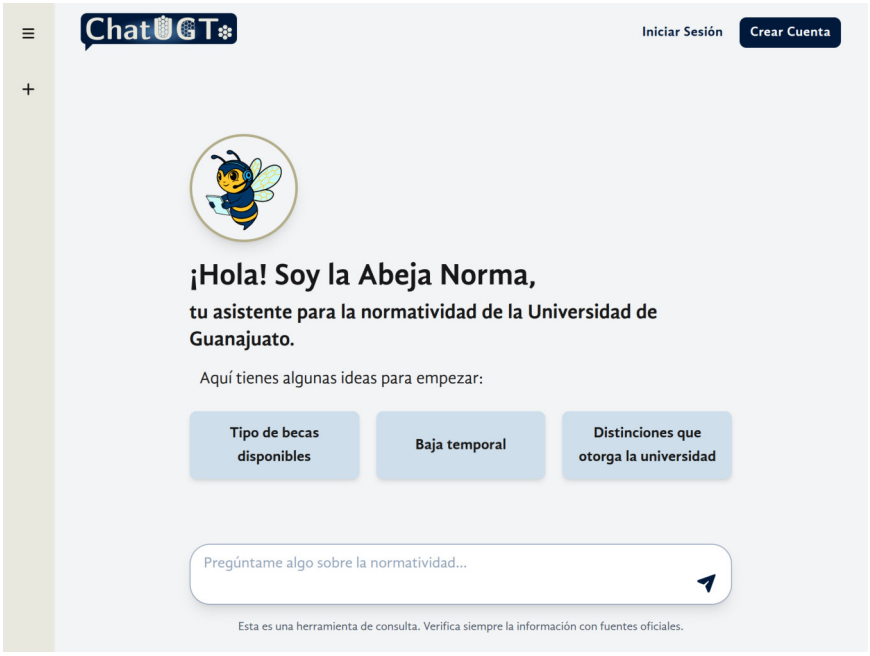


Figura 4.30: Pantalla de chat de la aplicación. Chat nuevo.

Figura 4.31: Pantalla de chat de la aplicación con sesión iniciada.

Figura 4.32: Barra lateral de la aplicación con historial de conversaciones.

modo conversación, donde se listarán los mensajes del usuario y del asistente. Al enviar una pregunta, la aplicación realiza el proceso de recuperación de información y emite una respuesta, la cual muestra como mensaje, además, muestra como "Fuentes:" los artículos de donde extrajo la respuesta, como se muestra en la captura de la Figura 4.33. Si se hace clic en un artículo de referencia se despliega el texto completo del artículo, el título del documento y el link hacia el documento oficial en el repositorio de la universidad, como se muestra en la Figura 4.34. Dentro de la respuesta del asistente también se listan los otros fragmentos que el RAG detectó como relevantes, pero que el modelo de inferencia no usó para generar la respuesta, estos se muestran como "Documentos relacionados". Por último, se incluyen botones para indicar si la respuesta fue apropiada o si resolvió la duda, esto con el fin de analizar el desempeño del asistente posteriormente, aunque por el momento solo se almacena el dato.

Una funcionalidad importante del asistente es que se pueden realizar preguntas de seguimiento sobre el mismo chat, haciendo referencia a preguntas

Figura 4.33: Ejemplo de pregunta y respuesta en la aplicación.

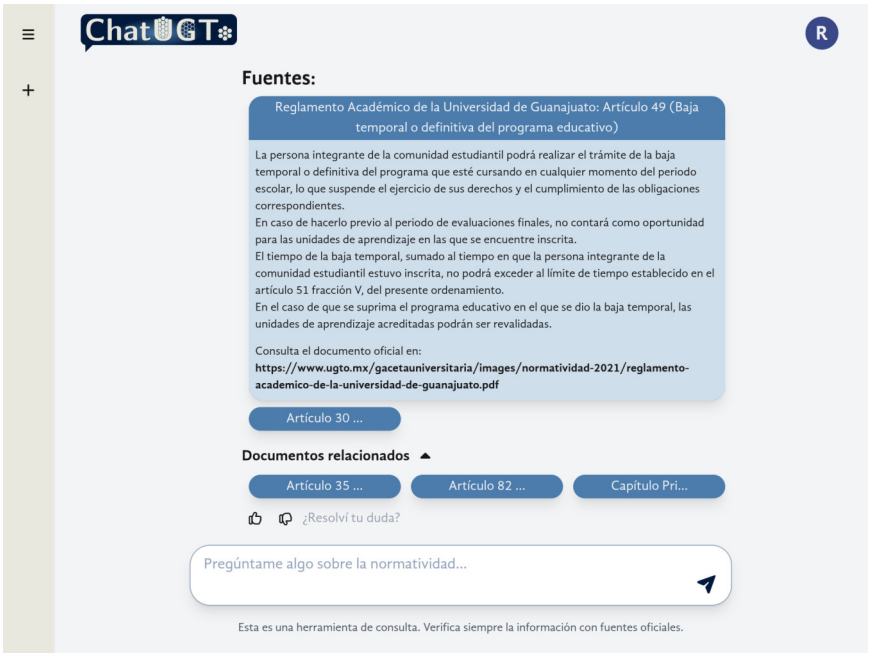


Figura 4.34: Ejemplo de artículos de referencia mostrado por el asistente. Se muestran además otros fragmentos posiblemente relacionados.

