Creación de un Agente RAG (Retrieval - Augmented Generation)

Grupo 4:
Cesar Carrera 00344613
Edgar Guzmán 00344822
Juan Diego Sánchez 00344455
Byron Vinueza 00345908

Resumen
El sistema implementado tiene como objetivo obtener respuestas precisas en función de documentos externos. Se describe el desarrollo y evaluación de un agente RAG integrando técnicas de recuperación de información y generación de texto con el propósito de responder consultas del usuario basados en archivos PDF.

Se carga los archivos PDF desde un directorio designado utilizando *PYPDFLoader,* luego se fragmenta mediante *RecursiveCharacterTextSplitter*. Cada fragmento se transforma en una representación vectorial empleando *OpenAIEmbeddings* y los embedding resultantes se guardan en *Chroma* que es un almacén vectorial configurado para búsquedas por similitud de coseno.

Se emplea un modelo GPT-4 para generar la respuesta: se recupera los fragmentos más relevantes de la base vectorial aplicando un umbral de similitud semántica preestablecido en 0.3, luego se emplea estos fragmentos como contexto para la generación de texto. Con este proceso se pretende garantizar un manejo eficiente de documentos extensos, una recuperación precisa y una mayor relevancia en las respuestas.

La arquitectura aquí descrita saca provecho del diseño modular de los componentes de LangChain, destacando su escalabilidad, modularidad y precisión en sistemas de diálogo basados en conocimiento.

Mediante este proceso se demuestra la capacidad del agente para extraer y sintetizar información de múltiples fuentes PDF destacando aplicaciones potenciales en soporte automatizado al cliente y asistencia en investigación.

Contexto del Agente RAG

¿Qué es un Agente RAG ?

La combinación de modelos de lenguaje de gran escala (LLMs) con mecanismos externos de recuperación de información ha dado lugar a una arquitectura híbrida conocida como **RAG (Retrieval-Augmented Generation)**. Esta arquitectura se ha vuelto fundamental en la construcción de agentes conversacionales más precisos y contextualizados.
Un agente RAG (Retrieval-Augmented Generation) es un sistema que combina la recuperación de información y la generación de texto. Utiliza modelos de lenguaje para responder preguntas basadas en documentos, permitiendo responder preguntas utilizando información contenida en archivos PDF.
“RAG allows LLMs to pull in facts from a knowledge base at query time, instead of relying solely on what the model was trained on.” (OpenAI, 2023).

Dentro del dominio de la inteligencia artificial, los agentes RAG, se encuentran dentro de la categoría de Agentes Expertos, que utilizan técnicas de recuperación de información y generación de texto. Combina algoritmos de búsqueda y modelos de lenguaje para dar respuestas precisas y concretas a preguntas de un dominio específico.

El enfoque RAG fue propuesto por primera vez por Lewis et al. (2020) como una forma de mejorar la precisión y actualidad de las respuestas generadas por modelos de lenguaje, especialmente en tareas como preguntas y respuestas, chatbots inteligentes y agentes conversacionales especializados.

Un agente RAG es un sistema híbrido que integra dos componentes principales:

1. **Un módulo de recuperación** (retriever), responsable de buscar información relevante desde una base de conocimientos externa.

2. **Un módulo generativo** (generator), encargado de elaborar respuestas utilizando tanto el contexto de entrada como la información recuperada.

Flujo de Trabajo de un Agente RAG con OpenAI y Chroma

1. Carga de documentos PDF
Se utiliza PyPDFLoader, una clase de la biblioteca LangChain, que permite extraer el texto de los PDF y convertirlo en documentos manipulables en Python.
El contenido extraído se convertirá más adelante en texto indexable para búsquedas semánticas. Este paso inicia el pipeline RAG al permitir usar documentos personalizados.

