TFM
RAG

nombres

4. EVALUACIÓN:

a. Metodología:

La evaluación del RAG tanto a nivel global como a nivel componente es fundamental para mejorar nuestro sistema, ya que es nuestra forma de medir su rendimiento en diversos aspectos y nuestro objetivo es ir mejorando este rendimiento.

Nuestra estrategia se fundamenta en el uso de los módulos que tiene llamaindex para evaluar diversos aspectos, tanto de las respuestas como de los retrievals, junto con datasets de evaluación, y un modelo “gold”, en este caso GPT-4, que usan estos módulos para evaluar. También hacemos uso de tablas de benchmarks públicos para una primera selección de modelos.

b. Benchmarks públicos:

Son tablas en las que se comparan modelos, a través de algunas de sus características (memoria necesaria, max tokens, etc.), y de los resultados que han obtenido en diversas categorías. Las categorías abarcan aspectos como deducciones lógicas, matemáticas simples, responder a preguntas de contexto general, etc. en el caso de LLMs, y clasificación, clustering, sumarización, etc. en el caso de embeddings. Estas tablas existen tanto para modelos LLM como para modelos de embeddings, y son un buen lugar por donde empezar a evaluar y seleccionar los modelos que vayamos a usar en nuestro sistema según nuestros recursos y nuestras necesidades.
==========
[a lo mejor incluirlos en el anexo, o si no dejar aquí los enlaces]
==========
(https://huggingface.co/spaces/mteb/leaderboard)
(https://huggingface.co/spaces/open-llm-leaderboard/open_llm_leaderboard)

b. Datasets:

Una pieza fundamental para evaluar son los datasets destinados a la evaluación. Estos vienen con un corpus, unas queries, y unas respuestas y/o unos contextos de referencia, dependiendo si son datasets para evaluar las respuestas o el retrieval. Para evaluar se comparan las respuestas o contextos obtenidos con los de referencia del dataset. Se puede tanto descargar datasets variados de llamahub, como generar nuevos datasets a partir de documentos (tarea que se puede automatizar en un pipeline). Hemos usado tanto datasets descargados como generados propios. Es conveniente evaluar con varios datasets de diferentes fuentes y ámbitos para que nuestro sistema sea robusto.

c. Tipos de evaluación usados:

LlamaIndex tiene varios módulos destinados a evaluar diversos aspectos del RAG:

- Faithfulness: evalúa si una respuesta concuerda con la información de los nodos en los que se ha basado (i.e. alucinaciones). Nos devuelve un score que puede ser 1 o 0.

- Relevancy: evalúa si la respuesta + nodos fuente concuerdan con la query. Útil para medir si la query ha sido respondida con la respuesta. Nos devuelve un score que puede ser 1 o 0.

- Correctness: evalúa la relevancia y exactitud de una respuesta generada comparándola con una respuesta de referencia. Nos devuelve un score del 1 al 5, y un feedback en el que explica el resultado de la evaluación.

- Semantic: calcula la similaridad entre los embeddings de la respuesta generada y de una respuesta de referencia. Nos devuelve un score del 0 al 1, y si pasa el umbral, que por defecto es 0.8.

- Guidelines: evalúa si en la respuesta generada se están cumpliendo las guidelines especificadas. Nos devuelve un score que puede ser 1 o 0, y un feedback en el que explica el resultado de la evaluación.

- Retriever: evalúa un retriever, comparando los documentos que obtiene para cada query dada con los correspondientes documentos de referencia de cada query. Nos devuelve los resultados de las métricas que le indiquemos, que en este caso son hit_rate, mrr, precision, recall, ap, ndcg.

