Sintef Competence Query Project

Åsmund Mjøs

Summer 2025

Abstract

Competence query search is an important upcoming application of Retrieval-Augmented Generation. I tested out different reranking techniques on a MRST based dataset, and found great results using semantic-match matrices. Using the most frequent bigrams to represent articles is an old technique, that mixed with modern embedding models, has great results. I present a full competence query program, my experiences in this field, and tips on how to adjust the program to general competence query searches.

1 Introduction

In this report, we will discuss both good and bad qualities of different techniques used throughout the project, such as query transformation through knowledge-based keyword generation, relevance scores through semantic match matrices, and text generation through cluster-seperation. We will also present the MRST competence query application, how to use it, and how it's been built. The goal of the project is to build an application that can reroute you to MRST researchers with the right and relevant knowledge, but just as much to build insight on, and test different RAG applications and methods.

2 Theory

Retrieval-Augmented Generation is a technique used to strengthen large language models' understanding of specific context, without training the model on data. In the simplest RAG applications, often called naive RAG, you embed the user's query, and retrieve relevant documents based on a similarity score called cosine similarity between that embedded query vector and the embedded document vectors. The relevant documents are fed into the large language model together with the user's query, such that the provided information

helps the language model assist the user. The idea is to input the documents with the highest semantic similarity as context for the language model. To strengthen naive RAG, you can implement techniques to rewrite the query before retrieval or re-rank the documents after retrieval.

3 Methods

It's important to note that the application is not meant to be like a chatbot at all. The sole purpose of the application is to generate an answer on who to contact about a certain topic, based on a knowledge database consisting of scientific articles, and academic textbooks.

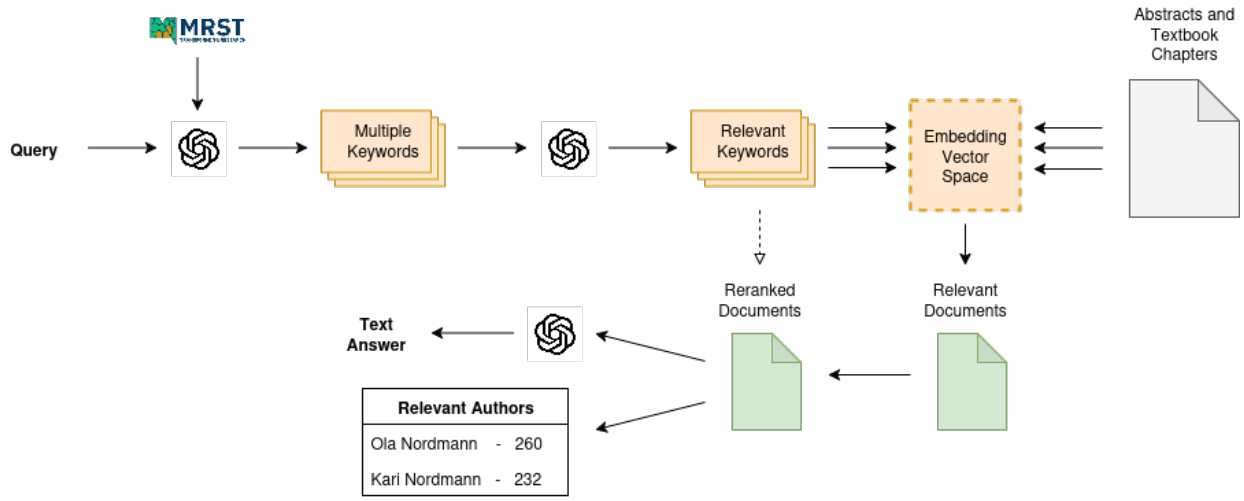


Figure 1: Pipeline of the Application

3.1 LangGraph Workflow

Figure 1 shows the general execution of the program. We use LangGraph, a python library developed by LangChain, to create a workflow through a graph. Writing the program with the LangGraph tool is not really necessary for the execution of the program, but done for debugging purposes, as you can follow a graph execution easily on the internet through LangSmith, LangChain's observability, debugging, and evaluation platform. Figure 2 shows a mermaid image of the pipeline of the MRST competence query application.