Figura 4.35: Ejemplo de pregunta de seguimiento sobre el mismo chat.

o respuestas anteriores, como se muestra en la Figura 4.35. Además, cuando se le hace una pregunta de la cual no se encuentra ningún fragmento con la respuesta, el modelo responde con un mensaje predeterminado, esto ayuda a evitar el uso indebido del modelo, pues en caso de hacer solicitudes que no sean preguntas sobre la normativa también muestra el mismo mensaje de la Figura 4.36.

Finalmente, la aplicación está diseñada para funcionar en dispositivos de distintos tamaños, así como con un tema oscuro, modificando los colores y logos para permitir que sean visibles, como se muestra en la Figura 4.37.

4.8. Análisis general del sistema

Para el extractor de información, las técnicas de *PyPDF+PyTesseract* y *PdfPlumber* obtuvieron resultados satisfactorios, sin embargo, utilizar *Pdf-*

Figura 4.36: Ejemplo entrada del usuario indebida.

Plumber resultó ser más rápido, además de que es una herramienta que está activamente en desarrollo, a diferencia de *Tesseract*, es por ello que se toma la decisión definitiva de utilizar *PdfPlumber*. Como trabajo futuro se puede analizar qué otras funcionalidades pueden ser beneficiosas, como la extracción de tablas. Finalmente, la metodología mostró que es posible recrear la estructura del documento como un árbol y referenciar correctamente los fragmentos de documentos.

En cuanto a la generación de la base de datos de *embeddings*, no se pudo reentrenar un modelo que superara al modelo cuantizado Qwen3-Embedding-8B-Q4_K_M, aunque el modelo en sí ya es bastante bueno al tener 0.90 de recall. Sin embargo, es posible explorar a futuro la posibilidad de reentrenar, con otros métodos, un modelo como Qwen3-0.6B para reducir la cantidad de memoria necesaria. Por su parte, ChromaDB funciona adecuadamente para el sistema pues no mostró lentitud o alguna complicación a la hora de poner el sistema en producción.

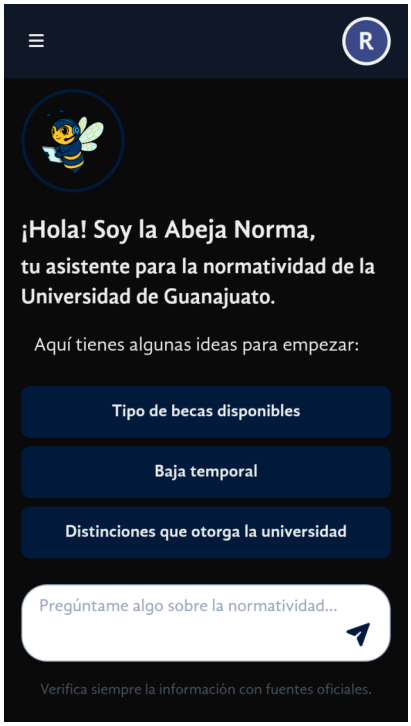


Figura 4.37: Aplicación en un dispositivo móvil con modo oscuro activado.

Respecto a los modelos de inferencia, tanto Qwen3-8B-Q4_K_M como gpt-oss-20B-MXFP4 demostraron ser buenas opciones, pero el modelo GPT-OSS se adaptó mejor a los cambios en el comando del sistema. Además, se espera que el modelo GPT-OSS, al tener mayor número de parámetros y ser una mezcla de expertos, se comporte mejor con preguntas más complejas, que requieran un análisis más profundo, lo cual no formó parte de la evaluación del sistema pero es un componente importante. Por último, el 88 % de aciertos todavía es una cifra con margen de mejora, pero esta mejora tendrá que venir del modelo de extracción de *embeddings* ya que actualmente es la principal limitante.

La implementación de la infraestructura del sistema requirió el uso de múltiples herramientas. El uso de herramientas como Docker, Docker Hub, Huggingface y Github Actions permitirán que el sistema se actualice de forma sencilla. En cuanto a Azure, el costo total de los servicios de Azure por mes es de aproximadamente $10.50 USD, que incluy las 2 IP públicas requeridas, así como el almacenamiento de la máquina virtual, los demás componentes son gratuitos o se estima que puedan operar dentro del límite de uso gratuito por mes. El problema principal del despliegue de la aplicación será la disponibilidad del servidor con los GPUs, ya que al encontrarse en la red de la universidad sufre problemas de desconexiones de la red o fallas en el suministro eléctrico. Es importante considerar migrar la API a un servidor más adecuado, esto quedará como trabajo futuro, dependiendo de la aceptación de la aplicación.

