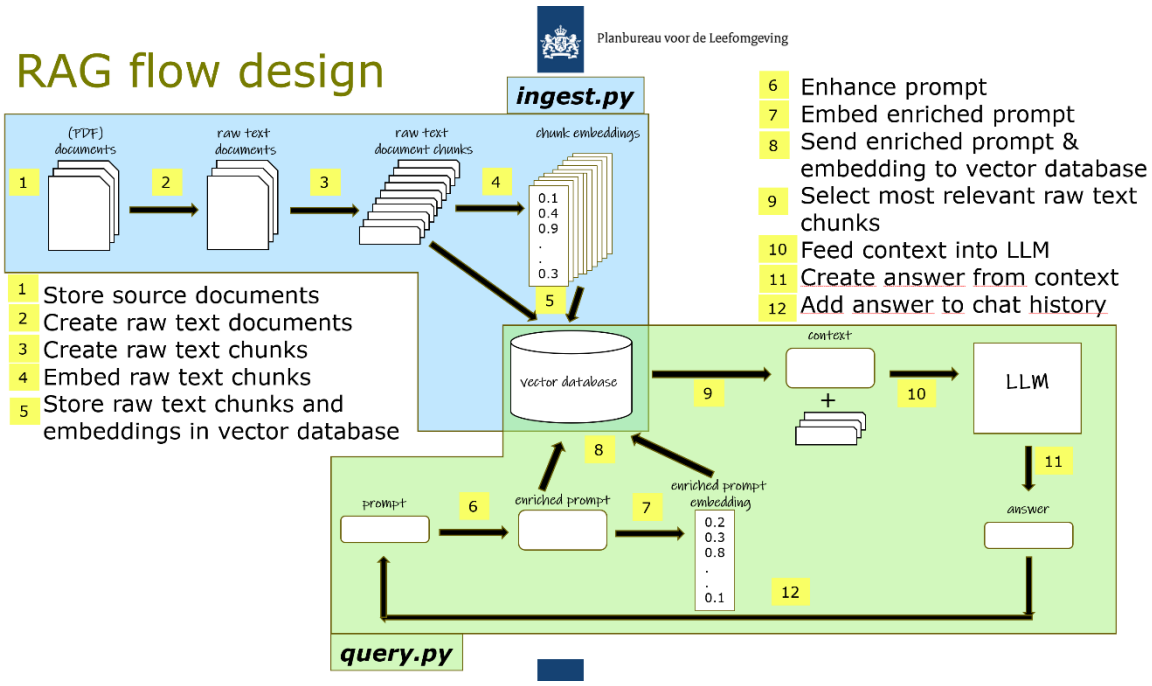
Chat with your docs!

A RAG (Retrieval Augmented Generation) setup for further exploration of chatting to company documents



How to use this repo

! This repo is tested on a Windows platform

Preparation

1. Clone this repo to a folder of your choice
2. Create a subfolder vector_stores in the root folder of the cloned repo
3. Create a file .env and enter your OpenAI API key in the first line of this file :
   OPENAI_API_KEY="sk-....."
   Save and close the .env file

- In case you don't have an OpenAI API key yet, you can obtain one here: https://platform.openai.com/account/api-keys
- Click on + Create new secret key
- Enter an identifier name (optional) and click on Create secret key

Conda virtual environment setup

1. Open an Anaconda prompt or other command prompt
2. Go to the root folder of the project and create a Python environment with conda using commandline command
   `conda env create -f appl-docchat.yml`
   NB: The name of the environment is appl-docchat by default. It can be changed to a name of your choice in the first line of the yml file
3. Activate this environment using commandline command
   `conda activate appl-docchat`

Pip virtual environment setup

1. Open an Anaconda prompt or other command prompt
2. Go to the root folder of the project and create a Python environment with pip using commandline command
   `python -m venv venv`
   This will create a basic virtual environment folder named venv in the root of your project folder NB: The chosen name of the environment is here appl-docchat. It can be changed to a name of your choice
3. Activate this environment using commandline command
   `venv\Scripts\activate`
4. All required packages can now be installed with command line command
   `pip install -r requirements.txt`