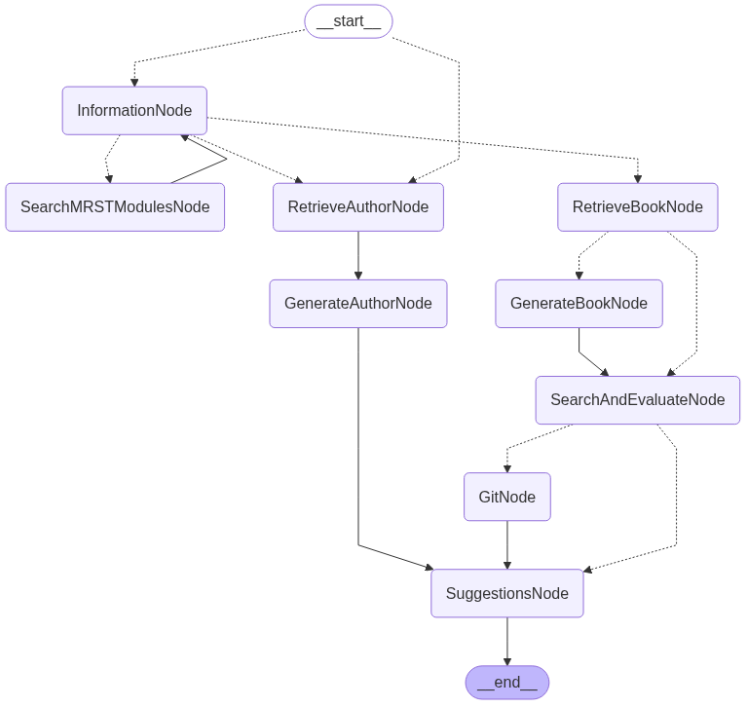


Figure 2: Mermaid image of the graph behind the MRST Competence Query. Solid lines represent mandatory paths, while dashed lines represent conditional paths.

The program starts its execution at InformationNode. Here, information like researchers mentioned in the user's query is extracted, and relevant keywords to the user's problem are generated. If the language model used to generate keywords finds it relevant to first read up on some specific modules in the MRST module page in order to generate specific mrst-keywords, it can. The specific module page is then scraped and fed back into the language model before it generates a set of keywords. This is represented by the node SearchMRSTModulesNode (although the node itself is never used as all logic is handled inside InformationNode)

If there are detected researchers' names in the users' query, the program moves to RetrieveAuthorNode where articles and textbook chapters written by the authors in the query are retrieved. The program then moves to the GenerateAuthorNode, where these documents are used to generate a text answer giving the user information about relevant work done by the researchers in their query.

Otherwise, the program moves to RetrieveBookNode, where relevant chapters from the MRST textbook is retrieved.

- If there are no chapters in the textbook with a high enough similarity score, the program moves to SearchAndEvaluateNode, where relevant articles are retrieved, evaluated, and used to generate a relevancy score for each author of the relevant articles. The Language Model also generates a text using the retrieved articles, to state who the user can contact about certain subtopics related to the user's query.

- If there are chapters in the textbook with higher cosine similarity than a certain threshold, the program takes a detour to GenerateBookNode, where the relevant chapters are used to generate a text answer giving the user information about where in the textbook the user might find relevant information, and who the authors of those relevant chapters are. The program then moves to SearchAndEvaluateNode.

The user has the option to choose wether or not it wants the program to search through the MRST repository for relevant files, and then list the authors with the most commits in relevant folders. This is handled in the GitNode.

