Máster en Big Data. Tecnología y Analítica Avanzada

Trabajo Fin de Máster

Desarrollo de Estrategias de Adquisición y Organización de Datos para un LLM

Autora
Blanca Martínez Rubio

Director
Carlos Morrás Ruiz-Falcó

Madrid
Enero 2025

Blanca Martínez Rubio, declara bajo su responsabilidad, que el Proyecto con título **Desarrollo de Estrategias de Adquisición y Organización de Datos para un LLM** presentado en la ETS de Ingeniería (ICAI) de la Universidad Pontificia Comillas en el curso académico 2024/25 es de su autoría, original e inédito y no ha sido presentado con anterioridad a otros efectos. El Proyecto no es plagio de otro, ni total ni parcialmente y la información que ha sido tomada de otros documentos está debidamente referenciada.

Fdo.: ..... Fecha: 12/01/2025

Autoriza la entrega:

El Director del Proyecto

Carlos Morrás Ruiz-Falcó

Fdo.: ..... Fecha: 12/01/2025

V. B. del Coordinador de Proyectos

Carlos Morrás Ruiz-Falcó

Fdo.: ..... Fecha: 12/01/2025

AUTORIZACIÓN PARA LA DIGITALIZACIÓN, DEPÓSITO Y DIVULGACIÓN EN RED DE PROYECTOS FIN DE GRADO, FIN DE MÁSTER, TESINAS O MEMORIAS DE BACHILLERATO

1º. Declaración de la autoría y acreditación de la misma.

El autor Dña. Blanca Martínez Rubio **DECLARA** ser el titular de los derechos de propiedad intelectual de la obra: Desarrollo de Estrategias de Adquisición y Organización de Datos para un LLM, que ésta es una obra original, y que ostenta la condición de autor en el sentido que otorga la Ley de Propiedad Intelectual.

2º. Objeto y fines de la cesión.

Con el fin de dar la máxima difusión a la obra citada a través del Repositorio institucional de la Universidad, el autor **CEDE** a la Universidad Pontificia Comillas, los derechos de digitalización, de archivo, de reproducción, de distribución y de forma gratuita y no exclusiva, por el máximo plazo legal y con ámbito universal, de comunicación pública, incluido el derecho de puesta a disposición electrónica, tal y como se describen en la Ley de Propiedad Intelectual. El derecho de transformación se cede a los únicos efectos de lo dispuesto en la letra a) del apartado siguiente.

3º. Condiciones de la cesión y acceso

Sin perjuicio de la titularidad de la obra, que sigue correspondiendo a su autor, la cesión de derechos contemplada en esta licencia habilita para:

> [(a)]Transformarla con el fin de adaptarla a cualquier tecnología que permita incorporarla a internet y hacerla accesible; incorporar metadatos para realizar el registro de la obra e incorporar "marcas de agua" o cualquier otro sistema de seguridad o de protección. Reproducirla en un soporte digital para su incorporación a una base de datos electrónica, incluyendo el derecho de reproducir y almacenar la obra en servidores, a los efectos de garantizar su seguridad, conservación y preservar el formato. Comunicarla, por defecto, a través de un archivo institucional abierto, accesible de modo libre y gratuito a través de internet. Cualquier otra forma de acceso (restringido, embargado, cerrado) deberá solicitarse expresamente y obedecer a causas justificadas. Asignar por defecto a estos trabajos una licencia Creative Commons. Asignar por defecto a estos trabajos un HANDLE (URL *persistente*).

4º. Derechos del autor.

El autor, en tanto que titular de una obra tiene derecho a:

[(a)]Que la Universidad identifique claramente su nombre como autor de la misma Comunicar y dar publicidad a la obra en la versión que ceda y en otras posteriores a través de cualquier medio. Solicitar la retirada de la obra del repositorio por causa justificada. Recibir notificación fehaciente de cualquier reclamación que puedan formular terceras personas en relación con la obra y, en particular, de reclamaciones relativas a los derechos de propiedad intelectual sobre ella.

5º. Deberes del autor.

[(a)]El autor se compromete a: Garantizar que el compromiso que adquiere mediante el presente escrito no infringe ningún derecho de terceros, ya sean de propiedad industrial, intelectual o cualquier otro. Garantizar que el contenido de las obras no atenta contra los derechos al honor, a la intimidad y a la imagen de terceros. Asumir toda reclamación o responsabilidad, incluyendo las indemnizaciones por daños, que pudieran ejercitarse contra la Universidad por terceros que vieran infringidos sus derechos e intereses a causa de la cesión. Asumir la responsabilidad en el caso de que las instituciones fueran condenadas por infracción de derechos derivada de las obras objeto de la cesión.

6º. Fines y funcionamiento del Repositorio Institucional.

La obra se pondrá a disposición de los usuarios para que hagan de ella un uso justo y respetuoso con los derechos del autor, según lo permitido por la legislación aplicable, y con fines de estudio, investigación, o cualquier otro fin lícito. Con dicha finalidad, la Universidad asume los siguientes deberes y se reserva las siguientes facultades:

- La Universidad informará a los usuarios del archivo sobre los usos permitidos, y no garantiza ni asume responsabilidad alguna por otras formas en que los usuarios hagan un uso posterior de las obras no conforme con la legislación vigente. El uso posterior, más allá de la copia privada, requerirá que se cite la fuente y se reconozca la autoría, que no se obtenga beneficio comercial, y que no se realicen obras derivadas.

- La Universidad no revisará el contenido de las obras, que en todo caso permanecerá bajo la responsabilidad exclusive del autor y no estará obligada a ejercitar acciones legales en nombre del autor en el supuesto de infracciones a derechos de propiedad intelectual derivados del depósito y archivo de las obras. El autor renuncia a cualquier reclamación frente a la Universidad por las formas no ajustadas a la legislación vigente en que los usuarios hagan uso de las obras.
- La Universidad adoptará las medidas necesarias para la preservación de la obra en un futuro.
- La Universidad se reserva la facultad de retirar la obra, previa notificación al autor, en supuestos suficientemente justificados, o en caso de reclamaciones de terceros.

Madrid, a 12 de enero de 2025

ACEPTA

Fdo.:  .....

Motivos para solicitar el acceso restringido, cerrado o embargado del trabajo en el Repositorio Institucional:

Abstract

This project enhances large language models (LLMs) through advanced data scraping, crawling, and chunking techniques, focusing on refining data extraction and organization from documents in various formats, including PDFs, DOCX files, and web pages. By implementing Retrieval-Augmented Generation (RAG), the system integrates up-to-date, relevant information from external databases, bolstering the LLM's ability to generate accurate and coherent responses.

Challenges such as optimizing processing speed and ensuring scalability to adapt to various document formats were addressed, using AWS Lambda to implement the developed algorithms. Additionally, there is an exploration of the best strategies to optimize LLM performance to ensure responses are as precise as possible. Comparative evaluations with ChatGPT-4 and TruLens testing demonstrate the system's effectiveness, showing enhanced response accuracy and relevance.

Future work will aim to expand processing capabilities to additional document formats and further refine chunking techniques, ensuring the system remains at the forefront of natural language processing technology.

Resumen

Este proyecto consiste en mejorar el rendimiento de los modelos de lenguaje de gran tamaño (LLMs) mediante técnicas avanzadas de extracción y segmentación de datos. Se centra en optimizar dichas técnicas a partir de documentos en varios formatos, como PDFs, archivos DOCX y páginas web. Utilizando la técnica de Generación Aumentada por Recuperación (RAG), el sistema integra información actual y relevante de bases de datos externas, mejorando así la capacidad del LLM para generar respuestas precisas y coherentes.

Algunos de los desafíos enfrentados han sido la optimización de la velocidad de procesamiento y asegurar la escalabilidad para adaptarse a diferentes formatos de documentos, usando AWS Lambda para la implementación de los algoritmos desarrollados. Además, se han investigado las estrategias más efectivas para maximizar el rendimiento del LLM, garantizando respuestas altamente precisas.

Las comparaciones con ChatGPT-4 y las pruebas realizadas con TruLens han demostrado la eficacia del sistema, evidenciando una mejora en la precisión y la relevancia de las respuestas. Los planes a futuro incluyen ampliar las capacidades de procesamiento a más formatos de documentos y refinar aún más las técnicas de segmentación y extracción de datos para mantener al sistema a la vanguardia de la tecnología de procesamiento de lenguaje natural.

Agradecimientos

A mi tutor, Carlos Morrás, por creer en mí desde el primer día y darme la oportunidad de hacer las prácticas con él.

A los profesores del máster por dedicar su tiempo y esfuerzo a ayudarnos a aprender lo máximo posible.

A mis padres, por darme la oportunidad de hacer este máster y por haberme apoyado siempre, en los buenos y en los malos momentos.

Y a mi equipo de Votconnect, por convertir mi paso por la empresa en una etapa de crecimiento constante, tanto personal como profesional. Gracias por haber hecho de esta experiencia algo que siempre recordaré con cariño y alegría.

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

1.1. Contexto y justificación

En las últimas décadas, el desarrollo tecnológico ha avanzado a un ritmo sin precedentes, impactando profundamente en múltiples ámbitos de la sociedad. Entre estos avances, la inteligencia artificial (IA) se ha convertido en uno de los campos más innovadores y prometedores. Dentro de la IA, la aparición de modelos de lenguaje grande, conocidos como Large Language Models (LLMs), representa un salto cualitativo significativo. Estos modelos, como GPT-3.5 y sus sucesores, han demostrado una gran capacidad para comprender y generar texto humano, lo que abre una gran variedad de posibilidades en campos tan diversos como la educación, la atención médica, el servicio al cliente o la creación de contenido.

Los LLMs no solo pueden realizar tareas básicas de procesamiento de lenguaje natural, como traducir y clasificar texto, sino que también pueden escribir código, responder preguntas complejas, ayudar a tomar decisiones y generar texto coherente y contextualmente relevante. Este nivel de sofisticación se debe a su entrenamiento con grandes volúmenes de datos, lo que les permite aprender patrones lingüísticos y conocimientos a partir de un amplio espectro de información textual.

A medida que las empresas y organizaciones buscan aprovechar el potencial de los LLMs para sus necesidades específicas, surge la necesidad de personalizar estos modelos. La personalización de un LLM implica adaptarlo para que pueda manejar y generar texto en contextos específicos, alineados con los requisitos y objetivos particulares de la organización. El primer paso en este proceso es la adquisición y organización de datos relevantes.

Obtener información de documentos y páginas web y adaptarla para el entre-

namiento de un LLM personalizado es un desafío técnico significativo. Es fundamental que los datos recopilados estén bien estructurados y divididos de manera coherente para que el modelo pueda aprender de ellos de manera efectiva. Este proceso, conocido como chunking, permite dividir los datos en unidades manejables que facilitan el procesamiento y análisis por parte del LLM. Al emplear técnicas avanzadas de scraping y crawling, es posible extraer datos de diversos formatos (como documentos Word, archivos TXT y PDFs) y desde diferentes fuentes web, garantizando así una base de datos rica y variada para el modelo.

1.2. Objetivos del TFM

1.2.1. Objetivo general

El objetivo general de este Trabajo de Fin de Máster es desarrollar un sistema eficiente de adquisición y organización de datos que permita la extracción y segmentación de información proveniente de documentos y páginas web. Dicha información se usará para mejorar la precisión y relevancia de las respuestas proporcionadas por un LLM en aplicaciones prácticas. Este sistema deberá emplear técnicas avanzadas de scraping y crawling para recopilar datos en diversos formatos y aplicar métodos de chunking para asegurar que los datos se estructuran de manera coherente y utilizable por el LLM.

1.2.2. Objetivos específicos

Los objetivos específicos son los siguientes:

1. **Investigar y analizar técnicas avanzadas de scraping y crawling:**
    - Revisar la literatura y las tecnologías actuales relacionadas con la extracción de datos web y su procesamiento.
    - Seleccionar las herramientas y bibliotecas más adecuadas para la implementación del sistema de scraping y crawling.
2. **Desarrollar un sistema de scraping y crawling:**
    - Implementar scripts que permitan la recolección de datos desde documentos en formatos comunes (DOCX, TXT, PDF) y páginas web.
    - Garantizar que el sistema pueda manejar grandes volúmenes de datos de manera eficiente y escalable.

3. **Implementar técnicas de chunking para la segmentación de datos:**
    - Desarrollar algoritmos que permitan dividir los datos extraídos en unidades manejables y coherentes.
    - Asegurar que los chunks de datos mantengan la coherencia semántica y sean adecuados para el uso del LLM.
4. **Evaluar la eficiencia del sistema de adquisición y organización de datos:**
    - Medir la eficiencia y precisión del sistema de scraping y crawling.
    - Evaluar la coherencia y utilidad de los datos segmentados.
5. **Integrar los datos procesados en el uso de un LLM:**
    - Preparar y alimentar el LLM con los datos segmentados.
    - Realizar pruebas preliminares para evaluar el impacto de los datos recolectados en el rendimiento del LLM.
6. **Documentar el proceso y los resultados obtenidos:**
    - Describir detalladamente el desarrollo e implementación del sistema, incluyendo los retos y soluciones encontradas.
    - Analizar los resultados y extraer conclusiones sobre la viabilidad y efectividad del sistema desarrollado.