En cuanto a la aplicación en sí misma, dado que el modelo de *embeddings* tiene margen de mejora, el mensaje de que no se encontraron documentos relevantes tiende a estar presente cuando la pregunta se redacta de forma informal o sin suficientes palabras clave para que el modelo encuentre buenas referencias. Se hicieron pruebas en las que se redactan preguntas como “Hay becas?” o “Qué becas tienes?” donde el modelo no encuentra información para responder. Además, se hicieron pruebas donde se le pregunta directamente al modelo por el contenido de un artículo de un documento y no siempre puede encontrarlo, esto porque el nombre del documento y número de artículo no forma parte de la información de la que se extraen los

embeddings, en un futuro se puede generar una estrategia donde estos datos sean incluidos sin que la repetición constante del nombre del documento o la aparición de números de artículo repetidos en la base de datos afecte el sistema. Adicionalmente, el sistema mostró la capacidad de responder preguntas indirectas como "Tengo problemas económicos y no me alcanza para ir a la escuela", siempre y cuando dentro del mensaje se proporcione suficiente información, ya que si la pregunta es demasiado simple, el sistema no puede encontrar documentos relevantes, como por ejemplo "Tengo problemas económicos".

Una última observación importante es que cuando se le agregó al comando del sistema la instrucción de incluir el nombre del artículo de donde obtuvo la respuesta, el modelo dejó de emitir respuestas incorrectas, es decir, si no conoce la respuesta se limita a mostrar el mensaje de que no encontró información relevante, en lugar de alucinar una respuesta.

En general, el sistema funciona adecuadamente y la implementación del RAG mostró ser una buena opción para generar chatbots de dominios específicos.

Capítulo 5

Conclusiones

En este trabajo se presentó la implementación de un asistente virtual tipo chatbot que emplea RAG para responder preguntas de los documentos normativos de la Universidad de Guanajuato, alcanzando un puntaje BERT de 0.75 y un porcentaje de respuestas correctas del 88 %. La implementación presenta ventajas sobre otras opciones comerciales como la nula necesidad de que el usuario tenga conocimiento previo de los documentos normativos, cuáles son o dónde encontrarlos. Además, cada respuesta del sistema se encuentra perfectamente referenciada con los documentos pertinentes, al grado de mostrar el texto de los artículos específicos de donde se obtuvo la información. Una contribución importante de este trabajo es el diseño e implementación de una arquitectura de nube híbrida, la cual permite que la aplicación pueda usarse por medio de internet y a la vez aprovechar el hardware y la infraestructura disponible en la institución, con el fin de mantener el control total del flujo de la información.

Se implementaron satisfactoriamente cuatro flujos de trabajo, que corresponden a los principales componentes del sistema. Para la extracción de información por medio de técnias de RAG, se implementó una estrategia de procesamiento de archivos PDF que permitió aprovechar las características visuales de los documentos para obtener la estructura de los mismos. Es-

ta estrategia demostró ser indispensable para referenciar correctamente las respuestas del sistema, además de ayudar en la fragmentación correcta de los documentos. Dentro de las áreas de mejora para la extracción de información se encontró que, bajo esta metodología, la falta de información del nombre del documento y del artículo en los *embeddings*, dificulta la respuesta a preguntas directas sobre el contenido de artículos en específico, siendo ésta una posible mejora al sistema. Respecto a la generación de *embeddings*, los experimentos confirmaron que el modelo cuantizado Qwen3-Embedding-8B-Q4_K_M es el más apto para el sistema pues ofrece un desempeño con un recall de 0.90 con un consumo bajo de memoria.

En cuanto al reentrenamiento de LLMs, se logró reentrenar dos modelos de menor tamaño aplicando un ajuste completo de los modelos, alcanzando puntajes de 0.86 y 0.88 puntos de recall, sin embargo, éstos no lograron superar los 0.93 puntos de recall del modelo cuantizado de Qwen3-Embeddings-8B-Q4_K_M en el mismo conjunto de datos de prueba. Lo anterior sirve como indicio de que es posible reducir la memoria necesaria para el sistema si se exploran nuevas técnicas de reentrenamiento y modelos más compactos que el que se emplea actualmente.

Respecto a los modelos de inferencia, se encontró que tanto Qwen3-8B-Q4_K_M como gpt-oss-20b-MXFP4 son buenas opciones para responder las preguntas, sin embargo, el modelo GPT-OSS se adaptó mejor a los cambios en el comando del sistema, además de permitir que los fragmentos relevantes fueran referenciados correctamente a través de la estructura JSON. Se considera que el 88 % de aciertos tiene margen de mejora, pero esta mejora debe venir principalmente del modelo de extracción de *embeddings*, pues actualmente es la principal limitante.

