INTRO RAG BAKE-OFF FOR TECHNICAL Q&A

INFORME DE LOS RESULTADOS OBTENIDOS

Manuel Pontón Sarrió & Francisco J. López Pacheco

Descripción del proyecto

El presente informe constituye nuestra respuesta al "*Concurso de Modelización de Problemas de empresa 2025*" organizado por la **Universidad Complutense de Madrid** (UCM) y corresponde al reto propuesto por **Management Solutions**, "*RAG Bake-Off for Technical Q&A*".

El proyecto tiene como objetivo **diseñar, implementar y evaluar un sistema de preguntas y respuestas (Q&A)** sobre un artículo técnico reciente —en este caso, un paper de *Meta SuperIntelligence Labs* que presenta el *framework RAG* llamado *REFRAG*— y **comparar cómo diferentes estrategias de recuperación de información (*Dense Retrieval* y *BM25*) ayudan a un modelo de lenguaje** (LLM) a responder preguntas de opción múltiple, en nuestro caso *Gemini AI*.

Estructura del Código

El proyecto consta de varios archivos, en el *main.py* se **asigna valores a las variables** y se llama a las funciones que constituyen el pipeline en sí. Los resultados se **guardan** en uno de los ficheros *.jsonl*. Las **funciones** están definidas y desarrolladas en el archivo *data.py* dentro del directorio *common* del proyecto.

Requisitos

Para la ejecución del código de respuesta hemos necesitado **descargar las librerías** indicadas en "*requeriments*" con especial cuidado con *faiss*, que dependiendo del sistema operativo es faiss (MAC) o *faiss-cpu* (Windows). Además se requiere de una **versión de *python* superior o igual a 3.11**, en particular usamos 3.11 en nuestras simulaciones.

Fases del Proyecto y Metodología

Para evaluar cómo distintos *pipelines* de recuperación de información afectan la capacidad de un modelo de lenguaje (LLM) al responder preguntas de opción múltiple, se siguió la siguiente metodología:

- ***Dataset inicial***

  Management Solutions nos proporcionó un archivo *.json* con **50 preguntas de elección múltiple** (A, B, C, D) sobre el *paper* de *REFRAG* junto a sus correspondientes respuestas. Se eligió este tema debido a la reciente publicación del artículo, de manera que el **LLM no tuviera conocimiento previo sobre él**.

- ***Variables del código***

  1. **"Tipo de Contexto"**: Nos referimos a **qué estrategia de búsqueda de información** vamos a utilizar, si le asignamos el valor 0, usamos el *LLM Baseline*, el valor 1, El *BM25*, el valor 2, el *Dense Retrieval* y el valor 3, una *estrategia híbrida* explicada en los pipelines. El nombre de tipo de contexto para seleccionar el modelo deriva de la función *get_context* presente en el archivo *data*.

Fases del Proyecto y Metodología

- ***Variables del código***

  2. **Tipo de *Chunking***: De nuevo, un entero entre 0 y 3 incluidos, seleccionamos una de las 4 estrategias que proponemos en el *chunking*.
  3. **"K"**: **Número de *chunks* que utilizará el modelo para generar la respuesta**, sólo debe tomar valores naturales (sin el cero) hasta el límite de chunks (104 para el caso del de distancia fija), nosotros sólo hemos utilizado de 1 a 10
  4. **Número de preguntas**: Naturales del 1 al 70, sólo coge las primeras
  5. **Número de iteraciones**: Cantidad de veces que quieres repetir el test de n preguntas.
  6. **Clave de usuario**: Se necesita introducir la ***api-key de google*** para utilizar *Gemini*. Como hemos usado una prueba de un plan de pago, no tenemos un límite de preguntas por minuto, si la clave del usuario es gratuita, se debe añadir un *timesleep. Por si acaso, dejamos aquí una: "AIzaSyCqV2TCeatdb4At2D4dyW8QXfIcRQy5goA".*

- ***Embedding & Chunking***

  Para la fragmentación del texto, utilizamos varias estrategias de chunking:

  1. **Fixed-Length Chunking**: Esta es la fragmentación por **distancia fija**, (aunque con un *overlap* del 25 %) comenzamos con ella debido a su sencilla programación. Utilizamos un *chunksize* de 200, *overlap* de 50.
  2. **Sentence Chunking**: Fragmentación del texto haciendo un *split* si encuentra o bien un punto, o bien un signo de exclamación o interrogación.
  3. **Paragraph Chunking**: Hacemos ahora un *split* cuando se encuentre un "\n" (Salto de línea).
  4. **Story Driven Chunking**: Este *chunking* es un **híbrido**, junta el de distancia fija junto a un **cribado de las partes** que no corresponden al *cuerpo del texto*.