1.3. Metodología

La metodología se organiza en varias etapas que abarcan desde la recolección de datos hasta su preparación para el uso en el LLM.

1.3.1. Herramientas y tecnologías

Para la implementación del sistema de scraping y crawling, se ha optado por utilizar el lenguaje de programación Python debido a su versatilidad y la amplia disponibilidad de bibliotecas especializadas en el procesamiento de datos.

La infraestructura subyacente del proyecto se basa en Amazon Web Services (AWS), aprovechando sus servicios de computación en la nube para asegurar escalabilidad y eficiencia. En particular, se utilizan AWS App Runner para el despliegue y ejecución del código de scraping en funciones AWS Lambda gestionadas por Step Functions, también de AWS.

1.3.2. Proceso de adquisición de datos

El flujo para la recolección de datos se inicia cuando el cliente sube los documentos o URLs a la página web de la empresa. Este flujo se describe en los siguientes pasos:

1. **Subida de Documentos y URLs:**
   - El cliente carga uno o varios documentos (en formatos como DOCX, TXT, PDF) o URLs directamente en la plataforma web de la empresa.
2. **Descarga y Extracción de Datos desde Documentos:**
   - Los documentos son subidos al servicio de almacenamiento de S3 y posteriormente son descargados y almacenados temporalmente en el sistema para su procesamiento.
   - Las URLs proporcionadas son procesadas mediante scripts de scraping que extraen el contenido relevante de las páginas web.
   - Los documentos son procesados utilizando bibliotecas específicas según el tipo de documento para extraer el texto.

1.3.3. Técnicas de chunking

Una vez extraídos los datos, se procede a su segmentación mediante técnicas de chunking. El chunking empleado en este proyecto es heurístico, basándose en un tamaño máximo predefinido de los segmentos y procurando dividir los datos por secciones naturales del texto siempre que sea posible. Este método permite crear unidades de datos manejables y coherentes que pueden ser procesadas eficazmente por el LLM.

El proceso de chunking, que se explicará en más detalle en secciones posteriores, se realiza en los siguientes pasos:

1. **Definición del Tamaño de Chunk:**
   - Se establece un tamaño máximo para cada chunk, asegurando que las unidades de datos no excedan la capacidad del modelo de lenguaje para procesarlas de manera efectiva.
2. **División Heurística del Texto:**
   - El texto se divide en chunks basándose en secciones lógicas como párrafos y subtítulos, siempre que sea posible. En caso de no ser viable, se divide el texto conforme al tamaño máximo definido.

1.3.4. Evaluación y validación

Aunque la evaluación del impacto de los datos en el rendimiento del LLM no forma parte directa de este TFM, existe una pipeline de evaluación desarrollada por la empresa que se encarga de hacer preguntas al LLM y evaluar sus respuestas en base a un conjunto de verdades establecidas (*ground truth*), lo que permite medir la efectividad del modelo de lenguaje.

1.3.5. Infraestructura y entorno de desarrollo

El desarrollo y despliegue del sistema se realiza utilizando los servicios de AWS, que ofrecen una infraestructura robusta y escalable. Las principales componentes de esta infraestructura incluyen:

1. **AWS Lambda:**
   - Servicio de computación serverless que permite ejecutar código sin necesidad de aprovisionar o gestionar servidores. Las funciones Lambda se activan en respuesta a eventos específicos, como la carga de un archivo, y escalan automáticamente con el volumen de solicitudes, proporcionando una solución flexible y eficiente para tareas de procesamiento en paralelo.
   - En este proyecto, AWS Lambda se utiliza para ejecutar las funciones que extraen datos de los documentos o URLs. Cada tipo de documento (por ejemplo, PDF, DOCX, TXT) tiene su propia función Lambda que procesa el documento y extrae el texto relevante.
2. **AWS Step Functions:**
   - Servicio de orquestación que permite coordinar múltiples servicios de AWS en flujos de trabajo serverless. Utiliza diagramas de estado para definir y controlar la secuencia de ejecución de múltiples funciones Lambda, facilitando la creación de aplicaciones distribuidas y procesos automatizados complejos.
   - En este proyecto, AWS Step Functions orquesta el flujo de trabajo de extracción de datos, coordinando la ejecución de múltiples funciones Lambda.
3. **AWS App Runner:**
   - Servicio que facilita la creación, implementación y escalado de aplicaciones web y API en contenedores. Proporciona una infraestructura que

simplifica el despliegue de aplicaciones basadas en contenedores, eliminando la necesidad de gestionar servidores o clústeres.

- En este proyecto, AWS App Runner se utiliza para invocar las Step Functions correspondientes a cada tipo de documento. Cada Step Function gestiona el proceso específico de extracción de datos, asegurando que se manejen correctamente los diferentes formatos de documentos (como PDF, Word, y TXT).

Esta metodología asegura que el sistema desarrollado sea eficiente, escalable y capaz de manejar grandes volúmenes de datos de manera efectiva, proporcionando una base sólida para la personalización de LLMs.

1.4. Estructura del documento

- **Capítulo 1 - Introducción:** presenta el contexto, la justificación del proyecto, y los objetivos. Describe brevemente la metodología empleada.
- **Capítulo 2 - Estado del Arte:** revisa la literatura y tecnologías relevantes en scraping, crawling, chunking de datos y personalización de LLMs.
- **Capítulo 3 - Desarrollo del Proyecto:** detalla el desarrollo del sistema, incluyendo herramientas y tecnologías utilizadas, el proceso de implementación de scraping y las técnicas de chunking. Explica la integración de estos componentes usando AWS Lambda, App Runner y Step Functions.
- **Capítulo 4 - Resultados:** se presentan y discuten los resultados obtenidos, incluyendo el impacto en el rendimiento del LLM y las limitaciones del proyecto.
- **Capítulo 5 - Conclusiones y Trabajos Futuros:** resume las principales conclusiones y propone posibles líneas de investigación futuras.

Capítulo 2

Estado del Arte

2.1. Introducción a los modelos de lenguaje grandes (LLM)

En los últimos años, los Modelos de Lenguaje de Gran Escala (LLMs, por sus siglas en inglés) han transformado significativamente el campo de la inteligencia artificial y el procesamiento del lenguaje natural. Los LLMs son una subcategoría de la inteligencia artificial generativa, y su popularidad ha crecido enormemente con la aparición de aplicaciones como ChatGPT de OpenAI.

2.1.1. Breve historia de los LLMs

El desarrollo de los LLMs ha evolucionado mucho a lo largo de los años. Desde los primeros modelos de lenguaje, se avanzó hacia las redes neuronales en los años 2000, que mejoraron significativamente la representación y procesamiento de secuencias de texto. Un hito importante fue la introducción de las redes de Transformadores (Transformers) en 2017 por Google, en el paper *Attention Is All You Need*. Los Transformers revolucionaron el campo permitiendo una paralelización eficiente y mejorando el procesamiento de secuencias largas.

En 2018, OpenAI lanzó el modelo GPT (Generative Pre-Trained Transformer), que demostró cómo entrenar un LLM en un gran conjunto de datos de manera genérica para luego ajustarlo a tareas específicas. La evolución continuó con GPT-3 en 2020, que mostró capacidades de *few-shot learning*, es decir, la capacidad de resolver muchas tareas con mínima instrucción adicional. En 2022, OpenAI introdujo InstructGPT, que implementa el Aprendizaje Reforzado a partir de Realimentación Humana (RLHF) para mejorar la calidad y reducir el sesgo de las

respuestas generadas por los modelos.

En 2023, OpenAI introdujo GPT-4, mejorando aún más las capacidades de sus predecesores con un mejor rendimiento en tareas de comprensión y generación de lenguaje. También se lanzó una variante optimizada, GPT-4 Turbo, que ofrecía mayores eficiencias en términos de costo y velocidad de procesamiento. Posteriormente, en mayo de 2024, se presentó GPT-4o [1], que acepta como entrada cualquier combinación de texto, audio, imagen y vídeo y genera, mucho más rápido que los modelos anteriores, cualquier combinación de salidas de texto, audio e imagen. En septiembre de 2024, OpenAI introdujo el modelo o1, diseñado para dedicar más tiempo al razonamiento antes de responder, mejorando su desempeño en tareas complejas de ciencia y matemáticas [2].

Además de los modelos de OpenAI, otras empresas han desarrollado LLMs que han demostrado tener un gran rendimiento. Claude, de Anthropic, y LLaMA de Meta, son ejemplos de modelos que han están a la vanguardia en términos de generación de texto y comprensión de lenguaje natural. Estos modelos han sido optimizados para ser eficientes y efectivos en diversas aplicaciones industriales y comerciales [3]. En 2024, Anthropic lanzó Claude 3.5 Sonnet, su modelo más avanzado hasta la fecha, capaz de procesar entradas de texto e imagen y generar salidas de texto con una ventana de contexto de 200k tokens. Por su parte, Meta presentó LLaMA 3.1 en julio de 2024, un modelo de código abierto con 405 mil millones de parámetros [4], y LLaMA 3.2 en septiembre de 2024, un modelo multimodal capaz de procesar tanto texto como imágenes.

2.1.2. Funcionamiento de los LLM

Los LLMs representan las palabras mediante vectores multidimensionales, conocidos como incrustaciones de palabras (*word embeddings*), que posicionan palabras con significados contextualmente similares cerca unas de otras en un espacio vectorial. Esto permite a los transformadores preprocesar el texto y comprender el contexto y las relaciones entre palabras y frases. Durante el entrenamiento, los modelos ajustan iterativamente sus parámetros para maximizar la probabilidad de predecir correctamente la siguiente palabra en una secuencia de texto, utilizando técnicas de autoaprendizaje. [5]

2.1.3. Importancia y aplicaciones

Los LLMs son increíblemente versátiles y pueden realizar una gran cantidad de tareas, como responder preguntas, resumir documentos, traducir idiomas, generar texto coherente y mucho más. Esta flexibilidad tiene el potencial de transformar

diversas industrias, desde la creación de contenido hasta el uso de motores de búsqueda y asistentes virtuales [6].

Algunos ejemplos de aplicaciones son:

- Generación de texto publicitario: modelos como ChatGPT, Claude, Llama 2, Cohere Command, y Jurassic pueden escribir copias originales y sugerir mejoras estilísticas. AI21 Wordspice, por ejemplo, sugiere cambios para mejorar el estilo y la voz de las oraciones.
- Respuesta a preguntas en bases de conocimiento: los LLMs pueden responder preguntas específicas basadas en información de archivos digitales, como lo hace AI21 Studio Playground.
- Clasificación de textos: pueden agrupar textos por significados o sentimientos, facilitando la medición de opiniones de clientes y la búsqueda de documentos.
- Generación de código: herramientas como GitHub Copilot utilizan LLMs para generar código en varios lenguajes de programación a partir de indicaciones en lenguaje natural. Ejemplos incluyen Amazon CodeWhisperer y OpenAI Codex.
- Generación de documentos: los LLMs pueden completar oraciones, escribir documentación técnica y crear historias, como lo hace Alexa Create al escribir cuentos infantiles.

2.1.4. Retrieval-Augmented Generation (RAG)

Retrieval-Augmented Generation (RAG) representa un marco de trabajo avanzado en inteligencia artificial que busca mejorar la calidad de las respuestas generadas por modelos de lenguaje grandes (LLMs) integrando información extraída de bases de conocimiento externas. Este enfoque permite a los LLMs apoyarse en información precisa y actualizada, ofreciendo así respuestas más confiables y comprensibles.

Los modelos de lenguaje grandes, a pesar de su capacidad para generar respuestas coherentes, pueden ser inconsistentes en su rendimiento. En ocasiones, proporcionan respuestas precisas, pero otras veces pueden responder con hechos aleatorios que no se tienen nada que ver con las preguntas formuladas. Esto se debe a que los LLMs procesan el lenguaje basándose en relaciones estadísticas entre palabras, sin comprender su significado real. RAG aborda este desafío al *anclar* el

modelo en fuentes de conocimiento externas[7].

Implementar RAG en sistemas de respuestas basados en LLMs ofrece dos beneficios principales:

- Acceso a hechos actuales y fiables: asegura que el modelo utilice la información más reciente, mejorando la precisión de las respuestas.
- Verificabilidad de las fuentes: permite a los usuarios consultar las fuentes de las respuestas del modelo, garantizando así la veracidad y fiabilidad de la información proporcionada.

La Figura 2.1 muestra el proceso que se lleva a cabo al formular una pregunta a un LLM que usa RAG.