2. División del texto (Chunking)
Los documentos PDF extraídos se dividen en fragmentos de texto más pequeños llamados *chunks*.
Se usa RecursiveCharacterTextSplitter, que divide el texto conservando la semántica (evita cortar frases a la mitad) y ajustándose a un tamaño definido en este caso se usó un tamaño de 5000 chunks porque esto mejora la calidad de las respuestas generadas, ya que se reduce el riesgo de pérdida de contexto, también los fragmentos más largos tienden a mantener coherencia de ideas, párrafos o secciones completas.

3. Creación de Embeddings

Cada fragmento de texto se convierte en un vector numérico o *embedding*, que captura el significado semántico del texto. Los embeddings permiten comparar textos por su significado, no por coincidencias exactas de palabras. Esto es clave para hacer una búsqueda semántica eficiente.
Se utiliza la clase OpenAIEmbeddings, que se llama a la API de OpenAI.

4. Indexación en Chroma
Los chunks y sus embeddings se almacenan en una base de datos vectorial llamada Chroma. Se utiliza la clase Chroma de LangChain, que actúa como base de datos en memoria o persistente. Aquí, cada embedding queda vinculado al texto original del fragmento. Esto es importante ya que nos permite hacer búsquedas rápidas de los fragmentos más parecidos a una pregunta del usuario, usando similitud de vectores en este caso por distancia coseno. Se decidió usar esta distancia porque, a diferencia de la distancia Euclidiana o Manhattan, esta distancia es la mejor para embeddings de texto. No se ve afectada por la magnitud del vector, lo que es útil porque los embeddings pueden tener distintas normas por el texto de entrada. (Manning et al. 2008).

5. Recuperación y generación

Cuando el usuario hace una pregunta:

1. Se genera el embedding de la pregunta.

2. Se consulta la base vectorial para obtener los fragmentos más relevantes.

3. Se construye un prompt que combina la pregunta + contexto.

4. Se pasa ese prompt a GPT-4 para generar una respuesta precisa.

Metodologías de Evaluación
1. Evaluación de la recuperación (IR)
1.1. Métricas automáticas
- Precision / Recall
  Mide el porcentaje de documentos relevantes entre los k primeros resultados.
- Mean Reciprocal Rank (MRR)
  Evalúa la posición de la primera respuesta relevante.
- nDCG (Normalized Discounted Cumulative Gain)
  Considera la posición y la relevancia graduada de todos los resultados recuperados.

Objetivo: asegurar que los fragmentos que aporta el vector store (Chroma) realmente contengan la información necesaria para responder la pregunta.

2. Evaluación de la generación (NLG)
2.1. Métricas de coincidencia textual
- BLEU, ROUGE, METEOR: comparan n-gramas con una respuesta “gold”. Útiles para tareas de resumen o QA extractivo.

- BERTScore: compara en “espacio de embeddings” las similitudes semánticas entre la salida y la referencia.

Limitación: estas métricas no captaron bien la veracidad ni la coherencia global.

2.2. Métricas de factualidad y consistencia

- QAGS / QUESTEval: generan preguntas secundarias sobre la respuesta y comprueban si un modelo puede contestarlas correctamente, midiendo la consistencia interna.
- Entailment-based: usan un modelo de NLI para verificar que la respuesta “entail” (concluye lógicamente) de un pasaje de referencia.

2.3. Evaluación humana

- Exactitud factual: un evaluador compara la respuesta con el documento original y califica (p. ej., “correcta / parcialmente correcta / incorrecta”).
- Coherencia y fluidez: se puntúa la legibilidad y la estructura lógica.
- Grado de hallucination: porcentaje de afirmaciones no respaldadas por el contexto.

3. Evaluación end-to-end

3.1. Métricas de tarea

- Exact Match (EM) / F1
  Para sistemas de pregunta-respuesta cerrada, mide la coincidencia exacta o parcial de tokens.
- Tasa de éxito
  Porcentaje de consultas respondidas de forma satisfactoria (según criterios predefinidos).

- Tiempo de respuesta y uso de recursos
  Mide la latencia total del pipeline y consumo de CPU/RAM/GPUs para escalabilidad.