Both the author route or the non-author route meet at the SuggestionsNode. Here relevant suggestions for what the user should search next is generated based on all of the generated information throughout the program. All the information gathered throughout the run is then presented to the user as we exit the graph.

3.2 Language Model

Throughout the program, we use language models both for text generation and for different kinds of logical tasks. We use OpenAI through Langchain. Loosely speaking, we use the model gpt-4.1-nano for text generation, while we use the models gpt-4o-mini and gpt-4.1-mini for logic based tasks.

3.3 Embedding

3.3.1 Embedding Model

The MRST website has an article base of around 1000 articles. These articles are embedded to vectors of dimension 768. I've implemented both embeddings using the general purpose embedding model `sentence-transformers/all-mpnet-base-v2` and the more specific model `allenai-specter`.

3.3.2 Chunking [Indexing]

In this project we have two completely seperate datasets, since the documents in the two convey different information. One consiting of the MRST textbooks, and one consisting of the scientific publications on the MRST publications page

For the articles, there are two different datasets, one where each paper is represented by its title and abstract. This is not ideal, since some abstracts are much more general in its language than others, however it keeps the dataset very structured and concise. In the other article dataset, we included the whole articles. In this dataset, we seperated each article into multiple chunks, so into smaller documents.

For the MRST textbooks, we seperated the textbooks into the different subchapters. This is not ideal, since there is much variation in how long each chunk/document then becomes.

Both the embedding models used throughout the project embed documents by separating the text into tokens, and embeds each token, and returns the document embedding as some sort of average of these embeddings. The ideal way to chunk documents is to use chunking sizes so that each document consists of 100-500 tokens. Figure 3 shows how the abstracts fall quite perfectly into this document size.
We also built a dataset consisting of

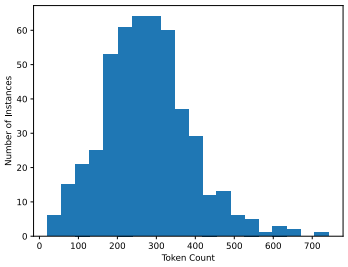


Figure 3: Histogram over token sizes for the scientific abstracts

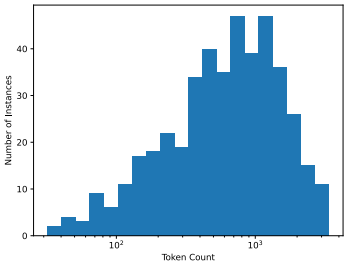


Figure 4: Histogram over token sizes for the subchapters in the MRST textbooks

YOU GOTTA INCLUDE A PICTURE OF THE TOKEN SIZES FOR CHUNKED UP ARTICLES AS WELL

3.3.3 Clustering

When the user presents a problem, the relevant documents are clustered into different sub-topics. These competence areas are presented to the user, and the clusters are presented as suggestions for further searches. This is done mainly because of 2 things: to rewrite the query to better understand the user's problem, and to let the user help choose what's relevant information. For our clustering technique, we ended up using hdbscan. I also tested using kmeans as the clustering algorithm, which gave results that gave much less sense.

3.4 Semantic Match Matrices

Based on the users' query, there are generated n keywords. If we assume that they're all relevant and precisely capture different topics related to the user's query, semantic similarity to these keywords seems like a natural way to rank the relevance of different papers. One method tested througout this project is to generate a list of k keywords related to each article, and calculate cosine similarity for each permutation, such that we get a $n \times k$ sized matrix of cosine similarity values. I tested 3 ways to calculate one final similarity score for this semantic match matrix. If we let x_{ij} represent the cosine similarity between query-keyword i and article keyword j, we could get the similarity score S from one of these 3 methods.

$$S = \frac{1}{n}\sum_{i=1}^{n}\max_{j=1}^{k}\{x_{ij}\} \qquad (1) \qquad\qquad S = \frac{1}{k}\sum_{j=1}^{k}\max_{i=1}^{n}\{x_{ij}\} \qquad (2) \qquad\qquad S = \frac{1}{nk}\sum_{i=1}^{n}\sum_{j=1}^{k}x_{ij} \qquad (3)$$

Equation 1 shows a method where you take the average of the maximum cosine similarity for each query generated keyword. This is done to try and best capture the different topics mentioned in the query. Equation 2 shows a second method, where you instead take the average of the maximum cosine similarity for each article generated keyword. This is done to try and best capture the different topics mentioned in the article. Equation 3 shows the final method, where you simply take the average over the whole matrix.