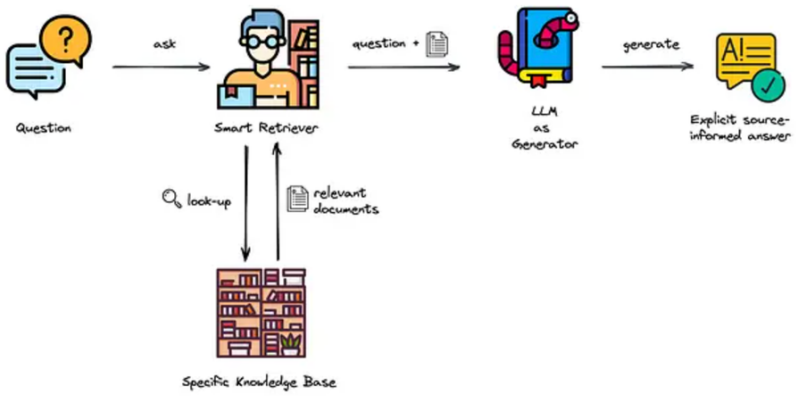

Figura 2.1: RAG

2.1.5. Futuro de los LLMs

El futuro de los LLMs es prometedor, con continuas mejoras en precisión y capacidades. A medida que se desarrollen versiones más avanzadas, se espera que los modelos sean más precisos y capaces de reducir sesgos y respuestas incorrectas.

Algunas de las nuevas posibilidades y transformaciones que se presentan en un futuro próximo son [5]:

- Entrenamiento audiovisual: además de texto, algunos LLMs están comenzando a entrenarse con entradas de video y audio, lo que debería acelerar el desarrollo y abrir nuevas aplicaciones, como en vehículos autónomos.

- Transformación del lugar de trabajo: los LLMs podrían automatizar tareas monótonas y repetitivas, impactando áreas como la administración, atención al cliente y redacción automatizada.
- Mejora en asistentes virtuales: se espera que los LLMs mejoren significativamente los asistentes virtuales, como Alexa, Google Assistant y Siri, permitiendo una interpretación más precisa de las intenciones del usuario y respuestas a comandos más sofisticados.

2.2. Técnicas de scraping y crawling

El scraping y crawling de datos son técnicas esenciales para la extracción de información de diversas fuentes como la web. Estas técnicas permiten automatizar la recopilación de datos estructurados y no estructurados, facilitando su análisis y utilización en modelos de lenguaje como los LLM.

- **Scraping:** consiste en identificar un sitio web u otra fuente que contenga información deseable y utilizar software para extraer la información del sitio en grandes volúmenes. Esta técnica permite a los usuarios acceder a documentos, como PDFs o Word, o a sitios web para extraer de ellos datos estructurados de forma automática. [8]
- **Crawling:** es el proceso mediante el cual un programa automatizado, conocido comúnmente como crawler o araña, navega por la web. Este bot analiza el contenido y el código de las páginas web, moviéndose de una a otra a través de los enlaces disponibles. Este método sistemático permite al crawler descubrir y registrar nueva información, facilitando la construcción de una amplia base de datos.[9]

2.2.1. Herramientas y tecnologías

En el ámbito del scraping y crawling, herramientas como Beautiful Soup y Scrapy son esenciales para la extracción eficaz de datos web. Beautiful Soup facilita el parseo (captura de información desde un formato y traducción a otro especificado) de documentos HTML y XML, ideal para proyectos que requieren limpieza de datos. Por su parte, Scrapy ofrece un framework robusto para scraping a gran escala, gestionando solicitudes y exportación de datos de manera eficiente.

Para documentos como PDFs y Word, herramientas como PDFMiner y PyPDF2 son cruciales. PDFMiner extrae texto de PDFs convirtiéndolos en texto plano, mientras que PyPDF2 permite manipular y extraer información de los PDFs.

python-docx es útil para trabajar con documentos Word, permitiendo la modificación y extracción de datos. PDFQuery se especializa en la extracción precisa de datos de PDFs mediante consultas XML, facilitando el manejo de documentos complejos. Estas son algunas de las herramientas que existen para proyectos de extracción de datos [10].

2.2.2. Casos de uso

El scraping de datos se ha convertido en una herramienta esencial para múltiples aplicaciones prácticas en diversos sectores, proporcionando una fuente valiosa de información para la toma de decisiones y la estrategia empresarial.

Algunos de sus usos son [11]:

- Inteligencia de negocios: el scraping se utiliza para recopilar inteligencia empresarial contenida en las páginas web. Las empresas analizan tendencias de contenido y comportamiento del usuario para optimizar sus estrategias de marketing digital y mejorar la participación del usuario.
- Determinación de precios: en sitios de reservas de viajes o plataformas de comparación, el scraping permite monitorear y comparar precios en tiempo real, ofreciendo a los consumidores la mejor opción disponible y ayudando a las empresas a ajustar sus precios en función de la competencia.
- Investigación de mercado y generación de leads: mediante el scraping de fuentes de datos públicas, las empresas pueden generar leads de ventas y realizar investigaciones de mercado exhaustivas.
- Integración con plataformas de compras online: el scraping facilita la recopilación y el envío de datos de productos desde sitios de eCommerce a plataformas de compras online como Google Shopping.
- Alimentación de modelos RAG: el scraping es crucial para desarrollar modelos de Retrieval Augmented Generation (RAG), que combinan la extracción de información relevante con la generación de texto. El scraping permite recopilar datos de diversas fuentes, proporcionando a los modelos RAG un repositorio rico y actualizado de textos para mejorar su capacidad de respuesta y precisión en la generación de contenidos y respuestas.

2.3. Procesamiento y organización de datos

En el ámbito del aprendizaje automático y los modelos de lenguaje, el procesamiento y la organización de datos son etapas muy importantes que afectan

directamente la calidad y eficiencia de los resultados. Los datos en bruto, a menudo en formatos diversos y desorganizados, deben ser procesados y estructurados adecuadamente para maximizar la utilidad de los LLM. Este proceso incluye la segmentación de grandes bloques de texto en fragmentos manejables y la conversión de diferentes formatos de archivo (como Word, TXT y PDF) a formas más procesables.

2.3.1. Técnicas de chunking

En el contexto de los modelos de lenguaje de gran tamaño (LLM), los embeddings juegan un papel fundamental. Los embeddings son representaciones numéricas de trozos de texto que capturan el significado y el contexto del contenido de una manera que las máquinas pueden entender. Estas representaciones permiten que el texto sea comparado y recuperado eficientemente con base en su similitud semántica.

Cuando se inserta contenido en una base de datos vectorial, se debe obtener el embedding de cada cada fragmento de texto. Luego, cuando se realiza una búsqueda o consulta a un LLM, la misma técnica de embedding se aplica a la consulta del usuario. La base de datos vectorial busca los embeddings que son más similares a la consulta, devolviendo los fragmentos de texto más relevantes al LLM, que los utiliza para responder de manera más precisa.

Ahora, para optimizar este proceso de indexación y recuperación, el chunking es esencial. El chunking implica dividir grandes bloques de texto en segmentos más pequeños y manejables. La principal razón para realizar chunking es garantizar que se está haciendo el embedding de un trozo de contenido con la menor cantidad de ruido posible, manteniendo su relevancia semántica. Esto se traduce en embeddings más precisos y, por lo tanto, en una recuperación de información más eficiente y precisa. [12].

Algunos ejemplos de técnicas de chunking son:

- Chunking de tamaño fijo: consiste en definir un número fijo de palabras por fragmento, y opcionalmente, incluir un solapamiento entre ellos para mantener el contexto semántico. Es una técnica computacionalmente eficiente y sencilla, ya que no requiere el uso de bibliotecas de procesamiento del lenguaje natural.

- Chunking sensible al contenido: estas técnicas aprovechan la naturaleza del contenido para aplicar chunking más sofisticado.

- División por oraciones: muchos modelos están optimizados para trabajar a nivel de oraciones. Hay varias herramientas para esto, como NLTK o SpaCy.

  Chunking recursivo: divide el texto en fragmentos más pequeños de manera jerárquica e iterativa utilizando diferentes separadores hasta lograr el tamaño o estructura deseada. Aunque los fragmentos no serán exactamente del mismo tamaño, tenderán a ser similares, manteniendo la coherencia semántica.
- Chunking especializado: para contenidos estructurados y formateados, como Markdown o LaTeX, se pueden usar métodos de chunking especializados que preservan la estructura original del contenido.
- Chunking semántico: en lugar de utilizar un tamaño global de chunking, se crean grupos de oraciones que tratan sobre el mismo tema o contexto. Esto se logra analizando la distancia semántica entre grupos de oraciones consecutivas, permitiendo delinear fragmentos que mantengan la coherencia temática. Este enfoque utiliza los embeddings para extraer el significado presente en los datos y agrupar oraciones relacionadas.

2.3.2. Formatos de datos comunes (DOCX, TXT, PDF)

Además del chunking, es fundamental manejar correctamente los diferentes formatos de archivo que contienen los datos que alimentarán a los LLM. Los formatos más comunes incluyen DOCX (Word), TXT y PDF, cada uno con sus propias características y desafíos específicos.

- Word (DOC/DOCX): Los archivos de Word son ampliamente utilizados para la creación de documentos con formato. Estos archivos pueden contener una variedad de elementos, como texto, imágenes, tablas y más.
- TXT: Los archivos TXT son simples archivos de texto sin formato que contienen solo texto sin ningún formato adicional. Son fáciles de procesar y no requieren herramientas complejas para la extracción de contenido.
- PDF: Los archivos PDF son populares por su capacidad de preservar el diseño y el formato de un documento en múltiples plataformas. Sin embargo, la extracción de texto de PDFs puede ser complicada debido a la naturaleza de su formato. Es importante utilizar herramientas adecuadas que puedan manejar correctamente la conversión y extracción del contenido de texto.

La combinación de técnicas de chunking efectivas con el manejo adecuado de diversos formatos de archivo asegura que los datos sean procesados de manera

eficiente y precisa, mejorando así la calidad y relevancia de los resultados obtenidos por los LLM.

2.4. Retos y desafíos en la extracción y procesamiento de datos

La extracción y procesamiento de datos para LLMs presentan diversos retos. Uno de los desafíos principales es la calidad y relevancia de los datos utilizados. En un LLM que utiliza un sistema RAG los datos deben ser precisos y actualizados para proporcionar respuestas pertinentes. Sin embargo, la recopilación de datos de múltiples fuentes puede introducir inconsistencias y ruido, lo que requiere un proceso riguroso de limpieza y normalización para asegurar la integridad de la información.

Además, la fragmentación (chunking) y estructuración de los datos son aspectos cruciales. Es necesario dividir correctamente los documentos en trozos manejables y contextualmente coherentes, lo cual es vital para la efectividad del sistema. Esto incluye técnicas de chunking que deben equilibrar el tamaño del fragmento para mantener la relevancia contextual sin perder información crítica.

Finalmente, la gestión de la latencia y la eficiencia en la recuperación de datos es un reto, ya que el sistema debe ser capaz de buscar y proporcionar información de manera rápida y precisa, sin sacrificar la calidad de las respuestas generadas.

Capítulo 3

Desarrollo del proyecto

3.1. Diseño del sistema de scraping y crawling

En esta sección, se detalla el diseño y la implementación del sistema de scraping y crawling utilizado para recolectar datos desde diversas fuentes. La recolección de datos es un componente crucial del proyecto, ya que proporciona la información necesaria para ser procesada y segmentada antes de su integración con un Modelo de Lenguaje de Gran Escala (LLM). El sistema se ha diseñado para extraer datos tanto desde documentos como desde la web, asegurando la captura de información relevante y precisa. A continuación, se describen las estrategias y técnicas empleadas.

3.1.1. Recolección de datos desde documentos

La recolección de datos desde documentos implica extraer información de diversos tipos de archivos, como documentos Word (DOCX), archivos TXT y documentos PDF. Esta sección aborda las técnicas específicas utilizadas para procesar y extraer contenido de cada tipo de documento. La selección de métodos adecuados para cada formato es fundamental para garantizar la precisión y eficiencia en la obtención de datos.

Documentos DOCX

1. **Inicio de la función (inicialización de variables)**

   El proceso se inicia con una función que recibe dos parámetros con la información necesaria para ejecutar la función Lambda, como el nombre del documento o su ruta dentro del sistema de almacenamiento de AWS, S3. En este paso se lleva a cabo la extracción de información relevante de dichos parámetros.

2. **Descarga del documento de S3**

   Una vez inicializadas las variables, el siguiente paso es descargar el documento desde S3 al dispositivo local donde se llevará a cabo el procesamiento. Para lograr esto se crea una ruta local, asignándole los permisos necesarios a la carpeta creada para asegurar que el archivo pueda ser leído y modificado según sea necesario.

   A continuación se procede a descargar el archivo desde S3. Hay una función correspondiente que se encarga de establecer una conexión con S3, utilizar la ruta proporcionada para localizar el archivo y descargarlo a la ubicación local, que es en la carpeta *tmp* (temporal).

