Implementación de un Sistema RAG para el Curso de Introducción a la Programación de la Universidad de Matanzas

MSc. Yosbel Peñate Barceló

January 3, 2025

Contents

1 Introducción

1.1 Contexto y Motivación

El curso de Introducción a la Programación de la carrera de Ingeniería Informática en la Universidad de Matanzas tiene como objetivo fundamental capacitar a los estudiantes en la solución de problemas mediante el uso de computadoras. El plan de estudios se estructura en torno a tres temas principales. Inicialmente, en el Tema I, se aborda la Algoritmización, donde los estudiantes aprenden a resolver problemas utilizando algoritmos informales. Posteriormente, en el Tema II, se introducen las Estructuras de control, cubriendo conceptos clave como tipos de datos, variables, constantes, asignación, expresiones, así como las estructuras de control alternativas y repetitivas, culminando con el estudio de funciones. Finalmente, en el Tema III, se exploran los Arreglos, tanto unidimensionales como bidimensionales, y se analizan algoritmos básicos para su manipulación. El curso proporciona una base sólida en los conceptos fundamentales de la programación, preparando a los estudiantes para abordar problemas computacionales de manera metódica y estructurada.

Los estudiantes que se inician en el aprendizaje de programación, y en particular al abordar el contenido específico del curso, a menudo se enfrentan a dificultades que pueden obstaculizar su comprensión y progreso. Uno de los principales desafíos surge de la dificultad para encontrar información específica dentro de los materiales del curso. Aunque las conferencias y guías de estudio proporcionan una base teórica, a menudo carecen de la granularidad necesaria para responder a preguntas puntuales que los estudiantes puedan tener al trabajar en ejercicios prácticos. Los conceptos clave pueden estar dispersos en varios documentos, lo que dificulta la localización de información específica cuando se necesita una referencia rápida o una aclaración sobre un detalle concreto. Esta dificultad en encontrar información puntual puede llevar a los estudiantes a dedicar un tiempo excesivo a la búsqueda, en detrimento del tiempo que dedican a la práctica.

Además, los estudiantes principiantes a menudo luchan con la necesidad de respuestas contextualizadas que estén alineadas con el contenido del curso. Aunque los materiales didácticos explican los conceptos fundamentales como las estructuras de control o los arreglos, los estudiantes necesitan ejemplos prácticos y aclaraciones que demuestren cómo aplicar estas herramientas a los problemas específicos planteados en los ejercicios y talleres del curso. Las respuestas generales o teóricas pueden resultar insuficientes para comprender cómo implementar un algoritmo de búsqueda en un arreglo unidimensional o cómo anidar estructuras de control para resolver un problema específico del curso. La falta de ejemplos y explicaciones que se vinculen directamente con el contenido y el nivel del curso puede dificultar que los estudiantes apliquen los conocimientos adquiridos de manera efectiva.

1.2 Objetivos del Proyecto

Objetivo general del proyecto: mejorar el acceso a la información y el soporte de aprendizaje en el curso de Introducción a la Programación de la Universidad de Matanzas mediante la implementación de un sistema de Generación Aumentada por Recuperación (RAG).

Objetivos específicos:

- Recopilar y organizar exhaustivamente todos los materiales del curso en un corpus digital.
- Diseñar e implementar un sistema RAG eficiente que integre un motor de búsqueda semántico y un modelo de lenguaje, asegurando respuestas relevantes y contextualizadas.
- Adaptar las respuestas generadas al contexto específico del curso, proporcionando ejemplos prácticos y explicaciones alineadas con los temas tratados.
- Facilitar la búsqueda de información puntual, permitiendo a los estudiantes encontrar rápidamente lo que necesitan.
- Evaluar y mejorar continuamente el sistema a través del feedback de usuarios y métricas de rendimiento.

1.3 Alcance del Proyecto

El alcance de este proyecto se centra en el desarrollo e implementación de un sistema de Generación Aumentada por Recuperación (RAG), específicamente diseñado para apoyar a los estudiantes del curso de Introducción a la Programación de la carrera de Ingeniería Informática de la Universidad de Matanzas. Este sistema actuará como una herramienta de aprendizaje que facilitará el acceso a información relevante y contextualizada, ayudando a los estudiantes a comprender y aplicar los conceptos clave del curso de manera más efectiva.

Materiales del Curso como Base de Conocimiento: El sistema RAG utilizará como base de conocimiento los materiales del curso, incluyendo las conferencias en formato PDF, las guías de estudio, los enunciados de ejercicios y talleres. Estos documentos, que abarcan la teoría y la práctica del curso, serán procesados para extraer su contenido textual, limpiarlo y tokenizarlo, generando así los embeddings que permitirán una búsqueda semántica eficiente. Adicionalmente, los materiales procesados se almacenarán en una base de datos vectorial, facilitando la recuperación rápida y precisa de la información relevante en respuesta a las preguntas de los estudiantes.