  En cuanto al embedding hemos utilizado un diccionario y la librería *faiss*.

- ***Modelos y Pipelines***

  Para cada *chunking* se pueden realizar los siguientes modelos:

  1. **LLM Baseline**: El modelo genera respuestas directamente sobre la pregunta sin acceso a recuperación de información y sin el uso de los *chunks*. Por esta razón también se nos pedía un LLM con fecha anterior al 1 de septiembre de 2025.
  2. **LLM + BM25 + Chunking**: Probamos como el BM25 -técnica de búsqueda basada en palabras clave- responde junto al chunking que queramos.
  3. **LLM + Dense Retrieval + Chunking** Manteniendo la libertad de *chunking* probamos nuestra otra estrategia de recuperación de información, el *Dense Retrieval*.
  4. **LLM + Estrategia Híbrida**: Aplica los 2 modelos de *retrieval* y si la intersección es no vacía selecciona ese chunk para el modelo y si es vacía, aporta la unión.

  Además para cada modelo se puede variar la cantidad de *chunks* que necesita (Variable K). Con estos *Pipelines* hemos realizado inicialmente unas 49 simulaciones.

- ***Métricas***

  1. **Número de aciertos en el Test**: Número entero en [0,70]
  2. **Source Attribution Accuracy:** Evaluamos la similitud entre los conjuntos: A (Índices de *chunks* "veraces") y B (Índices de *chunks* aportados). El caso ideal sería A = B. Definimos:
     - **Overlap:** #Intersección de A Y B, *chunks* correctos dentro de los escogidos.
     - **Recall:** Overlap/ #A, representa **la recuperación de información**, (Chunks correctos del modelo, entre los chunks correctos totales) en nuestro caso, es la medida más importante, ya que nos dice si los chunks están siendo bien elegidos para generar la respuesta en el cuestionario.
     - **Jaccard:** Overlap / #(A u B), representa chunks correctos de nuestro experimento entre todos los posibles.

Fases del Proyecto y Metodología

- ***Métricas***

  3. **Tiempo de recuperación:** Lo incluimos porque lo consideramos un factor determinante, pero cabe destacar que este no sólo varía con la complejidad computacional, sino que también varía por otros factores (tráfico del LLM, retardos en la generación de respuesta…) por dicha razón se puede justificar las diferencias de tiempo considerables (> 30 s) al ejecutar la misma simulación pero en diferentes momentos. **Ante esta situación decidimos que no es nuestra prioridad, pero lo tenemos en cuenta.**

- ***Construcción del Dataset***

  Con las métricas definidas decidimos preparar un plan de simulaciones:
  Manteniendo el Chunking 0, el de longitud fija y variando la k entre 1 y 10:
  - 10 Simulaciones, una por cada k, sólo para BM25 (Tipo de contexto = 1).
  - 10 Simulaciones, una por cada k, sólo para Dense Retrieval (Tipo de contexto = 2).

  Seleccionando k = 6 (óptima), variamos los tipos de chunking:
  - 8 simulaciones con BM25, 2 por cada Chunking (0, 1, 2, 3)
  - 8 simulaciones con Dense Retrieval, 2 por cada Chunking
  - 8 simulaciones con la estrategia híbrida, 2 por cada Chunking
  - 2 simulaciones con el baseline

  Aparte también se incluye también una simulación con un número de iteraciones mayor que uno, para comprobar que funciona junto a 2 idénticas pero en distinto tiempo.