3. **Parseo y extracción de texto**

   a) El primer paso es determinar si el documento tiene formato .doc o .docx. Si el documento está en formato .doc es necesario convertirlo, ya que la biblioteca que se va a utilizar acepta únicamente archivos con extensión .docx.

   b) A continuación se crea una estructura de directorios temporal para almacenar las imágenes extraídas del documento. De esta manera se garantiza un almacenamiento ordenado y accesible.

   c) Una vez preparadas las carpetas y el documento se pasa a la lectura del mismo utilizando una biblioteca llamada *python_docx*, que lee el contenido del archivo DOCX. Se inicializa un objeto Doc que es una instancia que encapsula toda la estructura del documento. A través de este objeto, se pueden acceder y manipular distintos componentes de dicho documento, como párrafos, tablas o imágenes, y brinda métodos para obtener información detallada sobre su contenido y estructura.

      Para poder extraer el texto de manera ordenada y jerarquizada se preparan variables para almacenar el texto y las secciones.

   d) Se itera sobre los párrafos del documento para extraer su contenido. Se analiza cada párrafo para determinar si contiene un encabezado y en caso de que sea así, cuál es su nivel jerárquico (título o subtítulo).

      La manera de estructurar el contenido es mediante una lista de diccionarios clave-valor en el que la clave es el encabezado (si hay), y el valor es el contenido de dicha sección. De esta manera se mantiene la estructura lógica del documento. Por lo tanto si se encuentra un nuevo

encabezado, se crea un nuevo diccionario y se agrega el contenido correspondiente. Cuando acabe la sección, el diccionario se añade a la lista.

Si un párrafo contiene imágenes, estas se extraen y se guardan en la estructura de directorios creada previamente. Se utiliza un modelo de lenguaje (Claude 3 Sonnet, de Antrophic) para obtener descripciones detalladas de las imágenes, que se incorporan al contenido del documento. Solo se analizan si ocupan más del 1 % de la página, asumiendo que las imágenes menores no son relevantes para el contenido.

Por ejemplo, dada la imagen de la Figura 3.1, la descripción proporcionada por el modelo sería:

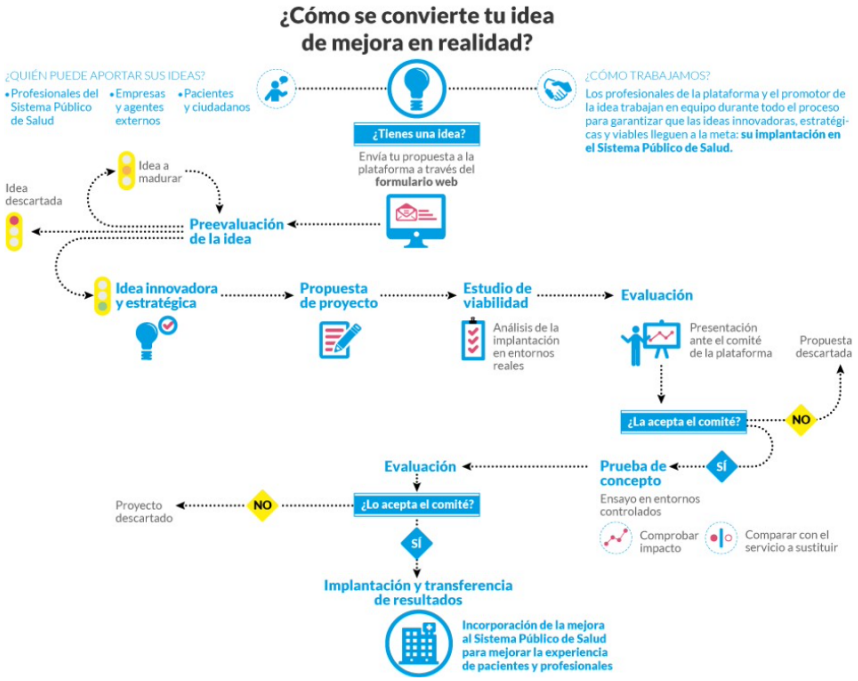

Figura 3.1: Workflow

> La imagen presenta un diagrama de flujo que ilustra el proceso mediante el cual una idea de mejora puede convertirse en realidad y ser implementada en el Sistema Público de Salud. A continuación, describiré detalladamente cada paso de este proceso:

Primero, las ideas pueden ser aportadas por profesionales del Sistema Público de Salud, empresas y agentes externos, así como por pacientes y ciudadanos. Si alguien tiene una idea de mejora, puede enviarla a través del formulario web de la plataforma correspondiente. Una vez la idea es recibida, pasa por una preevaluación inicial.

Si la idea resulta innovadora y estratégica, se elabora una propuesta de proyecto más detallada. Esta propuesta es sometida a un estudio de viabilidad, que incluye un análisis de su posible implementación en entornos reales. Posteriormente, la propuesta se presenta ante un comité de la plataforma para su evaluación.

En caso de que el comité acepte la propuesta, se realiza una prueba de concepto mediante ensayos en entornos controlados. Además, se compara el impacto de la implementación con el servicio que se pretende sustituir o mejorar. Si el comité vuelve a dar su visto bueno tras esta fase, el proyecto avanza hacia su implementación y transferencia de resultados.

Finalmente, la mejora se incorpora al Sistema Público de Salud con el objetivo de mejorar la experiencia de pacientes y profesionales. Por otro lado, si en algún punto del proceso el comité rechaza la propuesta, el proyecto se descarta.

Este es el proceso que se realiza con todas las imágenes.

e) A continuación se itera también sobre las tablas para obtener su contenido. Se extrae el texto de cada celda de la tabla. Si una celda contiene demasiado texto, este se divide en partes más pequeñas para hacerlo más manejable. La partición se hace de forma lógica, buscando, por ejemplo, los puntos (.) para asegurar que las divisiones no interrumpan el flujo del texto.

   Los datos extraídos se organizan en un formato estructurado, creando representaciones textuales de las tablas que incluyen encabezados y filas de datos. Se utiliza un diccionario para almacenar los datos de la tabla, con las claves correspondientes a los encabezados de las columnas y los valores como el contenido de las celdas. Si una fila contiene más celdas de las esperadas, se ajusta para asegurar que todos los datos se representen correctamente.

Este es un ejemplo de cómo se quedaría una tabla una vez scrapeada:

Semana	Ventas	Crecimiento
Semana 1	25	-
Semana 2	65	40
Semana 3	67	2
Semana 4	90	23

Figura 3.2: Tabla en el .docx

```
| Semana   | Ventas | Crecimiento |
---------------------------------
| Semana 1 | 25     | -           |
---------------------------------
| Semana 2 | 65     | 40          |
---------------------------------
| Semana 3 | 67     | 2           |
---------------------------------
| Semana 4 | 90     | 23          |
---------------------------------
```

Figura 3.3: Tabla después del scraping

f) Se itera sobre la lista que contiene las secciones, concatenando su contenido en el orden en que fueron encontradas. Esto asegura que el documento resultante mantenga la coherencia y la estructura original. El resultado final se devuelve como una cadena de texto que contiene todo el contenido extraído y organizado del documento original. También se proporciona la lista con los diccionarios de las secciones para usarla posteriormente.

4. **Prompt automático**

   Un prompt es una instrucción o conjunto de instrucciones proporcionadas a un modelo de lenguaje para guiar su generación de texto. En el contexto de los modelos de lenguaje avanzados, un prompt puede incluir información contextual, preguntas, o directrices específicas que ayudan al modelo a entender el tipo de respuesta que se espera.

   Por ejemplo, supongamos que tenemos la siguiente información:

   - Pregunta: `¿Cuál es la capital de Francia?`

Un ejemplo de prompt podría ser el siguiente:

```
Responde a la siguiente pregunta con información precisa
y detallada:
Pregunta: ¿Cuál es la capital de Francia?
Recuerda ser educado y amable.
```

Este prompt se pasa al modelo de lenguaje, como GPT-4, que lo utiliza para generar una respuesta precisa. Por ejemplo, la respuesta podría ser:

```
La capital de Francia es París.
```

En el contexto de este trabajo, utilizando un prompt se especifica a un modelo de lenguaje (ChatGPT-4) que busque en el contenido del documento el nombre de la empresa y su descripción y esta información se utiliza para actualizar los servicios del sistema, permitiendo al LLM responder preguntas de los usuarios de manera más informada y precisa.

5. **Chunking del texto**

   El paso siguiente en el proceso es el chunking del texto, que consiste en dividir el contenido en trozos más pequeños. Sin embargo, los detalles específicos de este paso se abordarán en una sección posterior.

6. **Almacenamiento del texto**

   A continuación, el texto se guarda tanto en Amazon S3, que es el almacenamiento de AWS, como en Pinecone.

   S3 se utiliza para almacenar de manera segura y escalable los documentos originales y sus versiones chunkerizadas.

   Pinecone es una base de datos vectorizada especializada en almacenar y gestionar embeddings. Un embedding es una representación vectorial de un texto que captura su significado semántico, lo que permite comparar fragmentos de texto de manera eficiente.

   Para cada chunk del texto, se genera un embedding utilizando el modelo `text-embedding-3-small` de OpenAI. Estos embeddings son vectores

numéricos que representan el contenido del texto de manera que textos con significados similares tendrán embeddings cercanos entre sí en el espacio vectorial.

El almacenamiento en Pinecone permite realizar búsquedas eficientes basadas en similitud de coseno. Cuando se hace una pregunta, se genera un embedding de la pregunta y se compara con los embeddings almacenados en Pinecone. El sistema recupera los fragmentos de texto cuyos embeddings son más similares al embedding de la pregunta, proporcionando respuestas relevantes y contextualmente apropiadas.

Una vez guardada la información en los lugares correspondientes se borra la estructura de carpetas que se había creado en *tmp*, para no saturar el almacenamiento.

7. **Resumen del documento**

   Finalmente se realiza un resumen del documento completo. Para llevar a cabo esta tarea, se utilizan los asistentes de OpenAI, que permiten subir el documento y, mediante un prompt específico, solicitar un resumen del contenido.

   El proceso comienza con la carga del documento en el asistente de OpenAI. A través de un prompt detallado, se le pide al modelo que genere un resumen conciso pero informativo del documento, detallando los procesos si hay alguno. El modelo procesa el documento y genera un resumen que captura la esencia del contenido, proporcionando una visión general rápida y útil del documento original.

   Este resumen también se sube al almacenamiento de AWS, S3, y a Pinecone.

Archivos TXT

Los archivos de texto (TXT) son un formato plano, lo que significa que no contienen imágenes, tablas ni ningún tipo de formato, solo caracteres de texto. Esta simplicidad facilita la extracción del texto, ya que no es necesario manejar elementos adicionales o complejos.

Aunque no es tan común trabajar con archivos de texto plano en comparación con otros formatos como DOCX o PDF, estos archivos todavía existen y se utilizan

para almacenar información.

El proceso de scraping para archivos de texto es muy similar al utilizado para los archivos DOCX, aunque con algunas simplificaciones debido a la naturaleza del formato plano. Los pasos a seguir son los siguientes:

1. **Inicio de la función**

   Se inicializan variables clave como service_id (que identifica la cuenta del usuario), la ruta en la que está almacenado el archivo en S3 y el document_id (identificador único del documento).

2. **Descarga del documento desde S3**

   El archivo de texto se descarga desde el almacenamiento de S3 a una ruta local en */tmp*. De esta manera el documento está disponible localmente para el procesamiento.

3. **Extracción del texto**

   La extracción del texto de un archivo TXT es directa y no requiere bibliotecas específicas. Esto se logra simplemente abriendo el archivo en modo lectura con `with open('ruta_al_archivo.txt', 'r') as file` y leyendo su contenido. El texto leído se guarda en una variable, lo que permite manipularlo y procesarlo en los siguientes pasos.

4. **Chunking del texto**

   Una vez extraído el texto, se procede a dividirlo en trozos más pequeños mediante el proceso de chunkerización. Este paso es crucial para manejar grandes volúmenes de texto, ya que permite que los fragmentos de texto sean más manejables y puedan ser analizados individualmente. Los detalles de este proceso se explicarán con más profundidad en una sección posterior.

5. **Almacenamiento del texto**

   Después del chunking, tanto el texto completo como los chunks resultantes se almacenan en S3. Este almacenamiento asegura la persistencia y disponibilidad del contenido. Adicionalmente, para cada chunk del texto, se generan embeddings utilizando el modelo `text-embedding-3-small` de OpenAI y se almacenan en Pinecone, una base de datos vectorial. De esta manera, cuando se hace una pregunta, se puede recuperar el fragmento de texto más relevante basado en la proximidad semántica.

A diferencia de otros formatos de documentos, no se genera un resumen para los archivos de texto plano. La razón principal es que este tipo de documento no

es muy común y, generalmente, no se justifica el esfuerzo adicional de resumir su contenido debido a su simplicidad inherente. La mayoría de los archivos de texto plano contienen información directa y concisa, lo que hace que el resumen sea menos necesario.