Tipos de Preguntas que el Sistema RAG Debe Responder: El sistema RAG estará diseñado para responder una amplia gama de preguntas, abarcando desde conceptos teóricos hasta la sintaxis de código y la resolución de problemas específicos del curso. Esto incluye consultas sobre definiciones y explicaciones de conceptos clave, preguntas sobre el funcionamiento de los algoritmos y las estructuras de datos, así como preguntas sobre la sintaxis y el uso de palabras reservadas de Java. Además, el sistema deberá ofrecer ejemplos de código que ilustren cómo aplicar los conceptos aprendidos, y ayudar a los estudiantes a abordar los problemas planteados en los ejercicios y talleres del curso, proporcionando explicaciones contextualizadas y ejemplos prácticos. Sin embargo, es importante destacar que el sistema debe estar diseñado para guiar el aprendizaje y ayudar al entendimiento, no para resolver los problemas por los estudiantes, ni para responder preguntas que estén fuera del ámbito del curso.

Usuarios Objetivo: Los usuarios principales de este sistema RAG son los estudiantes del curso de Introducción a la Programación. Se considerará que estos estudiantes son

principiantes en el mundo de la programación, por lo que las respuestas proporcionadas por el sistema serán claras, sencillas y evitarán el uso de jerga técnica innecesaria. Además, el sistema será accesible a través de una interfaz web intuitiva y fácil de usar, que permitirá a los estudiantes formular sus preguntas en lenguaje natural sin necesidad de tener conocimientos técnicos avanzados. La interacción con el sistema estará diseñada para ser lo más sencilla y natural posible, facilitando su uso por parte de todos los estudiantes del curso.

Énfasis en la Usabilidad: La usabilidad del sistema RAG será una prioridad clave. Se desarrollará una interfaz de usuario simple y fácil de usar, que permitirá a los estudiantes formular sus preguntas y recibir respuestas de manera eficiente. Las respuestas generadas por el sistema serán claras, concisas y fáciles de entender para los estudiantes principiantes. Además, el sistema incluirá un mecanismo para que los usuarios puedan proporcionar feedback sobre la calidad de las respuestas recibidas, lo que facilitará la mejora continua del sistema y su adaptación a las necesidades específicas de los estudiantes del curso.

2 Diseño del Sistema RAG

2.1 Arquitectura General del Sistema

El sistema RAG, cuya arquitectura se muestra en la **Figura 1**, se organiza como un flujo de procesamiento de información que integra el componente de Retrieval con el componente de Generation buscando responder a las consultas del Usuario de manera informada y contextualizada la arquitectura se compone de siete elementos principales representados en el diagrama el Usuario que inicia el proceso el Componente de Entrada (Input) que recibe la consulta el Componente de Recuperación (Retrieval) encargado de buscar información relevante el Almacén Vectorial (Vector Store) que gestiona los embeddings el Componente de Generación (Generation) que prepara el prompt el modelo de lenguaje Gemini y el Componente de Salida (Output) que presenta la respuesta El Usuario es el actor principal que inicia el proceso ingresando una consulta específica el Componente de Entrada (Input) recibe esta consulta preparándola para su procesamiento posterior este componente aplica pasos de limpieza normalización y tokenización El Componente de Recuperación (Retrieval) es responsable de buscar en la fuente de conocimiento la información relevante para la consulta procesada utilizando para ello el Almacén Vectorial (Vector Store) este componente realiza una búsqueda vectorial obteniendo los fragmentos más relevantes el Almacén Vectorial (Vector Store) es una base de datos especializada que gestiona las representaciones vectoriales de la Documentos almacenados en la fuente de conocimiento realizando la indexación búsqueda y almacenamiento de los embeddings el Componente de Generación (Generation) actúa como puente entre el Componente de Recuperación (Retrieval) y el modelo de lenguaje Gemini combinando la consulta original con los fragmentos recuperados para construir un prompt para el modelo Gemini el modelo Gemini utiliza su conocimiento interno y el contexto del prompt para generar una respuesta coherente y relevante el Componente de Salida (Output) recibe esta respuesta formateándola y presentándola al Usuario el flujo de datos comienza con la consulta del Usuario que pasa por el Componente de Entrada (Input) luego al Componente de Recuperación (Retrieval) que consulta al Almacén Vectorial (Vector Store) este retorna fragmentos que son procesados por el Componente de Recuperación (Retrieval) y enviados

al Componente de Generación (Generation) que genera el prompt para el modelo Gemini que produce la respuesta que el Componente de Salida (Output) presenta al Usuario de esta forma el sistema RAG aprovecha las capacidades de los LLMs como el modelo Gemini para generar texto y la búsqueda semántica del Almacén Vectorial ofreciendo resultados más precisos y contextualizados la estructura modular permite que cada componente sea adaptado a las necesidades del dominio y a los objetivos específicos del sistema.