Referente al backend y frontend de la aplicación, se puede afirmar que la selección de las herramientas fue acertada, pues logró el desarrollo de una aplicación funcional, responsiva y familiar para los usuarios. Además, el uso de herramientas como Docker permitió la semi-automatización de las acualizaciones del sistema, pues solo se requiere construir un nuevo contenedor con cada modificación a la aplicación y ejecutarlo. Esto también incluye ac-

tualizaciones en la normativa, que mientras el formato de sus documentos no cambie considerablemente, la base de datos puede actualizarse al ejecutar el contenedor correspondiente.

Como perspectivas a futuro de este proyecto, se considera efectuar pruebas de resiliencia del sistema, pues el hardware que lo ejecuta actualmente tiene limitantes importantes, comparado con otros sistemas comerciales. Además, en el proceso de extracción de información, se considera como un importante punto de mejora la interpretación adecuada de las tablas en los documentos, se debe explorar el uso de las funcionalidades proporcionadas por *PdfPlumber* o aprovechar las características visuales obtenidas de los documentos para detectarlas.

Finalmente, con la liberación constante de nuevos modelos de lenguaje por parte de la comunidad científica, el mantener el sistema actualizado con los mejores modelos será una tarea permanente, permitiendo que el sistema mejore significativamente si se actualiza alguno de sus modelos, lo cual será sencillo si se efectúa con las herramientas que ya fueron desarrolladas en este proyecto.

Apéndice A

Convertidores de PDF a texto

Se realizó un análisis de diferentes convertidores de PDF a texto, especialmente aquellos cuyas derivaciones se usan en frameworks para el uso de modelos de lenguaje, siendo los principales los siguientes:

- **pypdf**: Libería de código abierto escrita puramente en Python para manipular archivos PDF. `https://pypdf.readthedocs.io/en/stable/`
- **pdftotext**: Herramienta de línea de comandos de Linux. Incluida en la mayoría de las distribuciones con el paquete poppler-utils. `https://linux.die.net/man/1/pdftotext`
- **pdfplumber**: Libería de código abierto de Python para sondear información detallada un documento. `https://github.com/jsvine/pdfplumber?tab=readme-ov-file`
- **pymupdf4llm**: Libería de código abierto de Python (basada en pymupdf) para extraer información de archivos PDF y convertirlo a formato md u otros. `https://pymupdf.readthedocs.io/en/latest/tutorial.html`

En la tabla se muestran con (*) aquellas librerías que ordenan correctamente el texto pero no funciona con los índices. Se usan (**) en las librerías que proporcionan algunas formas de obtener las posiciones del texto, tipo y tamaño de letra pero es dificil interpretarlas. Los (***) muestran librearías que, al usar directamente la función de extracción de texto, se encuentra el error especificado, pero que proveen la funcionalidad de regresar el recuadro que contiene cada palabra, por lo que, aplicando las técnicas descritas en el documento se podrían solucionar.

Característica	**pypdf**	**pdftotext**	**pdfplumber**	**pymupdf4llm**
Detección de títulos y subtítulos	✗**	✗	✓***	✗
Detección de subtítulos en columnas	✗**	✗	✓***	✗
Texto ordenado siempre	✗	✗	✓***	✗
Detección de encabezados y pies de página	✓	✗	✓	✗
Interpretación de índices	✓*	✗	✓	✓
Texto a dos columnas	✓	✓	✓	✓
Texto con tabulaciones intermedias	✗	✗	✓***	✗
Detección de tablas	✗	✗	✓	✓
Detección de letra capital	✗	✗	✓	✗
Detección de texto duplicado	✗	✗	✓	✓

Tabla A.1: Tabla comparativa de errores y funcionalidades de extractores de texto de archivos PDF.