Documentos PDF

Los documentos PDF son probablemente el tipo de documento más común en el ámbito digital. Sin embargo, también son los más complicados de scrapear debido a la gran variedad de formatos y estructuras que pueden presentar. Por ejemplo, un documento PDF puede estar escrito en columnas o filas, contener tablas complejas, imágenes integradas, gráficos, formularios, y en algunos casos, pueden ser simplemente documentos escaneados que contienen texto como imágenes en lugar de texto seleccionable.

La presencia de estos elementos diversos y la falta de una estructura estándar clara dentro de los documentos PDF hacen que el scraping de este tipo de archivos sea especialmente complicado. Cada PDF puede requerir una estrategia de extracción diferente, y es posible que se necesiten múltiples herramientas y enfoques para manejar adecuadamente todos los tipos de contenido. Esto puede incluir el uso de herramientas de OCR (Optical Character Recognition) para PDFs escaneados, algoritmos específicos para detectar y extraer tablas, o técnicas avanzadas para manejar contenido en múltiples columnas.

Debido a esta complejidad, desarrollar una estrategia escalable que pueda utilizarse para todos los PDFs es una tarea compleja. Requiere un enfoque flexible y robusto que pueda adaptarse a las distintas formas en que la información puede presentarse en un PDF, garantizando que se pueda extraer y utilizar de manera efectiva en diferentes contextos.

Además, este proceso es más lento en comparación con la extracción de información de archivos TXT y DOCX. Estos permiten una extracción rápida de información debido a sus formatos internos más sencillos y estructurados, lo que facilita la lectura directa del contenido. Sin embargo, los PDFs pueden ser más densos y variados en su estructura interna, lo que ralentiza significativamente el proceso de extracción de información.

El proceso se ha dividido en tres Lambdas debido a que AWS impone una limitación de tiempo de ejecución de 15 minutos, lo cual puede ser insuficiente para procesar PDFs extensos o complejos. Para manejar esta restricción, el documento se divide inicialmente en lotes de 10 páginas. Cada lote es procesado de manera

independiente por instancias separadas de la función de parseo. Este enfoque no solo asegura que cada función se ejecute dentro del límite de tiempo, sino que también permite el procesamiento en paralelo, aumentando considerablemente la eficiencia del sistema.

La primera Lambda, *Load PDF*, se encarga de identificar y procesar información clave dentro del documentos PDF. Tras este procesamiento inicial se inicia la segunda Lambda, *Parse PDF*, que es ejecutada múltiples veces en paralelo, cada una procesando un lote distinto de 10 páginas. Esta función analiza las páginas y extrae la información relevante, almacenando los resultados de manera temporal. Finalmente, la Lambda *Finish Parse* se activa una vez completados todos los procesos de parseo. Esta función unifica los resultados de todos los lotes, consolidando la información en un solo conjunto de datos coherente y final.

El proceso se puede visualizar en la Figura 3.4

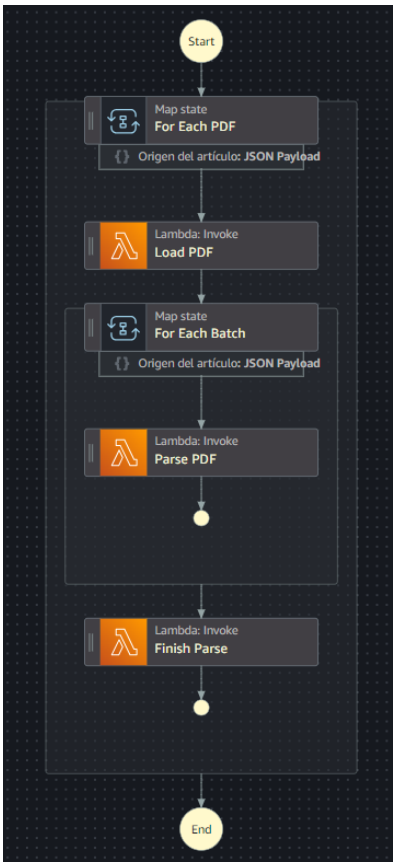

Figura 3.4: Proceso de scrapeo de PDFs. Step Function

Lambda de carga

Esta función está diseñado para identificar y procesar información clave dentro de documentos PDF, tales como el índice, títulos de secciones, y contenido relevante, así como para eliminar ruido (texto irrelevante o repetitivo).

El flujo desarrollado es el siguiente

1. **Descarga del documento desde S3**

   El proceso se inicia con la descarga del archivo PDF desde un bucket de S3 (un contenedor de objetos, de archivos), utilizando las credenciales y el cliente S3 proporcionado por boto3. Boto3 es el SDK (Software Development Kit) de AWS para Python y permite la integración y el uso de servicios AWS. Es decir, consiste en un conjunto de funciones de Python específicas para interactuar con los servicios web de Amazon.[13]. De esta manera se facilita la descarga del archivo PDF para que esté disponible localmente para su procesamiento.

2. **Búsqueda de ruido**

   A continuación se lleva a cabo una identificación y eliminación de dos tipos de ruido: simple y avanzado. El ruido simple incluye elementos repetitivos y fácilmente identificables que no contribuyen significativamente al contenido principal, como encabezados, pies de página y numeración de páginas. Para su detección y eliminación se utiliza la biblioteca *PyMuPDF*.

   Por otro lado, el ruido avanzado abarca bloques de texto que, aunque no están presentes en todas las páginas, se repiten en múltiples secciones del documento. Esto incluye frases o párrafos que aparecen en diferentes contextos, así como bloques de texto legal o citas que, a pesar de ser relevantes, resultan repetitivos. La identificación de este tipo de ruido requiere un análisis más detallado y complejo, donde se utilizan técnicas avanzadas para detectar patrones de repetición menos obvios a través de diversas páginas.

3. **Limpieza del texto y búsqueda del índice**

   PyMuPDF es nuevamente utilizado para analizar el texto de cada página del PDF. Una vez que se extrae el texto del documento, es esencial limpiarlo de elementos que no aportan contenido. Esto incluye eliminar referencias a números de página y el ruido que no contribuye nada. Primero se busca y elimina cualquier referencia a páginas, tales como *página 12* o *pag. 34* y a continuación se elimina del texto procesado las frases identificadas como

ruido en el punto anterior.

Con el texto limpio, el siguiente paso es la búsqueda del índice y para ello se buscan palabras clave como *índice*, *sumario* o *contenido*. Cuando se encuentra una de estas palabras se elimina del texto, ya que no proporciona información relevante, y se almacena el índice en caso de que se necesite más adelante.

4. **Búsqueda de títulos de secciones**

   Si se encuentra un índice en el punto anterior, se procede a extraer los títulos de las secciones del documento. Estos títulos son fundamentales para estructurar el contenido y facilitar su análisis posterior. En este punto, se utiliza la biblioteca *re* de Python para procesar el índice y extraer los títulos, eliminando números y puntos para normalizar los datos.

5. **Lectura del texto del PDF por páginas**

   Se lee el texto del documento página por página, utilizando *pdfplumber*, y se procesa para mejorar su legibilidad y formato. Primero se lee el texto de una página específica del PDF, luego se limpia eliminando espacios y saltos de línea innecesarios, específicamente quitando líneas que solo contienen espacios y reduciendo múltiples saltos de línea consecutivos a un máximo de dos. Finalmente, ajusta el texto juntando líneas que erróneamente fueron separadas, creando un flujo más natural del texto.

6. **Identificación del título del PDF**

   Utilizando la API de OpenAI, se analiza el texto de la primera página del PDF para identificar el título del documento. Para ello se utiliza un prompt diseñado para extraer el título, o indicar si no se encontró un título claro.

7. **Subida de resultados a S3**

   Finalmente, los resultados procesados, incluyendo las secciones, el ruido y el título del documento se suben de nuevo a un bucket de S3 para tenerlos disponibles en caso de necesitarlos para un uso posterior.

Lambda de scrapeo

El siguiente paso es extraer el contenido del documento, y para ello hay una segunda Lambda. El proceso vuelve a iniciarse descargando el PDF desde el bucket de S3 en el que está almacenado, así como la información que se obtuvo en el

punto anterior: el título del documento, el ruido (información no relevante) y las secciones. Se obtiene el contenido página a página.

Es importante mencionar que antes de empezar a extraer información de cada página del PDF, primero se verifica si la página está en orientación vertical u horizontal. Si una página está horizontal, se trata como si fuera una diapositiva de PowerPoint o una imagen, porque su contenido suele ser más gráfico. Esto requiere un método de procesamiento diferente para asegurarnos de extraer toda la información correctamente.

A partir de aquí los pasos que se siguen son los siguientes:

1. **Orientación horizontal**

   Si la orientación es horizontal se llama a una función que convierte la página en una imagen con extensión *.png* y se almacena en un directorio local para poder acceder a ella rápidamente. A continuación se hace una llamada a Claude, un LLM desarrollado por Anthropic [14], para que procese la imagen y extraiga el contenido útil. El prompt enviado es el mismo que en DOCX se utiliza para procesar las imágenes:

   > *You work for a company that writes documents. You have to scrap the text in the image and return it in spanish. Do not include information that is not in the image, only the text. Write it in a logical order. Note that the text is going to be divided into chunks. If there is no text in the image, return an empty string. If the image contains no conventional text but is a flowchart, explain the flowchart in a logical order, detailing the information in it. It is very important that you answer in spanish*

2. **Extraer texto y tablas si la orientación es vertical**

   Por el contrario, si la página es vertical el procedimiento es más complejo. En primer lugar se extrae todo el texto de la página utilizando la librería *PyMuPDF*. Luego se extraen las tablas utilizando otra librería llamada *camelot* y se almacenan como .csv en un directorio local.

3. **Extraer texto de las imágenes**

   A continuación se identifican las imágenes y se procesan como ya se ha explicado anteriormente, pasándoselas a Claude para que extraiga el texto que hay en ellas o las describa en caso de que no haya texto y sea, por ejemplo, un gráfico. La respuesta del modelo, ya sea el texto o la descripción de la imagen, se guarda en una variable aparte y no se integra aún con el texto

principal.

Originalmente, el proceso se llevaba a cabo utilizando el reconocimiento óptico de caracteres (OCR), que es la conversión de imágenes de texto en texto legible por máquinas, facilitando la extracción y procesamiento de datos de documentos escaneados [15]. Sin embargo, una vez que las tecnologías evolucionaron y el análisis mediante modelos de lenguaje se hizo posible, se cambió de estrategia, optando por esta nueva metodología por ser mucho más efectiva y precisa.

Las imágenes se guardan en un directorio local.

4. **Eliminar ruido**

   Una vez que completada la extracción del texto, el siguiente paso es la eliminación del ruido, es decir, de aquel contenido que no aporta valor al análisis, como pueden ser los encabezados y los pies de página. Al inicio del proceso se descargó un archivo desde S3 que contiene las líneas de texto que forman ese ruido. Luego, al procesar el texto extraído, comparamos cada línea con las líneas de ruido identificadas previamente y eliminamos aquellas que coincidan.

5. **Incorporar texto de las imágenes**

   El siguiente paso en el proceso consiste en integrar el texto extraído de las imágenes con el resto del documento. Para lograr esto, hay que identificar la posición precisa de cada imagen dentro del texto principal.

   Durante la extracción de las imágenes, no solo se captura el texto, sino que también se almacena en un diccionario junto con la última frase antes de la imagen. Posteriormente, al procesar el documento, recorremos el texto en busca de estas frases anteriores. Al localizarlas, insertamos en ese punto el texto correspondiente a cada imagen.

   Por ejemplo, para esta imagen:

Figura 3.5: Imagen en un PDF

Se guardaría un diccionario tipo:

{*"linea_anterior": "Por lo tanto, hacer deporte tiene muchos beneficios", "texto_img": "Beneficios del deporte en el cerebro: Disminuye los niveles de estrés y ansiedad. La práctica regular de actividad física contribuye a reducir los niveles de estrés y ansiedad, promoviendo un estado de bienestar mental y emocional..."*}

Luego se recorrería todo el texto que se ha extraído, y cuando se encuentre *linea_anterior*, se insertaría *texto_img* a continuación, manteniendo el orden del documento original.

Si no se logra determinar la posición de la imagen en el texto, el texto extraído de la imagen se añadirá al final del documento.

6. **Guardar resultados en S3**

   Una vez extraída y ordenada toda la información de la página, se suben a S3 tanto las tablas como el texto. A continuación se elimina todo lo que se había descargado en el directorio local, para no saturar el almacenamiento.

Lambda de finalización

El último paso es unir los resultados de cada página, lo cual se realiza en una nueva función Lambda. El proceso que se sigue es el siguiente:

1. **Descarga del texto de cada página**

   La función comienza descargando el texto correspondiente a cada página del documento, que está guardado en S3, y almacenándolo localmente en el directorio temporal */tmp*. Una vez que todas las páginas están descargadas, el contenido de estas se concatena en una única cadena de texto.

2. **Chunking del texto**

   A continuación se realiza la segmentación del texto, o chunking, para poder manejar los datos de manera más eficiente. Los detalles específicos de este proceso se explicarán más adelante en una sección dedicada, pero en general, el objetivo es maximizar la coherencia del contenido dentro de cada chunk mientras se mantiene una distribución manejable del texto.