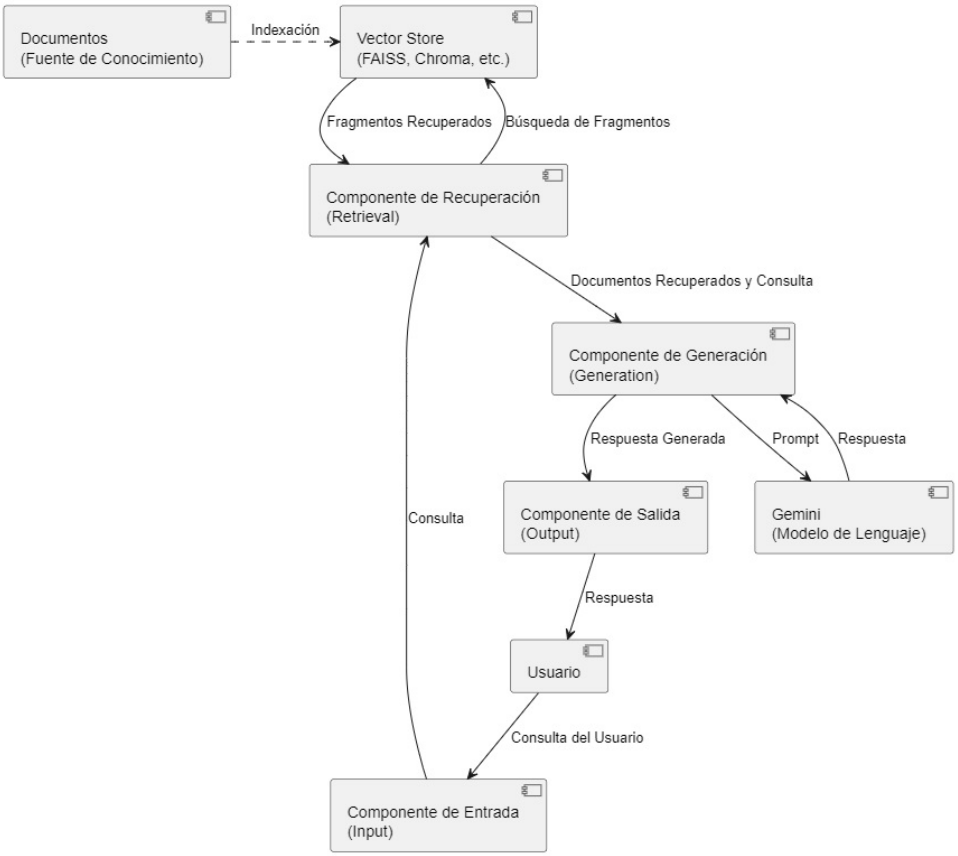

Figure 1: Diagrama de la arquitectura del sistema RAG **(Figura 1)**

Este diagrama de secuencia **Figura 2** ilustra el funcionamiento de un chatbot de inteligencia artificial basado en la técnica RAG (Retrieval-Augmented Generation) que opera a través de Telegram. El usuario inicia la interacción enviando un mensaje al bot, que a su vez lo reenvía a una API Gateway. Esta API Gateway dirige la solicitud a un Back-end Service, que se encarga de la lógica RAG. El Back-end Service primero consulta un Vector Store para obtener fragmentos relevantes de información según la pregunta del usuario. Luego, utiliza esta información contextual, junto con la pregunta original, para enviarla a un Language Model, el cual genera una respuesta. La respuesta es devuelta al Back-end Service, que a su vez la envía a la API Gateway, y esta última la transmite al bot de Telegram. Finalmente, el bot de Telegram entrega la respuesta generada al usuario. Este diagrama destaca la separación de responsabilidades entre los componentes: el bot como interfaz, el API Gateway como punto de entrada, el Back-end Service con la lógica RAG, el Vector Store para la recuperación de información, y el Language Model para la generación de respuestas. En resumen, el diagrama muestra el flujo completo de un sistema RAG, desde la interacción del usuario hasta la generación de una respuesta informada y contextual.