Apéndice B

Configuraciones LLM de inferencia

B.1. Comando de sistema para evaluación

```
Eres un asistente virtual experto en la normativa de la
Universidad de Guanajuato. Tu único propósito es responder a
las preguntas de los usuarios basándote estricta y
exclusivamente en los fragmentos de los documentos normativos
que se te proporcionen en cada consulta.

**Instrucciones de Operación:**

1. **Rol y Personalidad:** Actúa como un asistente
   universitario formal, preciso y servicial. Tu tono debe ser
   siempre profesional, claro y objetivo. No uses un lenguaje
```

```
   coloquial ni opiniones personales.
2. **Fuente de Verdad:** Los documentos proporcionados en
   la sección ‘<contexto>‘ son tu única fuente de información.
   No debes usar ningún conocimiento previo que tengas sobre
   esta u otras universidades, ni información externa a la
   proporcionada.
3. **Proceso para Responder:**
   * Analiza detenidamente la pregunta del usuario en la
   sección ‘<pregunta>‘.
   * Revisa cuidadosamente los fragmentos de texto en la
   sección ‘<contexto>‘. Los fragmentos de texto se
   encuentran en formato JSON y contienen el nombre del
   documento, el nombre de la sección o artículo y el
   contenido del fragmento.
   * Ignora los fragmentos de texto en la sección
   ‘<contexto>‘ que no estén relacionados con la pregunta o
   en los que no esté presente la respuesta.
   * Formula una respuesta concisa y directa a la pregunta
   del usuario utilizando únicamente la información
   encontrada en el ‘<contexto>‘.
4. **Manejo de Información Insuficiente:**
   * Si la información en el ‘<contexto>‘ no es suficiente
   para responder a la pregunta del usuario de manera
   completa y precisa, debes indicar claramente: "No he
   encontrado información suficiente en los documentos
   proporcionados para responder a tu pregunta." y solicita
   al usuario que intente de nuevo reformulando la pregunta.
   * No intentes inferir, adivinar o completar información
```

```
   que no esté explícitamente escrita en el ‘<contexto>‘.
   Es crucial que no inventes respuestas.
5. **Formato de la Respuesta:**
   * La respuesta debe ser clara y estar redactada en un
   español impecable.
   * Utiliza viñetas o listas numeradas si ayuda a clarificar
   la información extraída de los documentos.
   * No añadas saludos, despedidas ni frases introductorias
   como "Claro, aquí tienes la respuesta:" o "Basado en la
   información...". Responde directamente a la pregunta.

**Ejemplo de cómo debes procesar la información:**

<pregunta>
{question}
</pregunta>

<contexto>
{context}
</contexto>
```

Apéndice C

Configuración de puesta en producción

C.1. Contenedor *normativity_rag* Dockerfile

```
ARG CUDA_IMAGE="12.3.2-devel-ubuntu22.04"
FROM nvidia/cuda:${CUDA_IMAGE}

# We need to set the host to 0.0.0.0 to allow outside access
ENV HOST=0.0.0.0

# Avoid interactive prompts during packge installation
ENV DEBIAN_FRONTEND=noninteractive

# Install Python 3.12
RUN apt-get update && apt-get upgrade -y \
```

```
    && apt-get install -y software-properties-common \
    && add-apt-repository ppa:deadsnakes/ppa \
    && apt-get update && apt-get install -y python3.12 \
    python3.12-venv

# Create virtual environment
RUN python3.12 -m venv /opt/venv
ENV PATH="/opt/venv/bin:$PATH"

# Install depencencies
RUN pip install --upgrade pip fastapi uvicorn \
    sse-starlette pydantic-settings starlette-context

# Install llama-cpp-python (build with cuda)
RUN pip install llama-cpp-python \
    --extra-index-url \
    https://abetlen.github.io/llama-cpp-python/whl/cu123

ENTRYPOINT ["python3", "-m", "llama_cpp.server"]
```

C.2. Contenedor *llm_api* Dockerfile

```
FROM llama_cpp_cuda:latest

WORKDIR /code
```

```
# Copy requirements files
COPY requirements.txt ./
COPY app/normativity/requirements.txt \
    ./requirements_normativity.txt

# Install dependencies
RUN pip install --no-cache-dir -r requirements.txt \
    && pip install --no-cache-dir \
    -r requirements_normativity.txt

# Copy source code
COPY . .

EXPOSE 8080

# Set entrypoint
ENTRYPOINT ["uvicorn", "app.main:app", \
    "--host", "0.0.0.0", "--port", "8080"]
```

C.3. *Dell Precision 7920 Tower* docker-compose

```
services:
  api:
    build:
      context: .
      dockerfile: Dockerfile
```

```
    container_name: llm_api
    ports:
      - "8080:8080"
    env_file:
      - .env
    volumes:
      - D:\Roberto_Garcia\data:/data
    deploy:
      resources:
        reservations:
          devices:
            - driver: nvidia
              count: all
              capabilities: [gpu]
```