3.2. Pruebas de usuario (A/B testing)

- Comparar dos versiones del agente (p. ej., distinto tamaño de chunk o diferente prompt) con usuarios reales.
- Medir preferencias subjetivas: claridad, velocidad, utilidad.

4. Metodología de evaluación

1. Definir objetivos claros
   ¿Buscas máxima precisión factual? ¿Mayor cobertura de dominios? ¿Respuestas más creativas?
2. Seleccionar datasets y tareas
   Mezcla conjuntos públicos (MS MARCO, BEIR) con ejemplos reales de tu dominio.
3. Automatizar métricas
   Integra scripts que calculen Precision@k, ROUGE, BERTScore y QAGS tras cada cambio de modelo o pipeline.
4. Análisis de errores
   - Recuperación fallida: ¿el fragmento correcto nunca apareció? Ajustar embeddings o vector store.
   - Generación errónea: ¿el fragmento estaba bien, pero la respuesta se equivoca? Revisar prompt o modelo.

5. Iterar con evaluaciones humanas
   Sesiones de revisión periódicas con expertos del dominio que validen las salidas y señalan patrones de fallo.

5. Herramientas y librerías

- LangChain Evaluators (módulo langchain.evaluation): integra métricas de QA y summarization.
- Hugging Face Evaluate: implementa ROUGE, BLEU, BERTScore, METEOR, etc.
- Beir Toolkit: evaluación integrada en benchmarks de IR.
- QAEval / QuestEval: para medir factualidad post-generación.
- Custom dashboards: Grafana/MLflow para monitorizar métricas en producción.

6. Buenas prácticas finales

- Evaluar cada componente por separado (retrieval vs generation) y luego "end-to-end".
- Versionar el pipeline completo: anota datos, embeddings, prompt templates y modelo LLM usado.
- Incorporar feedback real: logs de interacción y métricas de usuarios en producción.
- Balancear trade-offs: a veces sacrificar un poco de precisión en retrieval mejora la latencia, o viceversa.

Conclusiones

El uso de agentes RAG potenciados por OpenAI ofrece una solución poderosa para la generación de lenguaje apoyada en información externa. Esta arquitectura habilita agentes inteligentes más confiables, personalizables y capaces de responder con precisión a preguntas específicas en tiempo real.

A través de herramientas como la API de embeddings, plugins de recuperación y la generación con GPT-4, OpenAI ha democratizado el acceso a este paradigma avanzado de inteligencia artificial.

El uso de un *chunk size* grande de 5000 nos ayuda porque los fragmentos largos contienen ideas completas, párrafos enteros o secciones enteras de un documento, también esto mejora la calidad del contexto que se entrega al modelo, permitiéndole generar respuestas más precisas y fundamentadas. Por otra parte, se logra evitar casos en que una idea queda dividida entre varios chunks pequeños lo que podría dificultar su recuperación completa.

Se determinó el uso de la distancia coseno porque esto permite recuperar fragmentos relevantes aunque no contengan las mismas palabras exactas que la pregunta del usuario. Como los embeddings varían en magnitud según el texto, esta métrica evita errores por diferencia de escala y se centra en orientación semántica.

Repartición de tareas

César Carrera

- Investigación de cómo funciona un agente RAG
- Inicio de creación de código para la generación de preguntas y respuestas

- Elaboración de informe

Edgar Guzman
- Investigación de cómo usar el API Key
- Comienzo de elaboración de código para leer archivos PDF
- Elaboración de informe

Juan Diego Sanchez
- Finalización de código para recopilar información de los PDFs.
- Recopilación de PDFs en una carpeta
- Elaboración de informe

Byron Vinueza
- Finalización del código para procesar la información de los archivos y generar preguntas y respuestas.
- Elaboración de informe

Bibliografía

Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., ... & Riedel, S. (2020). Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. *Advances in Neural Information Processing Systems*, 33. **https://arxiv.org/abs/2005.11401**

Manning et al., “Introduction to Information Retrieval”, 2008

OpenAI. (2023). *Retrieval Plugin Documentation*. **https://platform.openai.com/docs/guides/retrieval**