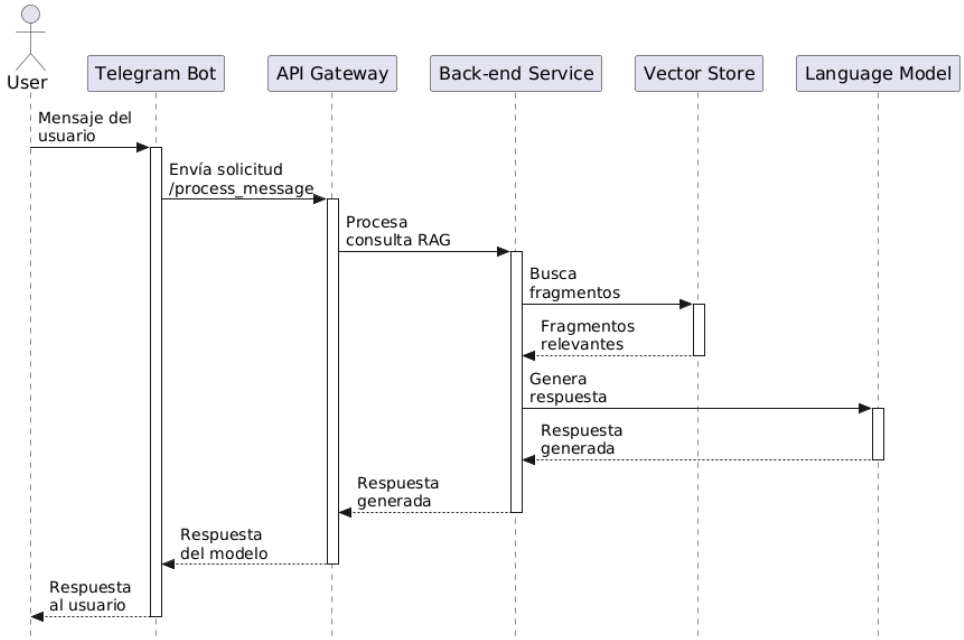

Figure 2: Diagrama de secuencia del bot **(Figura 2)**

2.2 Preprocesamiento de los Documentos

En la implementación del sistema RAG para el curso de Introducción a la Programación, el preprocesamiento de los documentos constituyó una etapa importante que implicó decisiones técnicas precisas. Para la extracción de texto desde los archivos PDF, se optó inicialmente por la biblioteca PyPDF2 por su sencillez. En esencia, PyPDF2 funciona analizando la estructura interna de un archivo PDF, localizando los objetos de texto y extrayendo su contenido en forma de cadenas. Esta biblioteca resulta efectiva con archivos PDF creados desde software de edición de texto, donde el texto es una capa editable dentro del documento. Sin embargo, su principal limitación reside en su dificultad para manejar PDFs con texto dentro de imágenes o con estructuras complejas, como tablas o columnas, a diferencia de bibliotecas como pdfplumber, que abordan estos desafíos con precisión, y Tesseract OCR, especializada en el reconocimiento óptico de caracteres para PDFs escaneados. Tras la extracción, el paso siguiente se centró en la limpieza y normalización. Este proceso inició con la eliminación de caracteres especiales sin valor semántico para el texto, como símbolos o caracteres no pertenecientes al alfabeto estándar. Después, se procedió a la conversión a minúsculas para asegurar uniformidad y evitar que el sistema tratase palabras escritas de manera diferente como términos distintos. Por último, para estandarizar las palabras y reducir las variaciones lingüísticas, se aplicó un proceso de lematización, que convierte las palabras a su forma base, logrando una representación consistente del vocabulario. La fragmentación del texto se realizó mediante una técnica específica que dividió el texto en bloques de tamaño fijo con solapamiento entre ellos. A diferencia de métodos que fragmentan el texto según la estructura del documento (párrafos, secciones) o mediante unidades semánticas complejas, el enfoque de fragmentación en bloques con tamaño fijo y solapamiento tiene la ventaja de asegurar que cada fragmento contenga una cantidad predecible de información y que no se pierda contexto relevante al final de un bloque y al inicio del siguiente. El solapamiento entre los bloques, en particular, resulta fundamental para evitar que información crucial entre

dos bloques sea omitida o que su contexto se pierda en el proceso; este solapamiento actúa como "ventana deslizante" que asegura que la información entre bloques siempre se considere, lo que difiere de la fragmentación por párrafos, con tamaño difícil de fijar, o la fragmentación por secciones, con fragmentos de longitud variable; sin considerar la estructura del texto, este enfoque asegura uniformidad y recuperación contextualizada, sin sesgos.

2.3 Modelo de Recuperación

Selección del modelo de embeddings: La selección del modelo de embeddings es crucial para el rendimiento del sistema de recuperación. En este proyecto, se considerará el modelo text-davinci-002, perteneciente a la familia GPT-3.5 de OpenAI, junto con embeddings de 1536 dimensiones [9]. **Justificación de la elección del modelo:** text-davinci-002 se destaca por su capacidad para procesar largas secuencias de texto, manteniendo una comprensión clara del contexto [9]. Esta característica es fundamental al trabajar con documentos extensos donde la información puede estar dispersa en varias secciones. El modelo puede rastrear el contexto de la consulta del usuario y recuperar o generar respuestas coherentes y relevantes dinámicamente [9]. La adopción de embeddings con una dimensionalidad de 1536 posibilita una representación más precisa de las intrincadas relaciones semánticas presentes en el texto, especialmente en aquellos documentos donde el significado se construye sobre sutiles distinciones en la articulación lingüística según [9]. Esto resulta particularmente útil para diferenciar términos similares pero contextualmente diferentes, lo que permite que el sistema recupere los fragmentos más relevantes [9]. **Generación de embeddings para los fragmentos de texto:** Una vez seleccionado el modelo, se procede a la generación de embeddings para cada fragmento de texto. Este proceso implica lo siguiente:

- Tokenización: El texto se divide en unidades más pequeñas, como palabras o frases, utilizando el modelo text-embedding-ada-002 de OpenAI [10].
- Conversión a vectores: Cada token se transforma en un vector numérico que representa su significado semántico, utilizando el modelo text-embedding-ada-002 [10].
- Almacenamiento de embeddings: Los vectores generados se almacenan en una base de datos vectorial, como ChromaDB [10].

Creación del índice de búsqueda vectorial: Para una búsqueda eficiente de los embeddings, se creará un índice de búsqueda vectorial utilizando ChromaDB [10]. Esta base de datos de código abierto se caracteriza por su simplicidad y estrecha integración con flujos de trabajo de aprendizaje automático, lo que la hace ideal para tareas de recuperación basadas en embeddings en proyectos de investigación y de menor escala [9]. **Optimización del índice:** ChromaDB utiliza el algoritmo de búsqueda de grafos Hierarchical Navigable Small World (HNSW) para una recuperación eficiente de los objetos coincidentes [10]. Este algoritmo ofrece un buen equilibrio entre velocidad y precisión en la búsqueda de vecinos más cercanos. Para optimizar aún más el índice, se pueden ajustar los parámetros de HNSW, como el número de conexiones por nodo y el tamaño de la lista de candidatos. También se puede considerar la adición de metadatos a los fragmentos de texto para una búsqueda más precisa. Por ejemplo, se pueden agregar etiquetas de tipo de documento o etiquetas de sección [9].