3.5 Chapter Graphs

The user also has the option to choose wether or not they want to generate chapter graphs for the relevant textbook chapters. Figure 5 shows an example of such an image. The user can then get an overview of relevant chapters, so they can "check" the relevance of the results by themselves.

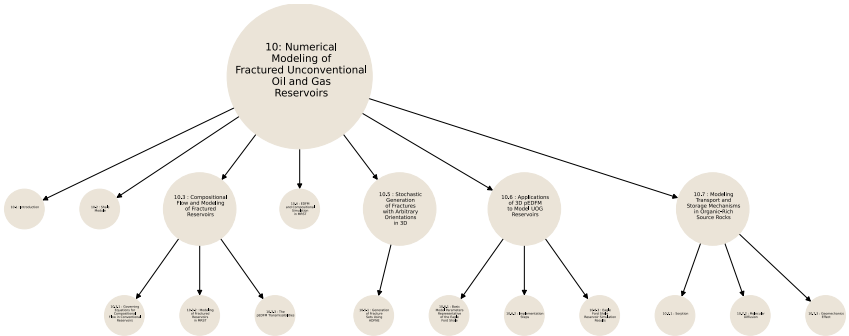


Figure 5: Example of Chapter Graph for chapter 10. If nodes are considered relevant, they will be showcased in orange. Only chapter graphs with relevant subchapters will be generated and presented to the user.

4 Results

4.1 Keywords Generated

Table 1: Keywords Generated before filtering

What can you tell me about eor methods	**Who should I contact about coupled flow and geomechanics for CO2**	**Who should I contact about polymer flooding**
enhanced oil recovery (EOR) polymer injection surfactant injection mobility ratio viscosity permeability reduction relative permeability curves capillary number black-oil model optimization in EOR	coupled flow geomechanics CO2 storage MRST-co2lab Discrete Fracture Matrix (DFM) fractured media carbon capture and storage (CCS)	polymer flooding enhanced oil recovery EOR techniques viscosity reduction permeability reduction surfactant injection MRST modules
Who should I call about linear solvers	**Who in MRST should I reach out to about Virtual Element Methods**	**Who should I reach out to about ad-blackoil**
linear solvers numerical methods computational mathematics MRST linear solvers	Virtual Element Methods VEM Poisson-type flow equations higher-order methods unstructured grids numerical analysis finite element methods finite difference methods	ad-blackoil black oil modeling reservoir simulation MRST modules SINTEF research

Table 2: Keywords generated after filtering

What can you tell me about eor methods	**Who should I contact about coupled flow and geomechanics for CO2**	**Who should I contact about polymer flooding**
enhanced oil recovery (EOR) polymer injection surfactant injection mobility ratio permeability reduction relative permeability curves capillary number optimization in EOR	coupled flow geomechanics CO2 storage MRST-co2lab Discrete Fracture Matrix (DFM) fractured media carbon capture and storage (CCS)	polymer flooding enhanced oil recovery
Who should I call about linear solvers	**Who in MRST should I reach out to about Virtual Element Methods**	**Who should I reach out to about ad-blackoil**
linear solvers MRST linear solvers	Virtual Element Methods VEM Poisson-type flow equations	ad-blackoil black oil modeling

Table 2 shows keywords generated after llm filtering, while Table 1 shows the keywords generated before filtering. The method seems to be working pretty well, but as seen in the keywords for the query 'Who should I call about linear solvers', we sometimes get few keywords, which might result in missing relevant documents, and some keywords are too general, and not specific enough. The keyword generator often includes general expressions like SINTEF, MRST or Reservoir simulation in the keywords, and the filtering removes quite a few, but not all of these general queries. The filtering technique consists of letting the llm grade the specificness of the techniques. One thing worth noticing is that this technique results in very different lengths of lists.