==========
Métricas: si queremos podemos meter un apartado explicando métricas, en vez de comentarlas brevemente en otros apartados. Ocuparía más espacio.
==========

d. Otras posibilidades a considerar:

A parte de los aspectos de la evaluación cubiertos en este trabajo, existen otras posibilidades que se pueden considerar si se quiere extender o mejorar esta parte, como son:

- librerías de terceros integradas en llamaindex: existe una gran cantidad de librerías de terceros destinadas a la evaluación (UpTrain, DeepEval, etc.), que nos permiten evaluar aún más aspectos (summarization, bias, etc.), automatizar procesos de forma sencilla, ver gráficas, etc.
- Evaluación del coste del sistema, teniendo en cuenta número de tokens y modelos utilizados.
- Ensamblado de métricas: se pueden conseguir resultados similares a los que se obtienen usando de un modelo caro como GPT-4 en la evaluación, mediante técnicas de ensamblado de señales más débiles (exact match, F1, ROUGE, BLEU, BERT-NLI and BERT-similarity). Se pueden considerar para tener evaluaciones baratas para un uso más extendido a lo largo del proceso de desarrollo, que nos permita darnos cuenta rápido de cambios en el rendimiento nuestro sistema.

e. Descripción del código:

Para la evaluación hemos creado tres archivos de código:

- dataset_downloader.py: descarga datasets de llamahub destinados a la tarea de evaluación.

- eval_batch_multiple_evaluations: ejecuta múltiples evaluadores simultáneamente sobre un modelo.
    - definimos la key de OpenAI como una variable de entorno. Esto nos permitirá hacer llamadas a modelos de OpenAI.
    - definimos el LLM “gold” que usaremos para evaluar nuestro modelo. En este caso GPT-4 (Tener en cuenta que las llamadas a este modelo son caras con respecto a otros, así que para pruebas podemos usar otros más baratos).
    - definimos los evaluadores. Los evaluadores usados en conjunto en este archivo pueden evaluar faithfulness, relevancy, correctness, semantic, y guidelines. Para las guidelines definimos primero las guidelines en un diccionario y a continuación creamos un evaluador por guideline. A todos les pasamos por parámetro el LLM “gold” del paso previo, menos al de semantic, que evalúa

similitud entre embeddings y puede pasársele un modelo de embeddings o que coja el que haya en Settings.

- cargamos de un archivo el dataset para evaluación que vayamos a usar, y sus correspondientes documentos. Alternativamente podemos generar un dataset a raíz de unos documentos dados, usando DatasetGenerator.
- del dataset sacamos las queries y las respuestas de refencia. Opcionalmente podemos reducir su número para ahorrar dinero/tiempo en caso de que el dataset sea más grande de lo que queremos.
- definimos el modelo a evaluar.
- inicializamos un BatchEvalRunner con todos los evaluadores, y llamamos a su método aevaluate_queries pasándole nuestro modelo, las queries y las respuestas de referencia, para obtener los resultados de la evaluación.
- analizamos los resultados. Para ello definimos la función get_eval_results, que nos calcula para un evaluador dado, una score como la media de las scores de todas las queries. Podemos también ver resultados de queries concretas para ver otros atributos a parte del score, como por ejemplo el feedback.

- eval_retriever: evalúa un retriever.
- definimos la key de OpenAI como una variable de entorno. Esto nos permitirá hacer llamadas a modelos de OpenAI.
- cargamos de un archivo los documentos que vamos a usar.
- creamos los nodos a partir de los documentos, eligiendo el chunk_size que queramos.
- a partir de los nodos, creamos el vector store index y el retriever.
- si queremos, podemos mostrar los resultados de usar el retriever con una query dada.
- definimos el LLM que usaremos para generar un dataset de queries y contextos a partir de los nodos.
- aquí, o bien podemos generar un dataset de parejas de queries y contextos usando la función generate_question_context_pairs, y luego guardarlo en un archivo, o bien podemos cargar dicho dataset de un archivo de uno que hayamos generado y guardado anteriormente. Es muy recomendable generarlos solo una vez y guardarlos, ya que cuesta tiempo y, dependiendo del modelo usado para generarlos, dinero, el generarlos.
- inicializamos el evaluador, pasándole las métricas que queremos y el retriever.
- si queremos podemos obtener los resultados para una query concreta.
- obtenemos los resultados para todo el dataset.
- definimos una función que toma los resultados y nos devuelve un dataframe con la media de los resultados de cada métrica, y la usamos para mostrar los resultados medios de cada métrica.