- ***Elaboración del informe***

  La última fase es la elaboración del análisis de datos y redacción de este informe. Además después de verlos decidimos mejorar el código.

Análisis de los datos y mejoras del código

Empezamos buscando la k óptima (respecto de los aciertos) para el BM25 y para el *Dense Retrieval*. Tomamos los datos de las primeras 20 simulaciones y obtenemos:

En función del Tiempo

Valores de K	Dense Retrieval	BM25
1	507,74	348,28
2	472,98	364,75
3	459,18	237,09
4	289,11	228,91
5	321,57	225,83
6	276,81	243,27
7	296,25	255,86
8	322,96	251,12
9	306,77	256,81
10	286,96	271,65

En función del Acierto

Valores de K	Dense Retrieval	BM25
1	51	61
2	59	67
3	59	68
4	66	68
5	66	68
6	67	69
7	66	68
8	66	68
9	67	68
10	67	69

Con estos datos vamos a fijar la k = 6 como valor óptimo, a continuación vamos a medir el acierto y el *source attribution score:*

- **Primera Métrica: Aciertos**

Modelo \ Chunking	Fixed-length	Sentence	Paragraph
BM25	69, 68, 69, 68	67, 67, 66, 66	55, 58, 55, 57
Dense Retrieval	67, 67, 67, 67	63, 62, 58, 58	51, 50, 51, 50
Hybrid	65, 65, 64, 65	60, 58, 54, 53	40, 44, 38, 37

Análisis de los datos y mejoras del código

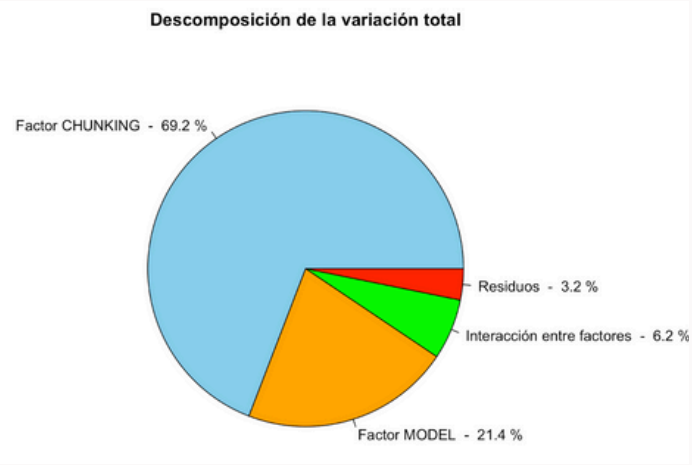
- ***Primera Métrica: Tiempo***

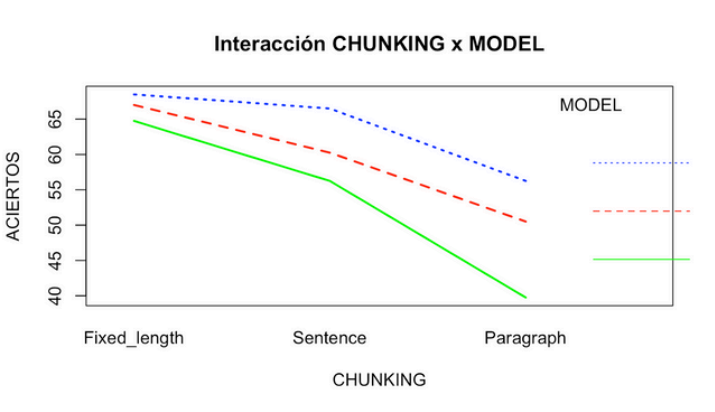
  De los resultados aquí representados se desprenden algunas conclusiones inmediatas.
  En cuanto a modelos de ***chunking:***
  - El *fixed length chunking* es, con diferencia, el mejor con un porcentaje de aciertos del 95,35%.
  - El *sentences chunking* ofrece unos resultados ligeramente inferiores que el anterior, promediando 87,14% de aciertos.
  - *Paragraph chunking* degrada sistemáticamente la recuperación, con un 69,76% de aciertos.