3. **Guardado del texto chunkerizado en S3**

   Una vez que el texto ha sido adecuadamente dividido, cada chunk se guarda de vuelta en el almacenamiento S3 para poder consultarlo o usarlo en procesos posteriores.

4. **Insercción en Pinecone**

   Con los chunks de texto ya almacenados en S3, el siguiente paso es la inserción de estos en Pinecone, una base de datos especializada en el manejo de vectores. Antes de la inserción, cada chunk de texto es transformado en un vector utilizando los embeddings de OpenAI, lo que permite una representación numérica del texto.

5. **Borrado de archivos temporales**

   Finalmente, una vez completados todos los pasos anteriores, se procede a limpiar el espacio de trabajo eliminando los archivos temporales almacenados en el directorio */tmp*. Este paso es importante para asegurar que no hay datos que ocupen espacio innecesariamente.

3.1.2. Recolección de datos desde la web

Un sitio muy común donde se almacena información importante es la página web de una empresa. Estas páginas no solo ofrecen datos directos sobre la organización, sino que también contienen enlaces a otras páginas que a su vez poseen información relevante. Por lo tanto, para realizar una extracción eficiente de todos estos datos, es necesario diseñar un sistema que sea capaz de obtener todas las URLs relacionadas con la empresa, de modo que el cliente no tenga que introducir cada una de ellas manualmente, si no que a partir de una se obtienen el resto de

enlaces. A este tipo de sistema se le denomina sistema de crawling.

Se define el Web Crawling como el proceso mediante el cual un programa navega sistemáticamente por Internet, explorando sitios web y siguiendo enlaces con el objetivo de recolectar contenido web en otro sistema. Todos los motores de búsqueda utilizan estos bots, conocidos como crawlers, para indexar los diferentes sitios web en sus sistemas de búsqueda [16].

Hay que hacer una distinción entre una URL y un sitemap, ya que la forma de procesarlos será diferente. Una URL es la dirección específica para acceder a un recurso individual en la web, como una página o imagen, y un sitemap es un archivo que enumera todas las URLs importantes de un sitio web, mostrando la estructura del sitio y las relaciones entre sus páginas [17].

3.1.3. Sistema de crawling

Los pasos que se siguen son los siguientes:

1. **Inicialización y preparación de la URL**

   Al recibir el enlace, la función crawler primero verifica y ajusta la URL añadiendo *https://* si es necesario. Esta normalización es muy importante para garantizar que las solicitudes subsiguientes a la web sean válidas.

2. **Detección y manejo de sitemaps**

   Si la palabra *sitemap* aparece en la URL, la función intenta utilizar una clase predefinida para extraer todas las URLs del sitemap.

   Si falla la extracción, se añade *www.* al inicio del dominio y se repite el proceso. Esto se hace porque muchos errores de acceso suelen deberse a la falta de dicho prefijo.

3. **Análisis de URLs en páginas web**

   Si el enlace proporcionado no contiene la palabra *sitemap*, se procede a analizar la URL.

   Primero se verifica si la URL es la página de inicio (*home*) del sitio web, y en caso de que sea así, se obtienen todas las URLs indexadas a esta página analizando el HTML. Comenzar el rastreo desde el *home* es estratégico para asegurar una exploración exhaustiva, pues típicamente contiene enlaces a

todas las secciones principales del sitio.

Posteriormente, se busca si existe un archivo *index.html*, que es la página principal o de inicio de cualquier sitio web y que actúa como un punto central que contiene enlaces a otras páginas importantes. Para hacer esto se comprueba si en la URL está presente la cadena de texto *index.html*. Si se encuentra el archivo se extraen las URLs asociadas a él.

Finalmente se analiza el HTML de la página para extraer todas las URLs anidadas, utilizando la biblioteca BeautifulSoup. Si este proceso falla, se añade *www.* a la URL y se intenta nuevamente.

4. **Gestión y filtrado de URLs**

   Las URLs recolectadas se suben a un archivo de texto en un bucket de S3 y, antes de proceder con el scraping, el sistema consulta si ya existe el archivo en S3 para verificar si la página ya ha sido scrapeada previamente, evitando así redundancias en el proceso. Después de la verificación, se hace una selección final de las URLs que necesitan ser scrapeadas.

5. **Filtrado y priorización final**

   Las URLs se filtran para eliminar duplicados y se descartan las que superan cierta longitud o no cumplen con criterios específicos de relevancia, por ejemplo si no incluyen el dominio de la web.

   Finalmente se ordenan las URLs, priorizándose aquellas que son más próximas a la URL original. De esta manera se garantiza que el sistema de crawling sea lo más eficiente posible.

3.1.4. Scraping de las URLs

Una vez que se han identificado las URLs de interés, el siguiente paso en el proceso es realizar el scraping de dichas URLs para extraer la información relevante. Este proceso se ejecuta en una función AWS Lambda, que permite manejar las operaciones de extracción de datos de manera eficiente y escalable. A continuación, se detallan los pasos que se siguen durante el scrapeo:

1. **Inicialización de variables**

   La función comienza recibiendo un evento que contiene los parámetros necesarios para el proceso. Este evento incluye el document_id, doc_name (que

en este contexto es la URL a procesar), namespace y otros identificadores relacionados con servicios externos.

2. **Scrapeo del texto**

   El proceso de scraping del texto se realiza en varias etapas:

   - **Extracción de contenido**: para la extracción efectiva de contenido de páginas web y evitar bloqueos por parte de los servidores, se utiliza la librería *requests*. Esta herramienta permite realizar peticiones HTTP a las URLs de interés, configurando headers específicos que simulan la actividad de un navegador web.

     Una vez obtenida la respuesta del servidor, se verifica el tipo de contenido mediante el análisis del *Content-Type*:

     - Si la respuesta es un documento PDF (*application/pdf*), se procede con un método específico para la extracción de texto de documentos PDF. Así se garantiza que se manejan adecuadamente los formatos que no son HTML.
     - Si el contenido no es un PDF, entonces se utiliza Langchain, un marco de trabajo de código abierto para crear aplicaciones basadas en modelos de lenguaje de gran tamaño (LLM) [18]. Langchain proporciona herramientas avanzadas para el análisis y procesamiento del contenido web, utilizando técnicas para parsear el HTML y extraer su texto.

   - **Identificación de ruido**: se procede a eliminar el ruido, entendido como aquel texto que aparece repetidamente en múltiples URLs, como pueden ser las cabeceras o pies de página.
   - **Eliminación de duplicados**: una vez obtenido el texto, se eliminan las líneas duplicadas para depurar el contenido y facilitar el análisis posterior.

3. **Búsqueda del título en los metadatos**

   Con el objetivo de organizar y catalogar mejor el contenido extraído, se busca el título de la página web dentro de los metadatos (específicamente en la etiqueta <*title*> del HTML). Este título es importante para entender el contexto del contenido y se utiliza posteriormente para el almacenamiento estructurado del texto.

4. **Chunking del texto**

   Este paso implica dividir el texto en segmentos o más manejables. Este proceso es fundamental para el tratamiento posterior de los datos y se abordará en detalle en una sección posterior.

5. **Almacenamiento de datos**

   Finalmente, el texto extraído se almacena en varios formatos y ubicaciones:

   - **Amazon S3**: se guarda el texto crudo y el texto dividido en chunks. Además, se actualiza un archivo de texto que mantiene un registro de todas las URLs que han sido procesadas, lo cual es útil para evitar repeticiones en futuras operaciones de scraping.
   - **Pinecone**: se utiliza para indexar y almacenar los embeddings de los textos para su posterior recuperación y uso en el LLM. Esto se realiza después de obtener representaciones vectoriales (embeddings) de los chunks de texto.

3.2. Técnicas de chunking para la segmentación de datos

3.2.1. Definición y alcance del chunking

El chunking es una técnica esencial para mejorar el rendimiento de los Modelos de Lenguaje de Gran Tamaño (LLMs), que enfrentan el desafío de una *ventana de contexto* limitada, es decir, solo pueden analizar y razonar con una cantidad limitada de texto a la vez. Esta limitación puede reducir su eficacia en tareas que requieren una comprensión contextual amplia, como la búsqueda semántica o el resumen de documentos. El chunking aborda este problema al dividir textos extensos en segmentos más pequeños y manejables, permitiendo que el LLM procese la información de manera óptima.

La elección del tamaño de los chunks tiene un impacto significativo en la relevancia de los resultados obtenidos. De cada chunk se hace un embedding, una representación vectorial de su contenido, y se ha comprobado que los chunks pequeños ayudan al modelo a concentrarse en el significado específico de cada oración, aunque chunks demasiado cortos pueden llevar a una falta de contexto suficiente. Por otro lado, los chunks más grandes, como párrafos completos o documentos, permiten al modelo capturar un contexto más amplio, aunque esto puede generalizar demasiado los embeddings y reducir el enfoque en el significado de las oraciones

individuales. [19].

El uso de chunking no se limita solo a mejorar los resultados de búsqueda, sino que también se extiende a aplicaciones como los agentes conversacionales, donde ayuda a los LLMs a entender mejor la intención del usuario al procesar mensajes entrantes en unidades coherentes y manejables. Así, los LLMs pueden generar conversaciones más naturales y efectivas. Además, los métodos de chunking pueden variar desde la división fija de tamaño hasta el chunking consciente del contenido, donde los segmentos se crean basados en la estructura semántica del texto [20].

En este proyecto, se ha desarrollado una estrategia heurística para la división del texto, que optimiza el uso de chunking en LLMs adaptándose dinámicamente a las características del contenido procesado.

3.2.2. Implementación del chunking en diferentes formatos

Para implementar el chunking en los formatos TXT, PDFs y en URLs se utiliza la misma estrategia. En cambio, para los archivos DOCX se hace una pequeña modificación para ajustarse a sus características específicas, aunque la esencia del proceso sigue siendo la misma.

Estrategia para PDF, TXT y URL

Los pasos que se siguen son los siguientes:

1. **Inicialización de variables**

   En primer lugar se inicializan las variables que determinan el tamaño mínimo y máximo del chunk, y un umbral de similitud entre dos frases.

2. **Creación de funciones**

   Se crean dos funciones que van a ayudar en el proceso de segmentar el texto:

   - Identificación de títulos: mediante una función específica, el sistema determina si una línea del texto es un título. Esta función considera título a cualquier línea escrita completamente en mayúsculas o aquellas líneas cortas (menos de 5 palabras) donde cada palabra empieza con mayúscula.
   - Cálculo de similitud entre oraciones: utilizando la biblioteca *spaCy*, esta función mide cuán similares son dos oraciones basándose en sus características lingüísticas. Esta similitud es crucial para decidir si segmentar o no el texto en ese punto.

3. **Proceso de división del texto**

   Hay dos variables importantes en este proceso, la variable `chunks`, que es una lista que almacena los segmentos de texto finales, aquellos que cumplen con los criterios de tamaño establecidos y están listos para ser procesados, y la variable `current_chunk`, que es una lista temporal que acumula líneas de texto a medida que se procesa el documento, ayudando a construir gradualmente los segmentos hasta que están listos para ser evaluados y trasladados a `chunks`.

   Teniendo esto en cuenta se sigue el siguiente proceso para dividir el texto:

   a) En primer lugar, el texto completo se divide en líneas individuales usando el comando `splitlines()` y cada línea se limpia de espacios extras al principio y al final con `strip()`.

   b) Se comprueba si la línea que se esta evaluando es un título, usando la función definida para ello, y si ya existe texto acumulado en `current_chunk`. En caso de que se cumplan estas condiciones se comprueba si este texto acumulado (sin incluir el título) supera el tamaño mínimo para ser considerado un chunk. Si es así, se añade a la lista de chunks (`chunks`) y se reinicia `current_chunk`.

      El objetivo de este paso es asegurar que, cuando sea posible, un título y el contenido que le sigue se agrupen en un chunk independiente del texto anterior. Esto se hace porque los títulos suelen introducir nuevas secciones o temas que probablemente no estén directamente relacionados con la información previa y, al separarlos en diferentes chunks se mejora el embedding, permitiendo que cada segmento represente de manera más precisa y relevante su contenido específico.

   c) Independientemente de si es título o no, la línea actual se agrega a `current_chunk`. Luego, se verifica si el texto acumulado hasta el momento excede el tamaño máximo permitido.

   d) Si `current_chunk` no supera el tamaño máximo, se sigue evaluando la siguiente línea. Se comprueba si es un título o no y se agrega a `current_chunk`. Después se vuelve a comprobar si supera el límite establecido.

   e) Si `current_chunk` supera el tamaño máximo, el texto se subdivide en oraciones usando expresiones regulares que detectan el final de las frases. Se cuenta en número de oraciones que hay y, en caso de que haya una sola, se incluye tal cual como nuevo chunk para evitar perder su

integridad. De esta manera se puede exceder el límite de tamaño pero se obtiene flexibilidad para evitar fragmentar información importante que podría perderse o malinterpretarse si se dividiera.