2.4 Modelo de Generación

- **Selección del LLM:** Para la generación de respuestas en este proyecto, se ha seleccionado Gemini, el modelo multimodal de lenguaje desarrollado por Google.
- **Justificación de la elección del LLM**
  - La decisión de utilizar Gemini se basa en un análisis de las capacidades de diversos LLMs, incluyendo GPT-4-1106-preview, Llama 2 y Mistral. Gemini se destaca por las siguientes razones:
    - **Rendimiento superior:** Gemini ha demostrado un rendimiento de vanguardia en numerosos benchmarks, incluyendo MMLU, donde alcanzó un nivel de precisión comparable al de expertos humanos [5]. Este alto rendimiento garantiza la generación de respuestas precisas y de alta calidad para las consultas de los usuarios.
    - **Capacidades multimodales:** A diferencia de otros modelos como GPT-4, Gemini está diseñado para procesar y comprender información multimodal, incluyendo texto, imágenes y audio. Esta capacidad abre un abanico de posibilidades para integrar diferentes tipos de contenido en el chatbot y ofrecer una experiencia más rica e interactiva a los usuarios.
    - **Precisión en tareas clave:** Las fuentes destacan que Gemini presenta una precisión superior al 94% en la generación de preguntas y la evaluación de respuestas [5], dos funcionalidades cruciales para un chatbot efectivo. Esta precisión asegura la creación de preguntas relevantes y la evaluación precisa de las respuestas proporcionadas por los usuarios.
  - Si bien la integración con LangChain para Gemini aún necesita ser verificada, su potencial en términos de rendimiento y capacidades multimodales lo convierten en la mejor opción para este proyecto.
- **Diseño del prompt para el LLM para la utilización de la información recuperada.**
  - El diseño del prompt para Gemini se enfoca en proporcionar al modelo la información necesaria para generar respuestas relevantes y precisas a las consultas de los usuarios. El prompt incluirá:
    - **Información contextual:** Se proporcionarán los fragmentos de texto recuperados mediante la búsqueda semántica, ofreciendo a Gemini el contexto necesario para comprender la consulta del usuario.
    - **Instrucción clara:** Se indicará a Gemini que genere una respuesta concisa y precisa que responda directamente a la pregunta del usuario, utilizando la información contextual proporcionada.
    - **Formato de salida:** Se especificará el formato de salida deseado para la respuesta (texto plano, lista, tabla, etc.), dependiendo de la naturaleza de la consulta.
  - **Ejemplo de prompt:**
    "Contexto: <fragmentos de texto recuperados> Pregunta: <pregunta del usuario> Instrucción: Genera una respuesta concisa y precisa a la pregunta del usuario utilizando la información del contexto. Formato de salida: <texto plano, lista con viñetas, tabla, etc.>"

2.5 Integración del Modelo de Recuperación y Generación

- **Descripción del proceso de recuperación de fragmentos relevantes.**
  - El proceso de recuperación de fragmentos relevantes se basa en la técnica de búsqueda semántica. Esta técnica permite encontrar información relevante en un corpus de documentos, incluso si no hay una coincidencia exacta de palabras clave. El proceso se puede describir en los siguientes pasos:
    1. Tokenización y embedding de la consulta: La consulta del usuario se tokeniza, es decir, se divide en palabras o frases individuales. Luego, se utiliza un modelo de embedding, como text-embedding-3-large de OpenAI, para convertir cada token en una representación vectorial [1].
    2. Búsqueda de similitud en la base de datos vectorial: La representación vectorial de la consulta se compara con las representaciones vectoriales de los fragmentos de texto almacenados en una base de datos vectorial, como ChromaDB [4] [2] [3]. Se utilizan algoritmos de búsqueda de vecinos próximos aproximados (ANN) para encontrar los fragmentos de texto más similares a la consulta.[1]
    3. Clasificación de los resultados: Los fragmentos de texto más similares se clasifican según su puntuación de similitud, calculada mediante la similitud del coseno[1]. Esto garantiza que los fragmentos más relevantes se presenten primero al LLM.[1]

- **Descripción del proceso de aumento de la consulta con la información recuperada.**
  - El proceso de aumento de la consulta implica combinar la consulta original del usuario con los fragmentos de texto recuperados para proporcionar un contexto más rico al LLM. El objetivo es mejorar la precisión y la relevancia de la respuesta generada.
  - En la conversación previa se definió un prompt para Gemini que incluye la información contextual, la pregunta del usuario y la instrucción para generar una respuesta precisa utilizando la información proporcionada. El prompt se estructura de la siguiente manera:

    Contexto: fragmentos de texto recuperados

    Pregunta: pregunta del usuario

    Instrucción: Genera una respuesta concisa y precisa a la pregunta del usuario utilizando la información del contexto. Formato de salida: texto plano, lista con viñetas, tabla, etc.
  - Al incluir los fragmentos de texto recuperados en el "Contexto" del prompt, se guía al LLM para que considere esta información al generar la respuesta, asegurando que la respuesta sea contextualmente relevante y esté respaldada por la información del corpus.

- **Descripción del proceso de generación de la respuesta por parte del LLM.**
  - El LLM, en este caso Gemini, recibe el prompt aumentado con la información recuperada y genera una respuesta en lenguaje natural. El proceso de generación de la respuesta es el siguiente:

1. Codificación del prompt: Gemini procesa el prompt y lo codifica en una representación interna.
2. Generación de texto: Utilizando su conocimiento del lenguaje y la información contextual del prompt, Gemini genera una respuesta palabra por palabra, prediciendo la siguiente palabra más probable en función del contexto.
3. Decodificación y formateo de la respuesta: La respuesta generada se decodifica y se formatea según las instrucciones del prompt, ya sea como texto plano, una lista con viñetas o una tabla.