4.2 Semantic Match Matrices

Table 3 and 4 shows the top 5 papers by the 3 different techniques for 2 different questions. The input was the top 20 articles from the first retrieval step. I have very little idea which list is the best, so I thought I'd present at least some data, so that experienced researchers maybe could get a feel for which method works/does not work at all.

Table 3: Articles retrieved for question: What can you tell me about eor methods

Average Query-Keyword Score	Average Article-Keyword Score	Total Average
- A combined Markov Chain Monte Carlo and Levenberg–Marquardt inversion method for heterogeneous subsurface reservoir modeling - Numerical interpretation of SCAL data with automated history matching tool - A coupled geomechanics and reservoir simulator with a staggered grid finite difference method - Simulation of hybrid Microsphere-SmartWater recovery process for permeable carbonates - Well placement subclustering within partially oil-saturated flow units	- A combined Markov Chain Monte Carlo and Levenberg–Marquardt inversion method for heterogeneous subsurface reservoir modeling - Simulation of production performance incorporating temperature-dependent pre-Darcy flow in low-permeability reservoirs - Numerical interpretation of SCAL data with automated history matching tool - A coupled geomechanics and reservoir simulator with a staggered grid finite difference method - Characterizing the impact of heterogeneity on miscible gas injection	- Well placement subclustering within partially oil-saturated flow units - Numerical interpretation of SCAL data with automated history matching tool - A coupled geomechanics and reservoir simulator with a staggered grid finite difference method - A combined Markov Chain Monte Carlo and Levenberg–Marquardt inversion method for heterogeneous subsurface reservoir modeling - Simulation of hybrid Microsphere-SmartWater recovery process for permeable carbonates

Table 4: Articles retrieved for question: Who should I contact about coupled flow and geomechanics for CO2

Average Query-Keyword Score	Average Article-Keyword Score	Total Average
- Evaluating geophysical monitoring strategies for a CO2 storage project - An improved dual-porosity dual-permeability modeling workflow for representing nonplanar hydraulic fractures - Investigation of CO2 microbubble assisted carbon sequestration and gravity-induced microbubble ripening in low permeability reservoirs - Fluid flow characterization framework for naturally fractured reservoirs using small-scale fully explicit models - Detecting subsurface fluid leaks in real-time using injection and production rates	- Evaluating geophysical monitoring strategies for a CO2 storage project - Detecting subsurface fluid leaks in real-time using injection and production rates - Investigation of CO2 microbubble assisted carbon sequestration and gravity-induced microbubble ripening in low permeability reservoirs - Towards a nonlinear transfer function in dual porosity models - Assessing grid resolution impact on the reliability of CO2 plume modeling in saline aquifer storage sites	- Investigation of CO2 microbubble assisted carbon sequestration and gravity-induced microbubble ripening in low permeability reservoirs - Evaluating geophysical monitoring strategies for a CO2 storage project - An improved dual-porosity dual-permeability modeling workflow for representing nonplanar hydraulic fractures - Detecting subsurface fluid leaks in real-time using injection and production rates - Towards a nonlinear transfer function in dual porosity models

5 Discussion of Results

5.1 Dataset

5.1.1 Textbook Dataset

I decided to not include the introductory book in the dataset. This was done so that Knut-Andreas Lie, the chief scientist of MRST, would not always become the most relevant researcher. A more accurate representation could be to give this book a higher cosine similarity threshold, or give chapters from the introduction book lower weighting in the final relevance score, but this quickly becomes very specific to MRST, as I'd probably have to give Knut-Andreas Lie a different relevance formula.

