NLP project

What we have now:

- extracted data from PubMed merged into single master sheet
- developed UI in Flask Framework - locally over docker
- abstract embedding:
  - split into chunks, each consisting of 3 sentences (kind of manually, at least not with the use of LangChain etc.)
  - each chunk is embedded separately using BERT from Hugging Face (mean pooling to create vectors of even length)
  - embeddings are stored in .csv
  - .csv used to populate vector database
  - BUT this is done only on first 100 articles for now
- Vector Database - Milvus
  - locally in a Docker container
- relevant document retrieval:
  - search of relevant chunk - by embedding the query using cosine similarity
  - top k chunks are returned

What we said is future work:

- key to success is experimenting with chunks (!)
- hybrid search → for keywords and semantic search (e.g. OpenSearch has it)
- LLM could improve the performance of Information Retrieval (IR) as we can e.g. choose best k with IR and ask LLM to choose best from this best k
- create Test Data should have 20-100 examples (can be generated with ChatGPT)
- use more data than only this testing part
- choose LLM
- add prompts generation to use LLM
- integrate with UI
- evaluation ?

Improvements:

- automate data acquisition
- chunking - other ideas
- other db ?
- LangChain

Ideas to check:

- Milvus - czy to się da w nim hybrid search ?
- LangChain jak działa - na nasze potrzeby
- jak to z UI połączyć sensownie, czy z tym co mamy to się da ?

https://milvus.io/docs/dynamic_schema.md

z tego co rozumiem, to tu ten search jest jako wartości tabeli czy spełniające liczbowe warunki, a u nas to bardziej byśmy chcieli aby aby słowa wyszukiwało w tekście, hmm czy się da ???

"you can perform a hybrid search by specifying a boolean expression that filters the scalar field or the primary key field"

brzmi jak nie to co chcemy

ale jest boolean, który można użyć: VARCHAR not in ["str1", "str2"] ?

trzeba by to spróbować umieścić do hybrid

https://milvus.io/docs/boolean.md

- jak działa to co Jacek zrobił
- czy da się to wgl puścić - M mówił, że umie
  - a na wszystkich danych?

Znaleziska:

https://github.com/llmware-ai/llmware

- tylko jeśli Milvus, to ma sens, jak zmienimy na OpenSearch, to LLMware nie wspiera
- no i też pytanie co wsadzać, jakie dane, bo chyba muszą być już po chunking etc.

Assignment 4

differences - what to change to implement in the project

Hugging Face - source of the llama 2 model

Pinecone - vector database

How does the pipeline look like?

Part 1. Data Preparation

1. Download data → all HTML files from the links on the webpage of LangChain documentation
2. ReadTheDocsLoader → function from LangChain that allows loading the documents from directory
3. Chunking:
   - Llama 2: maximum token limit is 4096
   - instructions how to count chunk size
   - use LangChain splitters
   - best practices which splitter, what parameters' values and why
4. Apply chunking to all documents and save chunks in logical way + unique ID to each chunk to know from which document they are + example how should it look like
5. dictionary with chunks into pd.DataFrame

6. transform chunks into vector embeddings using Hugging Face & LangChain
   - these embeddings will be used for similarity search between query & chunks to retrieve the most relevant chunks
   - they are using: sentence-transformers/all-MiniLM-L6-v2 model for creating the embeddings
   - Hugging Face Embeddings
7. Pinecone - setup environment (including metric)
8. Pinecone - loading processed (in batches) data into the database

Text Generation Pipeline

9. now we need a way to generate the answer from the retrieved chunks
   - text-generation pipeline from Hugging Face
   - enable quantization using bitsandbytes library
   - load correct model configuration from transformers library

->initialize a test-generator pipeline with HF that uses the Llama2 model to generale some text, given some input. We will then use this pipeline inside LangChain to build our Question-Answering system
-> for this pipeline we must initialize a language model and tokenizer for the language model

Question Answering Chain

-> for RAG in LangChain we need to initialize RetrievalQA or RetrievalQAWithSourcesChain
-> to initialize them, we need a LLM and Pinecone index
-> when we try to use only Llama2, we get some answer, but not well connected to our dataset, therefore we need to use our rag pipeline and only then ask Llama2 to answer our questions

Conversational Retrieval Chain

-> aim: extend the already created retrieval chain to be able to remember the previous questions and answer the current questions by looking at the previous context
-> use LangChain to create a conversational memory => ConversationalRetrievalChain
-> it takes chat history (list of messages) and new question and returns an answer to the new one
-> detailed description how it works - in assignment file

Task 2: Advanced RAG Techniques and Evaluation

Can also be useful for our project, as we have to also evaluate it

-> or maybe not, they focus on already annotated dataset - or maybe also can have similar???
-> this time they use another vector database - Chroma

Subtask 1. Build Contextual Compression in LangChain

-> split documents using TextSplitter from task1, embed with embedding model from previous task
-> make simple RAG pipeline that works on top of Chroma retriever
-> apply rag_simple pipeline to all questions in corpus and accumulate the answers
-> use ContextualCompressionRetriever to extract only the relevant information from documents, for that LLMChainExtractor can be useful
-> create context compressor RAG pipeline

Subtask 2.2. Evaluate

-> they explore BLEU, ROUGE, BERTScore metrics
-> HF has implementation for all those metrics
-> if the answers contain multiple lines, merge them to single string
-> in the assignment there are more variants of those metrics described