Ingesting documents

The file ingest.py can be used to vectorize all documents in a chosen folder and store the vectors and texts in a vector database for later use.
Execution is done in the activated virtual environment using commandline command:
`python ingest.py`

Querying documents

The file query.py can be used to query any folder with documents, provided that the associated vector database exists.
Execution is done in the activated virtual environment using commandline command:
`python query.py`

Ingesting and querying documents through a Streamlit User Interface

The functionalities described above can also be used through a User Interface.
The UI can be started by using commandline command:
`streamlit run streamlit_app.py`
When this command is used, a browser session will open automatically

Ingesting and querying documents through a Flask User Interface

The functionalities described above can also be used through a Flask User Interface. The Flask UI is tailored for future use in production and contains more insight into the chunks (used) and also contains user admin functionality among others.
For a more detailed description and installation, see the readme file under flask_app

Evaluation of Question Answer results

The file evaluate.py can be used to evaluate the generated answers for a list of questions, provided that the file eval.json exists, containing not only the list of questions but also the related list of desired answers (ground truth).
Evaluation is done in the activated virtual environment using commandline command:

```
python evaluate.py
```

Monitoring the evaluation results through a Streamlit User Interface

All evaluation results can be viewed by using a dedicated User Interface.
This evaluation UI can be started by using commandline command:

```
streamlit run streamlit_evaluate.py
```

When this command is used, a browser session will open automatically

User Stories for improvements

Every user story below has an indication whether it extends the functionality of the application (FUNC), or is related to optimize the results (EVAL).
Furthermore, user stories are divided into 2 groups:

- RESEARCH: RESEARCH user stories are not meant to change any code but require research and prepare for an actual BUILD task.
- BUILD: BUILD user stories change the code. They add functionality to the application or are performance related

All user stories have an association with one of the steps (1 - 12) in the pipeline pictured above and are written from the perspective of either the user of the application, or the developer of the application.

1. EVAL, BUILD, evaluation: As a developer I want to create a benchmark of the performance of the application. It is necessary to gather all questions and

answers available and update eval.json to get a complete set ready for evaluation. Add the associated folders and documents to the repo.

2. FUNC, BUILD, Ingestion (1): As a user I want to synchronize the (Chroma) vector database with the document folder I am using. If the document folder has changed (extra file(s) or deleted file(s)), either add extra documents to the vector database or delete documents from the vector database. For inspiration: contact Rob and/or look at https://python.langchain.com/docs/integrations/vectorstores/chroma#update-and-delete
3. FUNC, BUILD, Ingestion (2): As a user I want to query not only PDF's, but also other file types with text, like Word documents, plain text files, and html pages. For inspiration, see list of loaders in LangChain.
4. FUNC, RESEARCH, Ingestion (2) & Ingestion (3): As a developer I want to add metadata to the document chunks that can have added value, like authors, title, year of publication and summary of the document. Research what is the best way to do this, for ChromaDb.
5. FUNC, BUILD, Ingestion (2) & Ingestion (3): Implement the addition of a document summary and the filename to the metadata of the document
6. EVAL, RESEARCH, Ingestion (3): As a developer I want to determine the optimal settings for fixed-size chunking. Values of chunksize, chunk overlap and maximum number of chunks to add to context are available in see settings.py. Test the application performance of alternative settings of CHUNK_SIZE, CHUNK_OVERLAP and CHUNK_K.
7. EVAL, BUILD, Ingestion (3): As a developer I want to use an optimal set of chunks. Can we implement content-aware text chunking, keeping related content together in one chunk (up to a maximum chunk size)? For inspiration: https://github.com/nlmatics/llmsherpa#layoutpdfreader
8. EVAL, RESEARCH, Ingestion (3): As a developer I want to compare the performance of content-aware chunking with fixed-size chunking
9. FUNC, BUILD, Ingestion (4) & Retrieval (7): As a developer I want to generate the best answers to the user questions. The application currently uses OpenAI's text-embedding-ada-002 as embedding model. It is not the best one according to huggingface MTEB embedding leaderboard. See https://huggingface.co/spaces/mteb/leaderboard. Implement an alternative open-source, on-premise, embedding model.
10. EVAL, RESEARCH, Ingestion (4) & Retrieval (7): As a developer I want to evaluate the impact on performance from switching to the implemented open-source on-premise embedding model.
11. EVAL, BUILD, Retrieval (9): As a user I don't want the chatbot to hallucinate. Add a lower bound for the similarity score to filter out text chunks. If none of

the text chunks reaches the lower bound value, answer “I don’t know” (in the language of the user)?