5.1.2 Article Dataset

I generated the dataset by implementing different web scraping scripts based on the different journals websites. This worked, but took a lot of time, and for some of the journals I was prohibited from web scraping the abstracts, which means I only collected approximately half of the articles' abstracts. A more sophisticated way would be too web scrape google scholar based on the DOI, however google scholar does not like web scrapers. An option for other paper based competence query searches could be to use paid third-party APIs for scraping the google scholar site such as serpapi or Semanticscholar api.

5.1.3 Choice of embedding model

As stated in methods, both a general purpose embedding model and a specific-purpose embedding model was tested. My experience is that the general purpose model is more consistent, whilst the specific-purpose embedding model either hit very specific, or way too general. If there are released specific embedding models that are really well constructed and trained on big relevant datasets, I believe it could work well, but the specific model I tested called allenai/spectre did not outperform the general purpose models.

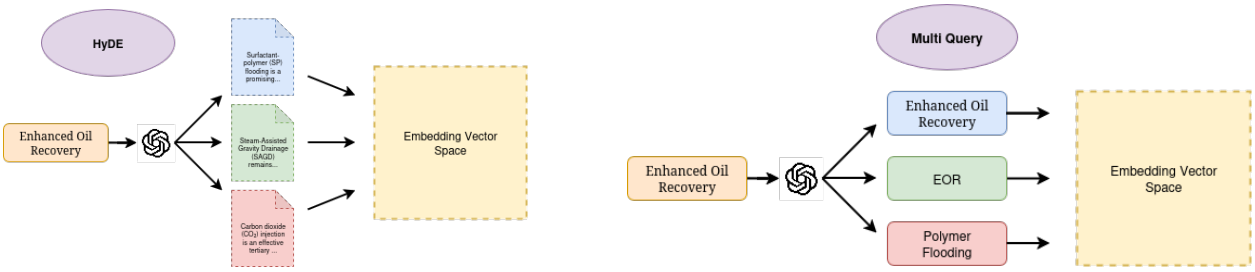
5.1.4 Using Pickle Files to Store Dataset

In this project I've used my own vector store instead of using common langchain vectorstores. I would recommend others to do the same, as it gives more flexibility, and it's really easy to make adjustments on different conditions for what types of documents to include when retrieving documents at certain times. I've saved the datasets as pandas dataframes loaded into pickle files.

5.2 Query Rewriting

5.2.1 Generating Keywords

Generating multiple keywords work mostly great, but have in general one flaw. If the user's query mentions two topics, and wants to learn who has expertize in the combination of the two, for example 'well modeling for CO2', generating keywords can broaden the search to both 'CO2' and 'well modeling', which might generate quite different answers than 'well modeling for CO2'.



(a) HyDE (b) Multi Query

Figure 6: Example of the two query transformation techniques tested on the query 'Enhanced Oil Recovery'

Since MRST is quite a specific toolbox, it helped a lot to have a specific context searching tool, for generating keywords. I would recommend creating your own context tool, if you are creating another competence query search, and choose query rewriting through keyword generation.

5.2.2 HyDE

I tested out both keyword generation and hypothetical document generation as query rewriting methods, and I felt keyword generation worked the best. You might have great success using the HyDE technique, but from my experiences, this needs quite specific document formats, and that is something one might not always have.

5.3 Reranking

5.3.1 Semantic Match Matrix

Scoring System

Based on my experience testing out the different semantic match matrix scoring systems, it's difficult to determine which of the 3 methods stated in section 3.4 is the superior similarity scoring method. Intuitively I find equation 1 the best, as it tries to capture the relevance of different topics relevant to the query. If the llm keyword generation doesn't work too well though, I find equation 2 the most promising, as it tries to capture the essence of the most common bigrams in the articles, which seems like a pretty consistent method. These thoughts are in no way properly tested, so there really is no scientific evidence. I added table 3 and 4, so it is possible for researchers with actual insight in the field to maybe get a feel for if you felt like one of the scoring systems worked very poorly, even though the data is very limited.