– El resultado final es una respuesta que combina el conocimiento general del LLM con la información específica recuperada del corpus de documentos, proporcionando una respuesta precisa, relevante y contextualmente apropiada a la consulta del usuario.

3 Implementación del Sistema

3.1 Herramientas y Tecnologías Utilizadas

Bibliotecas y frameworks de Python utilizados: La implementación del sistema requirió el uso de Python, núcleo del proyecto, junto a bibliotecas y frameworks específicos. La biblioteca 'google-ai-generativelanguage' facilitó la comunicación con el modelo Gemini, gestionando prompts y respuestas. 'huggingface-hub' proporcionó acceso a modelos pre-entrenados y recursos de la comunidad PLN. El framework 'Flask' se empleó para construir la API web y la interfaz de la aplicación, permitiendo la interacción con el sistema. Para la integración con la plataforma de mensajería Telegram se utilizó 'python-telegram-bot', que simplificó el proceso de creación del bot y facilitó la comunicación con los usuarios en dicho canal. El uso de estas herramientas refleja un enfoque modular en el diseño, donde cada componente específico del sistema se beneficia de la funcionalidad que proporcionan estas bibliotecas, y facilitó el desarrollo y la escalabilidad del sistema. La selección de estas herramientas consideró su capacidad para simplificar el desarrollo y acelerar la implementación del sistema RAG. Estas bibliotecas demostraron ser eficientes para realizar tareas específicas en un entorno científico.

Bases de datos vectoriales utilizadas: Para el almacenamiento y búsqueda de representaciones vectoriales de texto, se implementó 'chromadb', una base de datos vectorial eficiente en la recuperación semántica. ChromaDB permite almacenar y consultar representaciones vectoriales, lo que resulta esencial para sistemas de búsqueda basados en la similitud semántica. La selección de esta base de datos se basó en su facilidad de uso, su integración con Python, su licencia de código abierto y su eficiencia en la gestión de datos vectoriales. La API de ChromaDB permitió una rápida implementación, facilitando el indexado de los embeddings generados a partir del corpus de texto, lo que a su vez permitió una búsqueda eficiente. La escalabilidad de ChromaDB hace posible la manipulación de grandes volúmenes de datos vectoriales, garantizando la eficiencia del sistema en el procesamiento de un creciente corpus. Su documentación resultó relevante durante la implementación, siendo fundamental para comprender y aplicar los mecanismos de la base de datos en la tarea de recuperación de información. Esta herramienta sirvió como componente central para la funcionalidad de recuperación, permitiendo búsquedas rápidas y relevantes a las consultas.

Entorno de desarrollo: La gestión y el versionado del código se realizaron en 'vs-code', un entorno de desarrollo integrado en el sistema operativo 'Windows 11 Pro'. Visual Studio Code, fue elegido por su versatilidad y su soporte para la manipulación y la depuración del código. La selección de este IDE se fundamentó en su amplio conjunto de extensiones, las cuales ofrecieron soporte para diversos lenguajes, frameworks y herramientas útiles para el desarrollo del sistema RAG. La integración de VS Code con el sistema de control de versiones Git facilitó el registro y la gestión de cambios en el código, promoviendo la colaboración en el desarrollo del sistema. La versión Pro del sistema operativo Windows 11 facilitó la compatibilidad con bibliotecas, frameworks y herramientas necesarias durante el proceso de implementación. La elección de este entorno permitió un flujo de trabajo adecuado para la escritura, versionado, manipulación y procesamiento del código, junto a herramientas para la gestión del proyecto y el despliegue del sistema. Este entorno de trabajo facilitó el desarrollo del proyecto.

3.2 Desarrollo del Pipeline de RAG

Pasos detallados para la implementación del pipeline:

- **Carga de Documentos:** Inicialmente, el sistema carga los documentos desde las fuentes establecidas (PDFs). Se emplean bibliotecas como PyPDF2, pdfplumber, o Tesseract OCR, según la naturaleza del documento, para extraer el contenido textual. Esta etapa implica el análisis de la estructura del documento y la recuperación del texto en formato adecuado para su posterior procesamiento.

- **Preprocesamiento:** El texto extraído se somete a una serie de pasos de preprocesamiento, como la conversión a minúsculas, la eliminación de caracteres especiales y la lematización. Estas transformaciones buscan uniformizar el texto y reducir el ruido, facilitando la posterior indexación y recuperación de la información.

- **Fragmentación del Texto:** Después del preprocesamiento, el texto se divide en fragmentos de tamaño fijo con solapamiento fijo. Los valores exactos del tamaño del fragmento y el solapamiento se determinan empíricamente, basándose en pruebas y experimentación con el objetivo de optimizar la coherencia de la información recuperada. Esta técnica garantiza una cantidad predecible de información en cada fragmento, permitiendo que la recuperación sea precisa y contextual.

- **Generación de Embeddings:** Una vez preprocesado, el texto se convierte en representaciones vectoriales, conocidas como embeddings. Para ello, se emplea un modelo específico, en este caso, se recurrió al modelo predeterminado de ChromaDB, que transforma el texto en vectores que capturan su significado semántico. Estos embeddings permiten comparar textos en función de su similitud semántica.