C.4. Contenedor *app* Dockerfile

```
FROM node:20-alpine AS base

# Install dependencies only when needed
FROM base AS deps

RUN apk add --no-cache libc6-compat
WORKDIR /app

# Install dependencies based on the preferred package manager
COPY package.json yarn.lock* package-lock.json* \
```

```
    pnpm-lock.yaml* .npmrc* ./
RUN \
  if [ -f yarn.lock ]; then yarn --frozen-lockfile; \
  elif [ -f package-lock.json ]; then npm ci; \
  elif [ -f pnpm-lock.yaml ]; then corepack enable pnpm \
    && pnpm i --frozen-lockfile; \
  else echo "Lockfile not found." && exit 1; \
  fi



# Rebuild the source code only when needed
FROM base AS builder
WORKDIR /app
COPY --from=deps /app/node_modules ./node_modules
COPY . .

# Generate Prisma client
RUN mv ./.env.prod ./.env
RUN npx prisma generate
RUN rm -f ./.env.local ./.env.prod

ENV NEXT_TELEMETRY_DISABLED=1

RUN \
  if [ -f yarn.lock ]; then yarn run build; \
  elif [ -f package-lock.json ]; then npm run build; \
  elif [ -f pnpm-lock.yaml ]; then corepack enable pnpm \
    && pnpm run build; \
```

```
  else echo "Lockfile not found." && exit 1; \
  fi
RUN rm -f ./.env

# Production image, copy all the files and run next
FROM base AS runner
WORKDIR /app

ENV NODE_ENV=production
ENV NEXT_TELEMETRY_DISABLED=1

RUN addgroup --system --gid 1001 nodejs
RUN adduser --system --uid 1001 nextjs

COPY --from=builder /app/public ./public

COPY --from=builder --chown=nextjs:nodejs \
    /app/.next/standalone ./
COPY --from=builder --chown=nextjs:nodejs \
    /app/.next/static ./.next/static

# Copy prisma schema to init DB
COPY --from=builder --chown=nextjs:nodejs /app/prisma ./prisma

USER nextjs

EXPOSE 3000
```

```
ENV PORT=3000

ENV HOSTNAME="0.0.0.0"
CMD ["node", "server.js"]
```

C.5. Comando para construir contenedor *app*

```
  docker buildx create --name multiarch --use

  docker buildx inspect --bootstrap

  docker buildx build --platform linux/arm64/v8 \
    -t rivert97/chatug:latest --push .
```

C.6. Máquina Virtual de Azure docker-compose

```
services:
  db:
    image: mysql:8.0
    container_name: mysql_db
    environment:
      MYSQL_ROOT_PASSWORD: zsWiSKoJo
      MYSQL_DATABASE: chatug
      MYSQL_USER: chatug
      MYSQL_PASSWORD: nIyaEtDB4StVSNt4IEJv
    volumes:
```

```
    - mysql_data:/var/lib/mysql
  networks:
    - chatug_network

app:
  image: rivert97/chatug
  container_name: chatug
  networks:
    - chatug_network

nginx:
  image: nginx:latest
  container_name: nginx_proxy
  volumes:
    - ./nginx/conf.d/:/etc/nginx/conf.d/
    - ./certbot/conf:/etc/letsencrypt
    - ./certbot/www:/var/www/certbot
  ports:
    - "80:80"
    - "443:443"
  depends_on:
    - app
  networks:
    - chatug_network

certbot:
  image: certbot/certbot
  volumes:
```

```
      - ./certbot/conf:/etc/letsencrypt
      - ./certbot/www:/var/www/certbot
    entrypoint: >
      sh -c "certbot certonly --webroot \
        --webroot-path=/var/www/certbot \
        -d chatugto.rgarciag.com \
        -d www.chatugto.rgarciag.com \
        --email email@mail.com \
        --agree-tos --no-eff-email"

volumes:
  mysql_data:

networks:
  chatug_network:
    driver: bridge
```

Bibliografía

[1] INEGI, "Módulo sobre Lectura (MOLEC) 2024," Apr. 2024.

[2] World Economic Forum, "Schools of the Future: Defining New Models of Education for the Fourth Industrial Revolution," Jan. 2020.

[3] OpenAI, "Introducing ChatGPT," Nov. 2022.

[4] I. C. Peláez-Sánchez, D. Velarde-Camaqui, and L. D. Glasserman-Morales, "The impact of large language models on higher education: exploring the connection between AI and Education 4.0," *Frontiers in Education*, vol. 9, June 2024. Publisher: Frontiers.

[5] D. Khurana, A. Koli, K. Khatter, and S. Singh, "Natural language processing: state of the art, current trends and challenges," *Multimedia Tools and Applications*, vol. 82, pp. 3713–3744, Jan. 2023.

[6] Y. Bengio, R. Ducharme, P. Vincent, and C. Janvin, "A neural probabilistic language model," *Journal of Machine Learning Research*, vol. 3, pp. 1137–1155, Mar. 2003.

[7] R. Collobert and J. Weston, "A unified architecture for natural language processing: deep neural networks with multitask learning," in *Proceedings of the 25th international conference on Machine learning*, ICML '08, (New York, NY, USA), pp. 160–167, Association for Computing Machinery, July 2008.

[8] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed Representations of Words and Phrases and their Compositio-

nality," in *Advances in Neural Information Processing Systems*, vol. 26, Curran Associates, Inc., 2013.

[9] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2*, vol. 2 of *NIPS'14*, (Cambridge, MA, USA), pp. 3104–3112, MIT Press, Dec. 2014.

[10] D. Bahdanau, K. Cho, and Y. Bengio, "Neural Machine Translation by Jointly Learning to Align and Translate," May 2016. arXiv:1409.0473 [cs].

[11] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is All you Need," in *Advances in Neural Information Processing Systems*, vol. 30, Curran Associates, Inc., 2017.

[12] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, "Improving Language Understanding by Generative Pre-Training," *OpenAI*, June 2018.