12. FUNC, BUILD, Retrieval (9): As a user I want to know which returned chunks are the preferred ones. Add the similarity score of each chunk to the sources in the response and rank each chunk according to the similarity score
13. EVAL, BUILD, Retrieval (11): As a developer I want to evaluate the impact of switching from LLM gpt 3.5 to gpt 4
14. FUNC, BUILD, Retrieval (11): As a developer I want to have the option of using the application for documents that must be kept private, by setting alternative values in settings.py. Implement an alternative downloadable (open source) LLM like Llama 2, Zephyr-7B, Mistral 7B
15. EVAL, RESEARCH, Retrieval (11): As a developer I want to test the performance of the implemented open-source, on-premise, LLM.
16. FUNC, BUILD, Ingestion (4) & Retrieval (7) & Retrieval (11): As a user I want the option of starting a conversation with a summary of the document(s) in the folder. For inspiration,
    see https://python.langchain.com/docs/use_cases/summarization
17. FUNC, RESEARCH: Retrieval (6) & Retrieval (10) & Retrieval (11) & Retrieval (12): As a developer I want to know if the currently implemented loguru logger is suitable for logging all questions, retrieved contexts and generated answers to disk per session, with indication of which source folder was used.
18. FUNC, BUILD, Retrieval (6) & Retrieval (10) & Retrieval (11) & Retrieval (12): As a developer I want to log all questions, retrieved contexts and generated answers to disk per session, with indication of which source folder was used.
19. FUNC, RESEARCH, evaluation: As a developer I want to know if we are missing other valuable performance metrics. Research added value of additional metrics like ROUGE and BLEU and what is needed to implement them. For inspiration, see for
    example https://medium.com/@raphael.mansuy/evaluating-the-performance-of-natural-language-processing-nlp-models-can-be-challenging-ce6f62c07c35
20. FUNC, BUILD, Retrieval (11): As a user I want the application to use tools to interact with the world (e.g. use service APIs) if it is needed to generate a more reliable response and avoid hallucinations.
21. FUNC, RESEARCH, Retrieval (11): As a developer I want to learn how to use Agents and Tools functionality and how to add it to the existing code to create more powerful and flexible LLM based applications.
22. FUNC, BUILD, Retrieval (11): As a developer I want to add Agents and Tools functionality to the code, that can use Agents based on various LLMs, and Tools selectable from a predefined set.

23. FUNC, BUILD, Retrieval (11) (SoilWise): As a developer I want to add a Tool for geocoding, allowing the LLM to work with spatial coordinates and precise locations (other than what it might have seen during training).
24. FUNC, BUILD, Retrieval (11) (SoilWise): As a developer I want to add a Tool to access the soilgrids API, allowing the LLM to access more detailed soil information for locations, in order to generate an improved response.
25. EVAL, RESEARCH, Retrieval (11): As a developer I want to evaluate the impact and costs of using Agents and Tools with various LLMs (open and closed models).

User stories to work on after the pressure cooker session: User Interface: As a user I want to have the option of using the application for documents that must be kept private, by using a checkbox in the user interface. This will make the application use an implemented on-premise embedding model and an implemented on-premise LLM.

References

This repo is mainly inspired by:

- https://docs.streamlit.io/
- https://docs.langchain.com/docs/
- https://blog.langchain.dev/tutorial-chatgpt-over-your-data/
- https://github.com/PatrickKalkman/python-docuvortex/tree/master
- https://blog.langchain.dev/evaluating-rag-pipelines-with-ragas-langsmith/
- https://github.com/explodinggradients/ragas