En casos donde `current_chunk` contiene varias oraciones, se evalúa la similitud entre oraciones consecutivas para encontrar un punto de corte adecuado. La evaluación se centra especialmente en las dos últimas oraciones acumuladas; se calcula su similitud y, si es baja (por debajo del umbral definido), esto indica un cambio potencial en el tema, haciendo adecuada su separación. Si las oraciones son muy similares, se mantienen juntas para conservar el contexto y la coherencia del texto, buscando otro punto de corte que no supere el máximo tamaño permitido pero que, al mismo tiempo, no divida indebidamente el contenido.

f) Es posible que no se encuentre un punto para realizar la división que cumpla con los requisitos de tamaño máximo y de similitud. Esto ocurre cuando, incluso después de evaluar todas las combinaciones posibles de oraciones consecutivas, ninguna división resulta en un segmento que sea simultáneamente inferior al tamaño máximo y que tenga una baja similitud entre las oraciones que lo limitan.

En ese caso se realiza una evaluación inversa de la longitud de las oraciones: se itera desde el final hacia el principio del `current_chunk` y, al encontrar la última oración que no excede el tamaño máximo permitido, se corta el texto en ese punto. Además, se registra en una variable (`relations`) que existe una relación entre este chunk y el siguiente, indicando que están conectados.

4. **Final del texto**

   Cuando solo queda una frase se evalúa su longitud. Si supera el mínimo establecido se agrega a la lista de chunks como un segmento independiente, y si no lo supera, se añade al último chunk que había en la lista.

5. **Metadatos**

   Cada chunk generado se asocia con un conjunto de metadatos que contienen información clave sobre el texto. Estos metadatos pueden incluir, por ejemplo, el índice de inicio del chunk dentro del texto original, lo que facilita la referencia precisa al contenido.

   Más importante aún, cuando un chunk está relacionado con el siguiente, información almacenada en la variable `relations`, esta relación se indica

explícitamente en los metadatos. De esta manera cabe la posibilidad de trabajar con los chunks relacionados en el futuro, permitiendo procesarlos como una unidad cohesiva a posteriori.

Estrategia para DOCX

El proceso que se sigue es idéntico al explicado en el punto anterior pero con una pequeña modificación para manejar las tablas. En este nuevo enfoque se identifican las tablas en el texto y se tratan como segmentos independientes para la creación de chunks. Es decir, el contenido de la tabla no se mezcla con el resto del texto en un mismo chunk.

Para asegurar la coherencia y el orden dentro del documento, se almacena la última frase antes de la tabla. Esta frase se utiliza para integrar correctamente el chunk de la tabla en su lugar correspondiente en el documento, facilitando así una estructura más clara y ordenada de los chunks.

Además, al principio de cada chunk se agrega el nombre de la sección del documento a la que pertenece, lo que enriquece la información disponible cuando se busca el embedding más similar durante una consulta. Esto facilita la precisión y relevancia de los resultados al relacionar el contenido del chunk con su contexto específico en el documento.

3.2.3. Elección parámetros

Como ya se ha mencionado anteriormente, el tamaño del chunk, especialmente el tamaño máximo, es un factor muy importante debido a su impacto directo en el rendimiento de los LLMs.

Para determinar el valor óptimo de este parámetro, se han llevado a cabo pruebas usando un método que consiste en seleccionar un documento/página web como base, formular preguntas relacionadas con su contenido y registrar tanto la pregunta como la respuesta correcta (*ground truth*). Posteriormente, estas mismas preguntas se plantean al LLM, y las respuestas generadas se comparan con las respuestas correctas. La evaluación de las respuestas del LLM se realiza utilizando la librería *Llama-Index*, que permite asignar una puntuación entre 0 y 1 basada en la precisión y relevancia de las respuestas del modelo en relación con el *ground truth*.

A continuación, se presentarán los resultados de este procedimiento aplicado a dos PDFs diferentes.

Tamaño del chunk (caracteres)	PDF 1	PDF 2
128	0.8723	0.7903
256	0.8831	0.8628
384	0.9013	0.8797
512	0.8928	0.8942
640	0.8986	0.8919
768	0.9028	0.9028
896	0.8896	0.905
1024	0.8849	0.8943

Cuadro 3.1: Resultados de la precisión del LLM para diferentes tamaños de chunks

Estas pruebas muestran que el tamaño óptimo varía según el documento específico. Para el PDF 1, el tamaño de chunk que ofreció los mejores resultados fue de 768 caracteres, mientras que para el PDF 2 fue de 896. Estos resultados demuestran la dificultad de establecer una regla general que se aplique de manera uniforme a todos los tipos de documentos, dado que diferentes contenidos y estructuras pueden influir significativamente en la efectividad del modelo.

A lo largo de múltiples pruebas, que incluyen muchos documentos de diferente formato, el tamaño de chunk que finalmente se estableció como el más equilibrado y efectivo para un uso más general fue de 512 caracteres. Este tamaño proporciona una buena compensación entre la precisión y la gestión del contexto dentro de las limitaciones del modelo, demostrando ser eficaz en una gama más amplia de documentos.

3.3. Integración con el LLM

La integración de los procesos de scrapeo con el LLM es un flujo secuencial y automatizado que se detalla a continuación:

1. **Subida del documento**

   El cliente inicialmente sube el documento deseado a través de una interfaz de usuario en la página web. Este puede ser un documento en varios formatos, como PDF, DOCX o TXT, o una URL.

2. **Almacenamiento en S3**

   Una vez subido, el documento se almacena automáticamente en un bucket específico dentro de Amazon S3. Si es una URL se hace el crawling y se

obtienen todas las URLs indexadas a la URL especificada. Se sube a S3 un documento de texto con los enlaces que se van a scrapear.

3. **Detección y procesamiento del documento**

   Al detectar un nuevo documento en S3, un sistema automatizado identifica el tipo de archivo basado en su extensión y, según el tipo de documento identificado, se invoca una función AWS Lambda específica. Esta función está encargada de extraer el texto del documento según se ha explicado.

4. **Almacenamiento de texto y procesamiento posterior**

   El texto extraído se guarda nuevamente en S3 y se procesa para realizar el chunking como se describió previamente. Luego, se generan embeddings para cada chunk, que son representaciones vectoriales que capturan su significado semántico.

   Los embeddings generados se suben a Pinecone, una base de datos especializada en manejar datos vectorizados, facilitando búsquedas eficientes basadas en similitud semántica.

5. **Retrieval-Augmented Generation (RAG)**

   Cuando se plantea una pregunta al LLM, se utiliza una técnica conocida como Retrieval-Augmented Generation (RAG) para responder. El RAG combina la generación de texto con un sistema de recuperación de información.

   El sistema primero busca en la base de datos vectorial (Pinecone) conceptos y datos que parezcan similares a la pregunta formulada. Extrae estos datos, los reformula y genera una respuesta personalizada a la pregunta realizada. Esto convierte al RAG en una herramienta poderosa para las empresas que buscan aprovechar sus repositorios de datos existentes para mejorar las respuestas de su modelo de lenguaje [21].

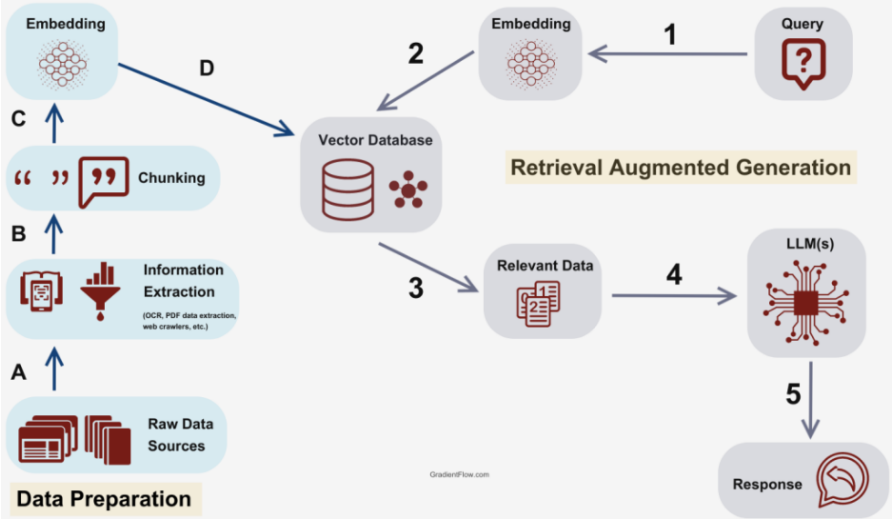

Figura 3.6: RAG [22]

La Figura 3.6 muestra el proceso que se lleva a cabo:

- Al recibir una pregunta o *query*, el sistema genera un embedding de esta pregunta, transformándola en una representación vectorial.
- Este embedding se utiliza para buscar en Pinecone (*Vector Database*) los embeddings más similares almacenados, que representan los chunks de texto generados en el proceso de scrapeo.
- Los textos correspondientes a los embeddings más similares (*Relevant Data*) se pasan al LLM. Esto permite al modelo utilizar, además del conocimiento adquirido durante su entrenamiento, información específica para formular una respuesta.

Este método permite personalizar el LLM, adaptando sus respuestas a las necesidades y contexto específicos de la consulta, lo que resulta en respuestas más precisas y relevantes.

Capítulo 4

Resultados

4.1. Impacto en el rendimiento del LLM

En este documento se detallan los resultados finales del proyecto, el cual ha evolucionado significativamente a lo largo de varios meses mediante la implementación de múltiples mejoras en el sistema. El LLM ha sido capaz de mejorar continuamente sus características y funcionalidades gracias a este enfoque incremental, que ha permitido alcanzar un gran rendimiento y exactitud. Gracias a estas mejoras, el LLM ha demostrado una habilidad excepcional para responder con precisión a la gran mayoría de las preguntas, permitiendo comparar su rendimiento con el de modelos avanzados como ChatGPT-4.

Sin embargo, para hacer el sistema escalable y práctico en diferentes contextos de uso, ha sido necesario realizar algunos compromisos. Uno de estos ha sido establecer un tamaño de chunk estándar, que si bien optimiza el rendimiento general en la mayoría de los documentos, en ciertos casos específicos podría no ser el ideal. Además, se ha optimizado la gestión de recursos para permitir que el modelo maneje un volumen alto de consultas simultáneas sin degradar la calidad de las respuestas. Este equilibrio entre precisión y escalabilidad ha sido clave para diseñar un sistema robusto y eficiente, adaptado a las necesidades de los usuarios en entornos dinámicos.

4.2. Análisis de resultados

Para evaluar y comparar el rendimiento del LLM personalizado con RAG frente a ChatGPT-4, se han realizado pruebas utilizando la herramienta de software TruLens. TruLens es una herramienta diseñada para medir de manera objetiva la calidad y la efectividad de las aplicaciones basadas en modelos de lenguaje a través

de funciones de retroalimentación[23].

Dentro de las métricas proporcionadas por TruLens, hay dos que son de especial interés: *Answer Relevance*, que mide la relevancia de las respuestas del LLM en relación a las preguntas formuladas, y *Ground Truth*, que evalúa la similitud entre las respuestas del LLM y las respuestas correctas predefinidas.

Para evaluar el rendimiento del sistema basado en RAG frente a ChatGPT-4, se ha utilizado un procedimiento basado en un documento específico. A partir de este documento se han creado un conjunto de preguntas y respuestas y luego, estas preguntas se plantearon tanto a ChatGPT-4 como al sistema LLM con RAG. Las respuestas fueron evaluadas usando las métricas de TruLens. Las pruebas se repitieron cuatro veces para obtener resultados consistentes.

Para que ChatGPT-4 pudiera responder adecuadamente, se utilizaron los Benchmark Assistants de OpenAI, proporcionándoles acceso a los documentos que sirvieron de base para las preguntas. En el caso del LLM, los documentos fueron procesados utilizando los métodos de scrapeo y chunking descritos en este proyecto. El texto extraído fue luego utilizado por el sistema de RAG para generar las respuestas a las preguntas formuladas.

La capacidad del LLM para proporcionar respuestas correctas indica que ha accedido a la información necesaria, validando así la efectividad de los procesos de scrapeo y chunking.

Prueba	**LLM**		**ChatGPT-4**	
	Ground Truth	**Answer Relevance**	**Ground Truth**	**Answer Relevance**
1	0.88	0.90	0.81	0.93
2	0.75	0.95	0.71	0.95
3	0.82	0.88	0.76	0.93
4	0.82	0.92	0.71	0.95

Cuadro 4.1: Comparación entre el LLM y ChatGPT-4

Haciendo la media de las puntuaciones se quedan los siguientes resultados:

Modelo	Media Ground Truth	Media Answer Relevance
LLM	0.82	0.91
ChatGPT-4	0.75	0.94

Cuadro 4.2: Media de las puntuaciones

Estás métricas muestran que el LLM supera a ChatGPT-4 en términos de *Ground Truth*, con una media de 0.82 comparado con 0.75, lo que indica que proporciona respuestas que son más cercanas a las esperadas. Por otro lado, ChatGPT-4 supera ligeramente al LLM en *Answer Relevance* con una media de 0.94 frente a 0.91, sugiriendo que sus respuestas, aunque menos precisas, son percibidas como más relevantes para los usuarios.

