Multimodal RAG with CLIP and Milvus

The project implements a Multimodal RAG pipeline that uses CLIP, Milvus, gpt-4, and ragas. This report sums up the reasoning behind such tool selection and possible ways to change or improve, depending on certain clarifications of the initial technical requirements.

Pipeline steps:

Scraping The Batch

- To scrape articles **BeautifulSoup** library was used.
- In order to properly separate articles and store the associated paragraphs and images, certain logic is presented right in the scraping stage instead of postprocessing.
- *Possible changes:* to have a safer way to debug and process data in the future, we could store raw scraped info with a tag and only after that add a specific processing step, which would handle what needs to go through the embedding process. Such an approach may help debug problems and fix issues with data without spending time on rescraping.

Creating embeddings

- In order to store embeddings of both images and texts, OpenAI's CLIP model *clip-vit-base-patch32* is used.
- Technically, this stage was the most questionable for me, as it is not clear if a user is allowed to search for pictures only or if the pictures are supposed to be returned next to articles only, and there is no need to do an overhead. I went with the second solution as there was no specification about the first one.
- But at the same time, multimodal RAG may be done in more ways than one presented. In my specific implementation, images are returned to the UI only if they are associated with specific articles, but

CLIP actually allows us to search for images, as it allows us to store both images and text in the same embedding space.
- *Note:* I have also tried grounding both modalities to texts by generating captions to images by using the BLIP model. This approach was not successful as the BLIP model generated very general descriptions, and if we were to allow users to search for images by description, we would need to be more specific. But at the same time, such an approach may be beneficial as we could use a powerful text embedding model.
- Text embedding is done on the paragraph level. If a text chunk has 700 or fewer characters, it is considered a paragraph. Not too big chunks of text are embedded, so we are able to search effectively.

Vector storage management

- I have selected the Milvus vector database as it was easy to set up, and it was the first time for me to use it, so I wanted to try something new.
- Similarity search was done by using L2 distance.
- I have also used unique identifiers to spot later paragraphs that were related to the same article.
- Reranking is performed on the returned 10 examples.

Handling answers based on the context via LLM

- Streamlit is used for UI.
- The LLM module takes the retrieved context and generates an answer only based on this. Prompt specifies that in case of context doesn’t have an answer, the model needs to be honest and provide user info that the answer is not present in the context.

Evaluating

- While using the RAG app, all requests and answers are stored in the log file. Later, this info is used for evaluation.
- Evaluation is done by the RAGAS lib based on answer relevancy and faithfulness.

- Answer relevance checks how relatable the answer is to the user query itself.
- Faithfulness tells if claims made by the LLM can be supported by context that was used.
- Since we do not have ground truth, I do not use such metrics as correctness, but it would be desirable.