  En cuanto a modelos de ***Dense retrieval***:
  - El modelo *BM25* funciona muy bien, probablemente impulsado por la gran cantidad de tecnicismos en el documento
  - Los modelos *Dense* y *Hybrid* tienen, notablemente, un peor resultado

  Nos hemos permitido hacer un análisis multivariante de estos datos en *RStudio* que ha arrojado los siguientes resultados:

```
> modelo <- aov(ACIERTOS ~ CHUNKING * MODEL, data = diseño_replicas_ord)
> summary(modelo)
               Df Sum Sq Mean Sq F value   Pr(>F)
CHUNKING        2 2008.4  1004.2  292.33  < 2e-16 ***
MODEL           2  622.9   311.4   90.66 1.05e-12 ***
CHUNKING:MODEL  4  180.3    45.1   13.12 4.64e-06 ***
Residuals      27   92.7     3.4
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```





Según se ve en el gráfico de sectores adjuntado al *dashboard*, *chunking* es responsable del 69,2% de la variabilidad en los aciertos. Esto va en la línea de lo que exponíamos anteriormente. También hay diferencias importantes entre los 3 métodos de *retrieval*. Sin embargo, su impacto es tres veces menor que el impacto del *chunking* (véase ‘Sum Sq’).
El error bajo 3.43 indica consistencia en las réplicas del experimento, y los F-values observados aseguran que la variación reflejada es real, no debida a errores.

El grado de interacción que se observa, nos lleva a pensar:
- Que el efecto del método de *retrieval* depende “algo” del tipo de *chunking*.
- Que el rendimiento del *chunking* depende “algo” del método de *retrieval*.

Análisis de los datos y mejoras del código

- ***Primera Métrica: Tiempo***

Pero el efecto de esta interacción es más limitado: 6,2% de la variabilidad, indicando que el comportamiento de los dos factores: CHUNKING y MODEL, son, en buena medida, independientes. Esto se traduce en que un buen/mal *chunking* lo es con todo modelo y viceversa, las tendencias no se invierten en ningún caso. Esto también se observa en las rectas "paralelas" del interaction plot (adjuntado al *dashboard*).

Esta ligera interacción que detecta el modelo va en línea con lo observado: los malos *chunking* son todavía peores con los modelos RAG

.

En esa línea, los modelos RAG han alcanzado **peor desempeño**, lo cual resulta muy sorprendente, pues estos modelos incorporan *embeddings*, un método más nuevo y sofisticado que tiene en cuenta el contexto, la semántica, etc. mientras que el modelo BM25 (que ha aportado mejores resultados) es más rudimentario: sólo tiene en cuenta la frecuencia de las palabras y otros parámetros léxicos.

Esto nos lleva a pensar que hemos fallado en la implementación de estos métodos RAG.

HIPÓTESIS: Creemos que la causa de estos resultados es **una deficiente lectura del pdf**, en la que el cuerpo del texto queda entremezclado con notas y pies de página, referencias, gráficos, tablas y demás contenido.

Vamos a hacer una prueba para comprobar de forma empírica que la hipótesis es correcta: vamos a completar un pipeline de *Dense Retrieval* con un texto de partida que es el cuerpo principal del pdf.

Y usando cambiando el flag "prueba" a True en el main hemos obtenido los siguientes resultados, que se guardan en el fichero: "Modelizacion_25_26_UCM_IntroRAGprueba.jsonl".

	Aciertos previos	Aciertos SÓLO con cuerpo de texto	Variación
Fixed-length	67	67	0
Sentence	60,25	59	-2%
Paragraph	50,5	70	+38%
TOTAL	59,25	65,33	**+10,25%**

Esta mejora de los resultados tras usar el cuerpo de texto puro señala, casi con total seguridad, que **el problema está en la lectura de datos**. Es decir, hemos comprobado que:

- BM25 es más robusto frente al ruido estructural
- RAG es vulnerable al ruido

Pero no despreciaremos el experimento ya realizado. Para poder completarlo con los resultados que esperamos de los modelos de embeddings, hemos implementado una solución que ataja específicamente el problema:

Implementamos un método de chunking híbrido: el **story-tell chunking**, aplicado sobre los chunks obtenidos en el fixed length chunking -los que dieron mejores resultados- nos permitirá eliminar el "ruido" del texto leído para mejorar la métrica de los embeddings.

Un experimento de dense retrieval con este chunking ha cosechado **67 aciertos, más de un 95%**, un porcentaje muy alentador, en la línea de lo esperado.

Análisis de los datos y mejoras del código

- ***Primera Métrica: Tiempo - Conclusiones***

  Usar *BM25* cuando:
  - El texto está sucio (PDF mal cargado).
  - El dominio es muy técnico y cargado de *keywords*.
  - Las preguntas contienen términos exactos del texto.

  BM25 es extremadamente robusto en entornos de ruido estructural

  Usar *Dense Retrieval (o Hybrid)* cuando:
  - El texto está limpio o se puede limpiar con *chunking* semántico.
  - Las preguntas requieren razonamiento semántico o paráfrasis.
  - Hay sinónimos, ambigüedades o lenguaje natural complejo.

  Dense Retrieval supera a BM25 en entornos semánticos bien estructurados

- ***Segunda Métrica: Source Attribution Accuracy***

  Como nuestra lectura de pdf incorporaba notas y pies de página, referencias, gráficos, tablas y demás contenido; las referencias del texto especificadas en el json no se encuentran explícitamente en ningún *chunk*.

  Para salvar este obstáculo **hemos creado la función** *find_reference_in_chunks()*.
  Esta función toma dos argumentos:
  - la lista de referencias aportadas en el json del enunciado
  - la lista de *chunks* del tipo correspondiente

  Y buscando en *substrings* de cada *chunk* devuelve un diccionario donde cada clave es una pregunta y la palabra es una list[int] que contiene los *chunks* donde está la referencia.
  Para obtener conclusiones más precisas, hemos hecho una comprobación visual de los resultados.

  De esta forma, en *estadistica.py*, comparamos:
  - A: { los *chunks* que han sido devueltos al modelo}
  - B: { los *chunks* donde se encontraba la referencia}

  Vamos a considerar dos de las métricas más representativas que hay en el script y vamos a ver cómo se comportan en todas las respuestas de todas las observaciones distribuidas según las variables CHUNKING y MODEL:

  PRIMERA: Para cada respuesta: **overlap = |A ∩ B|**

average overlap	Fixed-length	Sentence	Paragraph	Story-Driven
BM25	1,0267	0,9095	0,3785	1,1571
Dense Retrieval	0,8346	0,7000	0,3285	0,9714
Hybrid	0,8785	0,6714	0,1714	0,9142

  SEGUNDA: Para cada respuesta**: precision = chunks localizados / chunks devueltos**

average precision	Fixed-length	Sentence	Paragraph	Story-Driven
BM25	0,2883	0,1609	0,0630	0,1928
Dense Retrieval	0,1804	0,1166	0,0547	0,1619
Hybrid	0,3644	0,2532	0,0841	0,3835

Análisis de los datos y mejoras del código

- ***Segunda Métrica: Source Attribution Accuracy***

  Convenimos en considerar una buena SAA a toda medición **con overlap mayor que 0,8**. Señalaremos de verde los experimentos que logren ese nivel.

  Se observa que el factor más determinante para determinar la SAA de un experimento es el tipo de *chunking* elegido.

  El método story-driven es superior para tareas de source attribution, superando incluso al fixed-length, un método que también prueba ser muy robusto.
  Estos datos ilustran como el story driven chunking ha conseguido solucionar, al menos en gran medida, el problema que había subyacente con la lectura del pdf.

  En contrario, cabe resaltar lo mal que funciona el paragraph chunking, también en esta métrica, pues contiene demasiada información irrelevante y diluye los fragmentos pequeños de referencia.

  NOTA: Nótese que los valores altos que cosecha Hybrid en “precision” son debidos a que en este modelo se toma la intersección de los contextos de cada método, por lo que el denominador de la fórmula será generalmente menor; no porque sea un mejor modelo.

- ***Tercera Métrica: Tiempo de ejecución***

  Sobre esta métrica ya hemos hecho un breve análisis en la búsqueda del K óptimo.