En conjunto, los resultados indican que el LLM desempeña muy bien su función, proporcionando respuestas precisas y relevantes. Aunque existe un margen para la mejora, especialmente en términos de aumentar la relevancia de las respuestas para igualar o superar a ChatGPT-4.

Para ampliar la evaluación y obtener una visión más completa del rendimiento del modelo de lenguaje, se han realizado pruebas adicionales utilizando diferentes formatos de documentos. Específicamente, se han empleado ocho PDFs distintos que detallan procedimientos de una empresa. Para estas pruebas, se generaron 25 preguntas distintas que se formularon tanto al asistente de OpenAI como al LLM con RAG.

Las respuestas obtenidas por ambos sistemas fueron evaluadas para medir su precisión y relevancia y los resultados se muestran en el cuadro 4.3:

PDF	LLM		ChatGPT-4	
	Ground Truth	Answer Relevance	Ground Truth	Answer Relevance
1	0.88	0.95	0.69	0.86
2	0.80	0.88	0.83	0.88
3	0.87	0.83	0.73	0.87
4	0.80	0.82	0.88	0.80
5	0.85	0.90	0.90	0.85
6	0.80	0.80	0.80	0.85
7	0.83	0.89	0.89	0.92
8	0.84	0.89	0.90	0.89

Cuadro 4.3: Resultados del LLM y ChatGPT-4 por documento PDF

Y las medias de las puntuaciones son las siguientes:

Modelo	Media Ground Truth	Media Answer Relevance
LLM	0.834	0.872
ChatGPT-4	0.827	0.865

Cuadro 4.4: Media de las puntuaciones

Los resultados mostrados indican que el rendimiento del LLM es ligeramente superior al de ChatGPT-4 en ambas métricas evaluadas. El hecho de que el LLM obtenga resultados comparables e incluso ligeramente superiores a los de ChatGPT-4 es muy positivo, teniendo en cuenta que ChatGPT-4 es uno de los modelos de lenguaje más avanzados disponibles actualmente.

Todo esto sugiere que el procesamiento de documentos mediante scrapeo y chunking se está realizando de manera efectiva, permitiendo al LLM extraer y utilizar la información relevante con gran precisión. Esto valida tanto la metodología utilizada como la calidad del sistema en comparación con una de las tecnologías líderes en el campo.

4.3. Limitaciones del proyecto

Las limitaciones del proyecto, aunque son desafíos significativos, también han guiado las estrategias de optimización y ajuste en el diseño del sistema. A continuación se detallan estas limitaciones y las soluciones implementadas:

- **Gestión de llamadas a modelos de lenguaje**

  Las interacciones con modelos de lenguaje externos, como ChatGPT o Claude, para realizar tareas como procesar imágenes pueden quedar bloqueadas en un estado de procesamiento indefinido, lo cual paraliza el proceso y puede llevar a fallos. Para manejar esta situación, se han implementado mecanismos de *timeouts* y, en algunos casos, se ha recurrido al uso de multithreading para terminar procesos que no responden.

- **Limitaciones temporales en AWS Lambda**

  Las funciones Lambda tienen una restricción de ejecución de 15 minutos, lo cual representa un desafío para procesos prolongados. Esto es particularmente crítico en el scrapeo de PDFs, que es un proceso complejo y que consume mucho tiempo. La solución implementada ha sido dividir el scrapeo en tres funciones Lambda distintas, permitiendo así completar el proceso dentro de los límites de tiempo establecidos.

- **Selección de la arquitectura de servicios en la nube**

  Durante las fases iniciales del proyecto, se exploraron varias arquitecturas, incluyendo el uso de instancias EC2 y AppRunner. Sin embargo, se enfrentaron problemas como la incompatibilidad de librerías y dificultades en la instalación de las mismas. A pesar de las limitaciones de tiempo de las Lambda, esta arquitectura resultó ser la más adecuada, ofreciendo la mejor combinación de flexibilidad, manejo de dependencias y facilidad de despliegue.

- **Optimización de la velocidad de scrapeo y chunking**

  Se consideró la implementación de multiprocessing para acelerar el scrapeo y el chunking, scrapeando una página por cada proceso. Sin embargo, esta estrategia introdujo una complejidad adicional y problemas de estabilidad, con procesos que ocasionalmente se colgaban y no finalizaban. Por lo tanto, se decidió no adoptar esta aproximación para mantener la estabilidad y simplicidad del sistema.

- **Desafíos en el manejo de PDFs**

  La variedad de formatos de los documentos PDF ha sido uno de los mayores retos. Cada PDF puede presentar un formato distinto, lo cual complica enormemente el proceso de scrapeo y extracción de información precisa y relevante. La adaptabilidad y robustez del proceso de scrapeo han sido cruciales para manejar esta diversidad.

Estas limitaciones han marcado áreas importantes para futuras mejoras, asegurando que cada etapa del desarrollo ayude a hacer el proyecto más preciso y eficiente en general.

Capítulo 5

Conclusiones y Trabajos Futuros

5.1. Conclusiones

A lo largo del proyecto, se han abordado y cumplido los objetivos planteados inicialmente en el Capítulo 1. En primer lugar, se realizó una exhaustiva investigación y análisis de las técnicas avanzadas de scraping y crawling, revisando la literatura y las tecnologías actuales para seleccionar las herramientas y bibliotecas más adecuadas. Este estudio permitió la implementación de un sistema robusto y actualizado que realiza la extracción de datos desde documentos en formatos variados como DOCX, TXT y PDF, así como desde páginas web.

El desarrollo del sistema de scraping y crawling se ha llevado a cabo con éxito, logrando un sistema capaz de manejar grandes volúmenes de datos de manera eficiente y escalable. Además, se implementaron técnicas de chunking para la segmentación de datos, garantizando que los chunks mantenían la coherencia semántica necesaria para ser útiles en contextos de personalización de LLMs.

La eficiencia del sistema de adquisición y organización de datos se ha evaluado, demostrándose su alta precisión y la relevancia de los datos extraídos y segmentados. Se estableció una pipeline de evaluación con TruLens que permitió medir la calidad de las respuestas del LLM en comparación con los benchmarks de la industria, como ChatGPT-4.

La integración de los datos procesados con el LLM se realizó mediante la implementación de Retrieval-Augmented Generation (RAG), que ha demostrado mejoras significativas en el rendimiento del modelo.

Finalmente, el proceso de documentación ha sido continuo y detallado, asegu-

rando que cada etapa del desarrollo y sus resultados estén claramente registrados. Este documento no solo refleja los esfuerzos y logros del proyecto, sino que también actúa como una guía valiosa para futuras investigaciones y desarrollos en el campo.

5.2. Propuestas para trabajos futuros

Mirando hacia el futuro, el desarrollo continuo y la mejora del sistema serán cruciales para mantener su relevancia y eficacia en el cambiante mundo de los modelos de lenguaje grandes. A continuación se presentan varias propuestas enfocadas en estos objetivos:

- **Actualización continua con nuevas tecnologías**

  Dada la rápida evolución de las tecnologías de scraping, crawling y procesamiento de lenguaje natural, es fundamental mantener el sistema actualizado con las últimas innovaciones. Esto implica revisar periódicamente el estado del arte y adoptar nuevas herramientas y técnicas que puedan mejorar la eficacia y eficiencia del sistema.

- **Mejora de la velocidad de scrapeo**

  Optimizar los procesos de scrapeo para acelerar la recopilación de datos es esencial. Esto podría lograrse mediante la mejora de los algoritmos existentes, la implementación de soluciones de hardware más rápidas, o la utilización de técnicas de programación paralela y distribuida.

- **Expansión a nuevos formatos de documentos**

  Ampliar la capacidad del sistema para procesar otros formatos de documentos, como presentaciones de PowerPoint, podría aumentar significativamente su utilidad. Esto permitiría extraer datos de una gama más amplia de fuentes de información, haciendo el sistema más versátil.

- **Chunking más inteligente**

  Desarrollar métodos más avanzados para la segmentación de datos que adapten el tamaño y la forma de los chunks a las características específicas de cada documento ayudaría a mejorar la calidad del sistema. Esto podría incluir variar el tamaño del chunk según en la densidad de información, la naturaleza del contenido o la estructura del documento.

Estas áreas de desarrollo no solo mejorarán la eficiencia y efectividad del sistema actual, sino que también asegurarán que pueda adaptarse y responder a los desafíos emergentes en el ámbito de la inteligencia artificial.

Bibliografía

[1] *Hello GPT-4o*. Accedido: 2024-06-01. Mayo de 2024. URL: https://openai.com/index/hello-gpt-4o.

[2] *Introducing OpenAI o1*. Accedido: 2025-01-05. URL: https://openai.com/o1.

[3] *Inteligencia artificial generativa: Introducción a los LLMs*. https://www.gobierto.es/blog/inteligencia-artificial-generativa-introduccion-a-los-llms. Accedido: 2024-06-01.

[4] *Meta AI: LLaMA 3.1*. Accedido: 2025-01-05. URL: https://ai.meta.com/blog/meta-llama-3-1/.

[5] *¿Qué son los modelos de lenguaje de gran tamaño (LLM)?* Accedido: 2024-06-01. URL: https://aws.amazon.com/what-is/large-language-model/.

[6] Daivi. *7 top large language model use cases and applications*. Accedido: 2024-06-01. URL: https://www.projectpro.io/article/large-language-model-use-cases-and-applications/887.

[7] *What is retrieval-augmented generation (rag)? - IBM research*. Accedido: 2024-06-16. URL: https://research.ibm.com/blog/retrieval-augmented-generation-RAG.

[8] Anina Ot. *What is data scraping? definition how to use it*. Accedido: 2024-06-01. Sep. de 2023. URL: https://www.datamation.com/big-data/data-scraping/.

[9] Francisco Torreblanca. *Qué Es el crawling y su importancia en seo*. Accedido: 2024-06-01. Feb. de 2021. URL: https://www.esic.edu/rethink/marketing-y-comunicacion/crawling-seo.

[10] *Walkthrough Top Python Libraries for PDF Processing*. Accedido: 2024-06-01. URL: https://www.educative.io/courses/pdf-management-python/walkthrough-top-python-libraries-for-pdf-processing.

[11] *Data Scraping*. Accedido: 2024-06-01. Dic. de 2023. URL: https://www.imperva.com/learn/application-security/data-scraping/.

[12] Roie Schwaber-Cohen. *Chunking Strategies for LLM Applications*. Accedido: 2024-06-02. URL: https://www.pinecone.io/learn/chunking-strategies/.

[13] Luis Celis. *Comenzando con AWS boto3*. Accedido: 2024-06-12. Mar. de 2020. URL: https://medium.com/@luiscelismx/comenzando-con-aws-boto3-876fd0d6686f.

[14] *Anthropic Help Center*. Accedido: 2024-06-14. URL: https://support.anthropic.com/es/articles/7989434-que-es-claude.

[15] *What is OCR? - optical character recognition explained - AWS*. Accedido: 2024-06-14. URL: https://aws.amazon.com/what-is/ocr/.

[16] *WEB CRAWLING AND WEB SCRAPING*. Accedido: 2024-06-14. Abr. de 2023. URL: https://www.datacentric.es/en/web-crawling-and-web-scraping/.

[17] *What Is a Sitemap? Website Sitemaps Explained*. Accedido: 2024-06-14. URL: https://www.semrush.com/blog/website-sitemap/.

[18] *What is Langchain? - Langchain explained - AWS*. Accedido: 2024-06-15. URL: https://aws.amazon.com/what-is/langchain/.

[19] *Maximizing LLM Performance with Effective Chunking Strategies for Vector Embeddings*. Accedido: 2024-06-15. URL: https://vectorshift.ai/blog/maximizing-llm-performance-with-effective-chunking-strategies-for-vector-embeddings.

[20] Faacute;bio Serrano. *Chunking Strategies for LLM Applications*. Accedido: 2024-06-15. Abr. de 2024. URL: https://medium.com/@fcatser/chunking-strategies-for-llm-applications-dfd44e17b163.

[21] Chia Jeng Yang. *An introduction to rag and simple/ complex rag*. Accedido: 2024-06-15. Abr. de 2024. URL: https://medium.com/enterprise-rag/an-introduction-to-rag-and-simple-complex-rag-9c3aa9bd017b.

[22] Ben Lorica. *Techniques, challenges, and future of augmented language models*. Accedido: 2024-06-15. Oct. de 2023. URL: https://gradientflow.com/techniques-challenges-and-future-of-augmented-language-models/.

[23] TruEra. *Trulens*. Accedido: 2024-06-16. URL: https://www.trulens.org/.