- **Indexación:** Los vectores resultantes se indexan en una base de datos vectorial, ChromaDB, para la búsqueda rápida y eficiente. Esta base de datos permite encontrar los fragmentos de texto más relevantes para una consulta, usando la similitud vectorial. La indexación requiere la organización de los embeddings en una estructura de datos que facilite la búsqueda rápida, y eficiente a través de algoritmos de vecinos próximos aproximados (ANN).

- **Búsqueda:** Al recibir la consulta de un usuario, se repiten los pasos de preprocesamiento y generación de embeddings, y se realiza una búsqueda en el índice para

encontrar los fragmentos de texto más relevantes. La búsqueda emplea la similitud del coseno para comparar la consulta con los fragmentos indexados.

- **Generación de Respuesta:** Finalmente, los fragmentos de texto recuperados se incorporan a un prompt, junto con la consulta original del usuario, para que Gemini genere una respuesta coherente, precisa y contextualmente relevante. El prompt también puede incluir instrucciones para el formato de la respuesta (texto plano, lista, tabla, etc.).

4 Evaluación del Sistema

4.1 Diseño de la Evaluación

Definición de las métricas de evaluación
Para evaluar la eficacia del sistema de chatbot, se utilizarán las siguientes métricas:

- **Relevancia de los documentos recuperados:**
  - *Precisión:* Se medirá la proporción de documentos recuperados que son realmente relevantes para la consulta del usuario.[8]
  - *Exhaustividad:* Se evaluará la capacidad del sistema para recuperar todos los documentos relevantes del corpus.[8]
- **Precisión y calidad de las respuestas generadas:**
  - *Exactitud:* Se determinará si las respuestas generadas por el LLM son factualmente correctas y están alineadas con la información proporcionada en los documentos recuperados.[8][6]
  - *Completitud:* Se evaluará si las respuestas proporcionan toda la información relevante para la consulta del usuario.[8]
- **Fluidez y coherencia de las respuestas:**
  - *Gramaticalidad:* Se verificará que las respuestas estén bien formadas gramaticalmente y sintácticamente.[8]
  - *Cohesión:* Se evaluará la fluidez del lenguaje y la conexión lógica entre las oraciones de la respuesta.[8]
- **Cobertura de las respuestas:** Se analizará la capacidad del sistema para responder a una amplia gama de preguntas y temas dentro del dominio de conocimiento.[8]

Métricas específicas
Se utilizarán las siguientes métricas específicas:

- METEOR: Evalúa la fluidez y la adecuación de la respuesta generada, considerando la sinonimia y la paráfrasis.[8]
- ROUGE: Mide el grado de solapamiento entre la respuesta generada y las respuestas de referencia, evaluando la exhaustividad, la precisión y la puntuación F1.[8]

- BLEU: Cuantifica la similitud entre la respuesta generada y las respuestas de referencia, analizando la presencia de n-gramas compartidos.[8]
- Perplejidad (PPL): Evalúa la capacidad del modelo lingüístico para predecir la siguiente palabra en una secuencia, lo que indica la fluidez y la naturalidad de la respuesta.[8]
- Similitud del coseno: Mide la similitud semántica entre la consulta del usuario y los documentos recuperados, lo que permite evaluar la relevancia de los resultados de la búsqueda.[8]
- Puntuación F1: Combina la precisión y la exhaustividad para proporcionar una medida general del rendimiento del sistema en la recuperación de información.[8]

Herramientas de evaluación

Se utilizarán las siguientes herramientas para la evaluación:

- ChromaDB: Se utilizará para la búsqueda semántica y la evaluación de la similitud del coseno.[6] [1]
- LangChain: Proporcionará herramientas para la tokenización, el embedding y la evaluación de las respuestas generadas.[7] [6] [1]

Aspectos adicionales a considerar

Además, se considerarán los siguientes aspectos:

- Facilidad de uso: Se evaluará la interfaz de usuario del chatbot para determinar su facilidad de uso e intuitividad.
- Tiempo de respuesta: Se medirá el tiempo que tarda el sistema en generar una respuesta, teniendo en cuenta la eficiencia del proceso de recuperación y generación.
- Robustez: Se analizará la capacidad del sistema para manejar consultas complejas o ambiguas, así como errores en la entrada del usuario.

La evaluación del sistema se lleva a cabo con la participación de usuarios reales, que interactuan con el chatbot y proporcionan retroalimentación sobre su experiencia. Los resultados de la evaluación se utilizan para mejorar el sistema de chatbot y garantizar que satisfaga las necesidades de los usuarios.

Metodología de evaluación:

- *Creación de un conjunto de datos de Ground Truth:* La metodología de evaluación se sustenta en dos pilares fundamentales: la creación de un conjunto de datos de Ground Truth y las pruebas con usuarios reales. La creación de un conjunto de datos de Ground Truth es esencial para medir la precisión y relevancia de las respuestas generadas por el sistema RAG. Este conjunto de datos se construye con preguntas, respuestas ideales y los documentos de referencia correspondientes, permitiendo una comparación objetiva del rendimiento del sistema. Esta etapa asegura que las métricas de evaluación se calculen a partir de una base sólida de información verificada, que sirva como una base para la evaluación cuantitativa del sistema.