Article Keywords

Many articles have keywords listed. For the semantic match matrices, we generate keywords for each relevant article, as described in section 3.4, and it could be smart to use those keywords as the article-keywords. I tested out using these keywords instead of the most frequent bigrams on a small dataset, and had great results, but I failed to collect keywords for every article, which is why I decided to go with the most frequent bigrams. Bigrams seem to work well on a big scale, and is very general and automatically works for many datasets. One thing worth noticing about the bigram technique is that for many abstracts, there might only be one or two bigrams that actually get repeated, however when generating the most common bigrams, the result of applying most common bigrams seem to represent the abstracts well.

5.3.2 Other Rerankers

The standard choice for reranking, is to use large language models or Cross-Encoders. Cross-Encoders are individual language models that are trained to return a score informing us how relevant two queries are. They

therefore have 2 inputs, 2 queries that it tries to score how much have in common. Cross-Encoders are very powerful, especially for shorter queries, but are computationally expensive. I tested out one cross-encoder, and ranking just one article based on a users query took approximately 2 seconds. If we use abstracts instead, it took 1 second, so doing this on maybe 200 articles seem like a bad idea. This is the reason we used frequent bigrams, since the computational time is negligible compared to using language models or cross encoders to rerank the articles.

5.4 Clustering

Clustering the relevant articles, and presenting the clusters to the user is another way to let the user choose what's relevant. I found the most frequent bigrams for each cluster, and got an llm to generate a cluster description for each cluster based on the top bigrams. I found hdbscan to work great for this task. When reducing the clusters down to 2 dimensions using umap, most clusters seem very intuitive. Some of the clusters generated very relevant names, and got described well, while some generated badly named descriptions. I let the clusters be suggestions for further searches. This means that the the next search query is the generated cluster name. Another interesting approach could be to use the documents already retrieved in this cluster for further clustering, and still search in the textbooks on the generated cluster name as query.

5.5 Text Generation

When the program finds relevant textbook chapters, and tries to generate text, I only input the retrieved chapters (simply the orange node), but I also tried inputting a little context (the parent node), which worked reasonably well, sometimes better, and sometimes a bit worse (as the language model got too much information). You could maybe try and first get an llm to summarise the relevant chapters and their context first, before inputting that summarisation into the language model as context, but I didn't test that out.

6 Implementations for other competence query searches

Most logic and code should be pretty general, and it should be possible to transfer most functions to a general competence query search, or to a different specific research area. The Book Nodes can easily be transferred by gathering a dataset for other textbooks, and can be removed if you want to apply this on general searches. `GitNode` can be changed to search through other repositories, by simply specifying the path to another git repository instead of the MRST one. Otherwise you can just remove it as it's not a main part of the program. You'll also have to change some prompts, so that they are not too MRST specific. Most code logic should be general enough to use though. You will have to remove the web search tool `web_search_mrst` defined in the folder `src/mrst_competence_query/tools`, and either replace it by a similar search tool specified for your project, or just remove it altogether. The tool is used to generate keywords based on the query, so it is not a strictly necessary tool, but it helps in order to generate relevant enough keywords. It might also be smart to tweak some of the parameters defined in the code, such as `gamma` defined in `SearchAndEvaluateNode` or `kappa` in `SuggestionsNode` which is used to calculate the relevance score for each researcher. Another thing that might need changing is how to choose a threshold for cosine similarity in the first retrieval.

The most important thing to consider when creating different competence query applications, is that the application only gets as good as the dataset. If you want specific answers, and you want to show subtle nuances between documents, people or fields, the most important thing is having a well structured, well documented and large dataset.