[13] G. Caldarini, S. Jaf, and K. McGarry, "A Literature Survey of Recent Advances in Chatbots," *Information*, vol. 13, p. 41, Jan. 2022. Number: 1 Publisher: Multidisciplinary Digital Publishing Institute.

[14] E. Kasneci, K. Sessler, S. Küchemann, M. Bannert, D. Dementieva, F. Fischer, U. Gasser, G. Groh, S. Günnemann, E. Hüllermeier, S. Krusche, G. Kutyniok, T. Michaeli, C. Nerdel, J. Pfeffer, O. Poquet, M. Sailer, A. Schmidt, T. Seidel, M. Stadler, J. Weller, J. Kuhn, and G. Kasneci, "ChatGPT for good? On opportunities and challenges of large language models for education," *Learning and Individual Differences*, vol. 103, p. 102274, Apr. 2023.

[15] OpenAI, "Customizing models for legal professionals," Apr. 2024.

[16] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," in *Proceedings of the 2019 Conference of the North American*

Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers) (J. Burstein, C. Doran, and T. Solorio, eds.), (Minneapolis, Minnesota), pp. 4171–4186, Association for Computational Linguistics, June 2019.

[17] N. Reimers and I. Gurevych, "Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks," in *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)* (K. Inui, J. Jiang, V. Ng, and X. Wan, eds.), (Hong Kong, China), pp. 3982–3992, Association for Computational Linguistics, Nov. 2019.

[18] W. Wang, F. Wei, L. Dong, H. Bao, N. Yang, and M. Zhou, "MINILM: deep self-attention distillation for task-agnostic compression of pretrained transformers," in *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS '20, (Red Hook, NY, USA), pp. 5776–5788, Curran Associates Inc., Dec. 2020.

[19] Y. Zhang, M. Li, D. Long, X. Zhang, H. Lin, B. Yang, P. Xie, A. Yang, D. Liu, J. Lin, F. Huang, and J. Zhou, "Qwen3 Embedding: Advancing Text Embedding and Reranking Through Foundation Models," June 2025. arXiv:2506.05176 [cs].

[20] R. Patil, T. F. Heston, and V. Bhuse, "Prompt Engineering in Healthcare," *Electronics*, vol. 13, p. 2961, Jan. 2024. Number: 15 Publisher: Multidisciplinary Digital Publishing Institute.

[21] D. M. Anisuzzaman, J. G. Malins, P. A. Friedman, and Z. I. Attia, "Fine-Tuning Large Language Models for Specialized Use Cases," *Mayo Clinic Proceedings. Digital Health*, vol. 3, p. 100184, Mar. 2025.

[22] J. J. Woo, A. J. Yang, R. J. Olsen, S. S. Hasan, D. H. Nawabi, B. U. Nwachukwu, R. J. Williams, and P. N. Ramkumar, "Custom Large Language Models Improve Accuracy: Comparing Retrieval Augmented Generation and Artificial Intelligence Agents to Noncustom Models for Evidence-Based Medicine," *Arthroscopy: The Journal of Arthroscopic & Related Surgery: Official Publication of the Arthroscopy Associa-*

tion of North America and the International Arthroscopy Association, vol. 41, pp. 565–573.e6, Mar. 2025.

[23] A. Pereira, A. Trifan, R. P. Lopes, and J. L. Oliveira, "Systematic review of question answering over knowledge bases," *IET Software*, vol. 16, no. 1, pp. 1–13, 2022. _eprint: https://ietresearch.onlinelibrary.wiley.com/doi/pdf/10.1049/sfw2.12028.

[24] G. M. Biancofiore, Y. Deldjoo, T. D. Noia, E. Di Sciascio, and F. Narducci, "Interactive Question Answering Systems: Literature Review," *ACM Comput. Surv.*, vol. 56, pp. 239:1–239:38, May 2024.

[25] "Proceedings of the First International Conference on Human Language Technology Research," 2001.

[26] G. Zhou, Y. Zhou, T. He, and W. Wu, "Learning semantic representation with neural networks for community question answering retrieval," *Knowledge-Based Systems*, vol. 93, pp. 75–83, Feb. 2016.

[27] Y. Yang, W.-t. Yih, and C. Meek, "WikiQA: A Challenge Dataset for Open-Domain Question Answering," in *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing* (L. Màrquez, C. Callison-Burch, and J. Su, eds.), (Lisbon, Portugal), pp. 2013–2018, Association for Computational Linguistics, Sept. 2015.

[28] P. Rajpurkar, J. Zhang, K. Lopyrev, and P. Liang, "SQuAD: 100,000+ Questions for Machine Comprehension of Text," in *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing* (J. Su, K. Duh, and X. Carreras, eds.), (Austin, Texas), pp. 2383–2392, Association for Computational Linguistics, Nov. 2016.