- *Pruebas con usuarios reales:* Las pruebas con usuarios reales son indispensables para evaluar la utilidad y la usabilidad del sistema RAG en un entorno de uso real. A través de estas pruebas, los usuarios interactuarán con el chatbot, formulando preguntas y evaluando la relevancia, precisión y calidad de las respuestas generadas. Esta etapa permite identificar aspectos de mejora que podrían no ser evidentes en una evaluación basada únicamente en datos, además de proporcionar un feedback cualitativo sobre la experiencia de usuario. La combinación de ambas aproximaciones asegura una evaluación rigurosa y exhaustiva del sistema RAG.

4.2 Resultados de la Evaluación

- Presentación de los resultados de la evaluación usando las métricas definidas.
    - Presentación de los resultados de la evaluación usando las métricas definidas.
    - Análisis de los puntos fuertes y débiles del sistema.
    - Identificación de áreas de mejora.

5 Análisis y Discusión

5.1 Beneficios del Sistema RAG

- Mejora del acceso a la información y la comprensión de conceptos.
- Reducción del tiempo de búsqueda y solución de dudas.
- Aumento de la autonomía del estudiante.
- Potencial para personalizar el aprendizaje.

5.2 Desafíos y Limitaciones

- Dificultades técnicas encontradas durante el desarrollo.
- Limitaciones del sistema RAG (comprensión de preguntas ambiguas, posibles errores en las respuestas).
- Potenciales sesgos y limitaciones en el conjunto de datos y los modelos.
- Recomendaciones para superar estos desafíos.

6 Conclusiones y Trabajo Futuro

6.1 Conclusiones

- Resumen de los logros del proyecto.
- Respuesta a los objetivos planteados.
- Reflexión sobre el impacto del sistema en el proceso de aprendizaje.

6.2 Trabajo Futuro

- Propuestas para mejorar el sistema RAG (fine-tuning, nuevos modelos, mejorar la calidad de datos).
- Posibles extensiones del sistema (integración con otras herramientas, personalización más avanzada).
- Investigaciones futuras.

7 Anexos

7.1 Código fuente (snippets relevantes)

7.2 Tablas y gráficos con resultados de evaluación.

7.3 Glosario de términos técnicos.

7.4 Bibliografía y referencias.

References

[1] Niken Aisyah Maharani Herwanza, Nazruddin Safaat Harahap, Febi Yanto, and Fitri Insani. Penerapan langchain retriever dengan model chat openai dalam pengembangan sistem chatbot hadis berbasis telegram. *JTIM : Jurnal Teknologi Informasi dan Multimedia*, 6:70–83, 5 2024.

[2] Cheonsu Jeong. A study on the implementation of generative ai services using an enterprise data-based llm application architecture. *Advances in Artificial Intelligence and Machine Learning*, 3:1588–1618, 9 2023.

[3] Jason M. Keith, Amin Amirlatifi, Shahram Rahimi, Subash Neupane, and Sudip Mittal. Bark plug: The chatgpt of the bagley college of engineering at mississippi state university. *ASEE Annual Conference and Exposition, Conference Proceedings*, 6 2024.

[4] Marios Evangelos Mamalis, Evangelos Kalampokis, Fotios Fitsilis, Georgios Theodorakopoulos, and Konstantinos Tarabanis. A large language model agent based legal assistant for governance applications. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 14841 LNCS:286–301, 2024.

[5] Manoj Kumar Manmathan, Pankaj Agarwal, Suraj Ravi Shiwal, Nitin Bhore, Shagun Singal, and Bhaskar Saha. Organization-wide continuous learning (owcl): Personalized ai chatbots for effective post-training knowledge retention. *J. Electrical Systems*, 20:2568–2581, 2024.

[6] Frank Morales. Towards robust and interpretable text-to-sql generation: A mistral-based approach with comprehensive evaluation. *Authorea Preprints*, 7 2024.

[7] Chin Siang Ong, Nicholas T. Obey, Yanan Zheng, Arman Cohan, and Eric B. Schneider. Surgeryllm: a retrieval-augmented generation large language model framework for surgical decision support and workflow enhancement. *npj Digital Medicine 2024 7:1*, 7:1–5, 12 2024.

[8] Irina Radeva, Ivan Popchev, Lyubka Doukovska, and Miroslava Dimitrova. Web application for retrieval-augmented generation: Implementation and testing. *Electronics 2024, Vol. 13, Page 1361*, 13:1361, 4 2024.

[9] Antony Seabra, Claudio Cavalcante, Joao Nepomuceno, Lucas Lago, Nicolaas Ruberg, and Sergio Lifschitz. Dynamic multi-agent orchestration and retrieval for multi-source question-answer systems using large language models. 2024.

[10] Sabrina Toro, Anna V Anagnostopoulos, Sue Bello, Kai Blumberg, Rhiannon Cameron, Leigh Carmody, Alexander D Diehl, Damion Dooley, William Duncan, Petra Fey, Pascale Gaudet, Nomi L Harris, Marcin Joachimiak, Leila Kiani, Tiago Lubiana, Monica C Munoz-Torres, Shawn O'Neil, David Osumi-Sutherland, Aleix Puig, Justin P Reese, Leonore Reiser, Sofia Robb, Troy Ruemping, James Seager, Eric Sid, Ray Stefancsik, Magalie Weber, Valerie Wood, Melissa A Haendel, and Christopher J Mungall. Dynamic retrieval augmented generation of ontologies using artificial intelligence (dragon-ai). *Journal of Biomedical Semantics 2024 15:1*, 15:1–16, 12 2023.