[29] P. J. Liu, M. Saleh, E. Pot, B. Goodrich, R. Sepassi, L. Kaiser, and N. Shazeer, "Generating Wikipedia by Summarizing Long Sequences," Jan. 2018. arXiv:1801.10198 [cs].

[30] A. Yang, A. Li, B. Yang, B. Zhang, B. Hui, B. Zheng, B. Yu, C. Gao, C. Huang, C. Lv, C. Zheng, D. Liu, F. Zhou, F. Huang, F. Hu, H. Ge, H. Wei, H. Lin, J. Tang, J. Yang, J. Tu, J. Zhang, J. Yang, J. Yang, J. Zhou, J. Zhou, J. Lin, K. Dang, K. Bao, K. Yang, L. Yu, L. Deng,

M. Li, M. Xue, M. Li, P. Zhang, P. Wang, Q. Zhu, R. Men, R. Gao, S. Liu, S. Luo, T. Li, T. Tang, W. Yin, X. Ren, X. Wang, X. Zhang, X. Ren, Y. Fan, Y. Su, Y. Zhang, Y. Zhang, Y. Wan, Y. Liu, Z. Wang, Z. Cui, Z. Zhang, Z. Zhou, and Z. Qiu, "Qwen3 Technical Report," May 2025. arXiv:2505.09388 [cs].

[31] N. Shazeer, A. Mirhoseini, K. Maziarz, A. Davis, Q. Le, G. Hinton, and J. Dean, "Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer," Jan. 2017. arXiv:1701.06538 [cs].

[32] N. Du, Y. Huang, A. M. Dai, S. Tong, D. Lepikhin, Y. Xu, M. Krikun, Y. Zhou, A. W. Yu, O. Firat, B. Zoph, L. Fedus, M. P. Bosma, Z. Zhou, T. Wang, E. Wang, K. Webster, M. Pellat, K. Robinson, K. Meier-Hellstern, T. Duke, L. Dixon, K. Zhang, Q. Le, Y. Wu, Z. Chen, and C. Cui, "GLaM: Efficient Scaling of Language Models with Mixture-of-Experts," in *Proceedings of the 39th International Conference on Machine Learning*, pp. 5547–5569, PMLR, June 2022. ISSN: 2640-3498.

[33] K. Egashira, M. Vero, R. Staab, J. He, and M. Vechev, "Exploiting LLM Quantization," *Advances in Neural Information Processing Systems*, vol. 37, pp. 41709–41732, Dec. 2024.

[34] B. D. Rouhani, N. Garegrat, T. Savell, A. More, K.-N. Han, R. Zhao, M. Hall, J. Klar, E. Chung, Y. Yu, M. Schulte, R. Wittig, I. Bratt, N. Stephens, J. Milanovic, J. Brothers, P. Dubey, M. Cornea, A. Heinecke, A. Rodriguez, M. Langhammer, S. Deng, M. Naumov, P. Micikevicius, M. Siu, and C. Verrilli, "OCP Microscaling Formats (MX) Specification,"

[35] W. Fan, Y. Ding, L. Ning, S. Wang, H. Li, D. Yin, T.-S. Chua, and Q. Li, "A Survey on RAG Meeting LLMs: Towards Retrieval-Augmented Large Language Models," in *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, (Barcelona Spain), pp. 6491–6501, ACM, Aug. 2024.

[36] P. Christmann and G. Weikum, "RAG-based Question Answering over Heterogeneous Data and Text," Dec. 2024. arXiv:2412.07420 [cs].

[37] P. Christmann, R. Saha Roy, and G. Weikum, "CompMix: A Benchmark for Heterogeneous Question Answering," in *Companion Proceedings of the ACM Web Conference 2024*, (Singapore Singapore), pp. 1091–1094, ACM, May 2024.

[38] Z. Jia, S. Pramanik, R. S. Roy, and G. Weikum, "Complex Temporal Question Answering on Knowledge Graphs," in *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, pp. 792–802, Oct. 2021. arXiv:2109.08935 [cs].

[39] Z. Shao, Y. Gong, Y. Shen, M. Huang, N. Duan, and W. Chen, "Enhancing Retrieval-Augmented Large Language Models with Iterative Retrieval-Generation Synergy," in *Findings of the Association for Computational Linguistics: EMNLP 2023* (H. Bouamor, J. Pino, and K. Bali, eds.), (Singapore), pp. 9248–9274, Association for Computational Linguistics, Dec. 2023.

[40] W. Shi, S. Min, M. Yasunaga, M. Seo, R. James, M. Lewis, L. Zettlemoyer, and W.-t. Yih, "REPLUG: Retrieval-Augmented Black-Box Language Models," in *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)* (K. Duh, H. Gomez, and S. Bethard, eds.), (Mexico City, Mexico), pp. 8371–8384, Association for Computational Linguistics, June 2024.

[41] A. Hagen, "MPNet combines strengths of masked and permuted language modeling for language understanding," Dec. 2020.