Problem Statement:

Background

With the growing demand of multiple factors, this is an era of continuous research and developments. Scientists and researchers are working hard to provide advanced technologies and methods to serve the well-being of a greater society. However, they struggle a lot when it comes to gathering relevant background information on any research topic. They need to spend a substantial amount of time to go through previous research papers for literature reviews and decide whether a particular paper is of their interest or not.

Objective

To solve the above problem, the goal of this project is to **build an efficient Knowledge Management System capable of querying and retrieval to help fellow researchers with the recommendation of research papers relevant to their topic of interest**. The system which we will build accepts research papers (pdfs) and the user query as inputs and outputs those papers which are highly relevant. We will take into consideration the information retrieval and aspect identification techniques of Natural Language Processing (NLP) to identify key aspects of chosen research papers and extract its summarized information. Similarity scores will be measured afterwards between user query and extracted data to pick the most relevant research papers as per user's interest.

Data:

The dataset for testing our NLP model will be some (belonging to diversified domains) research papers from websites like Google Scholar or Research Gate. We will then apply random sampling to select 5-6 research articles out of the collection.

Solution:

To accomplish all below mentioned tasks we have applied adequate solution strategies.

Task 1:

Keywords extraction from each header along with text summarization of each section.

Solution Steps -

- Import the sample research pdf within coding environment.
- Extract 'Title' of input pdf using '**PyPDF2**' library and check if it is showing an accurate output.
    - Split the text of the pdf by newline delimiter and assume the Title can be within first two lines mostly.

- Also if the first two lines come as empty string, then consider the file name of pdf as Title.
- For those pdfs showing title as '**None**' create a customized function for finding out the font size and font name details of every single line.
- Chalk out the text having maximum font size and consider it to be the Title for most of the time.
- Pre-process the text by word tokenization, lowercase conversion and remove less important words like stop words.
- The title is now processed and ready to use further as a variable '*clean_title*'.

➢ Now find the section headers of given pdf.

- Use the same function created for font size and font name attributes after finding the '*abstract_index*' previously to determine the details of header '**Abstract**'.
- Using that same information extract all other headers names given within pdf (considering most of the research papers have same font details for its section headers like 'Abstract', 'Introduction' etc.) and store within a list '*filtered_items*'.
- Remove '**References**' section from list of headers.
- Also, remove some unwanted parts from headers like Author names and Acknowledgements etc. Create final filtered list of headers '*new_items'*.
- Once all section names are extracted, format it in the exact way it is given within the original pdf. This will help to extract the text of each individual section.
- The cleaned header names are now stored within '*new_output*'.

➢ Parse the entire text of pdf.

- Extract the text of a pdf using **PyPDF2** and store it with *'extracted_text'*.
- Remove the '**References**' section from the extracted text for this phase of experiment.
- Pre-process the text using regex functions such as remove non-ascii characters, URLs etc and store it within '*cleaned_text*'.

➢ Now once the section headers and entire pdf text are properly extracted, find the text from each individual section using the indices of section headers.
(For ex. Using the consecutive index of '**Abstract**' and '**1 Introduction**' we can extract the text of **Abstract**.

- Finally, store the text from individual sections within a list as '*section_extraction*' and do some more pre-processing like stop-word removal and lower-case conversion.

- Summarize important sentences of each section using LSA (Latent Semantic Analysis).

  To achieve above, we may follow below steps:

  - Vectorize the sentences of each section using *TfidfVectorizer* function.
  - Apply LSA (Latent Semantic Analysis) on vectorized sentences (use *TruncatedSVD* function of *sklearn* library)
  - Once the *lsa_matrix* is defined which holds the components of singular vector decomposition (SVD) then calculate the lsa_scores of each sentence.
  - The sentences are grouped by their associated sections, and each sentence is paired with its LSA score. This allows sentences to be easily ranked within their respective sections.
  - For each section, the sentences are tokenized using the '*tokenize_sentences*' function. The sentences for each section are stored in the '*section_sentences*' dictionary.
  - For each section, sentences are sorted based on their LSA scores in descending order. The top-ranked sentences (specified by '*num_top_sentences'*) are then selected and printed for each section.
  - Append the top_sentences of each section within a list *'top_ranked_sentences'*.
  - Then make that *top_ranked_sentences* a dictionary which hold the key-value pair respectively as each section headers and its top-ranked sentences.

- Use an LLM model to compare the quality of summaries generated using LSA model.
  - Import the pre-trained **BART** model and tokenizer using Hugging Face Transformers library which is a state-of-the-art NLP model container.
  - Also load the English word embedding model from '**spaCy**'.
  - Define a short function '*select_top_sentences*' for determining the top sentences of each section.
  - The '*summarize_sentences'* function encodes the text, generates summary, and then returns the summary after decoding.
  - Store the section summaries as per respective section names within a dictionary.

Task 2:

Filter based on similarity scores of text summaries with user query.

Solution Steps -

- Process the extracted summaries, query and title text and convert the final section summaries (top 5 sentences) into **TF-IDF** vectors along with query and extracted title.

- Calculate the **Cosine Similarity** between the query and other vectors (**title/section summaries**) and store those as key-value pair of a dictionary named '*similarity_scores'*.
- Take the average of each section to measure overall similarity score of any individual pdf.
- Display the respective section similarity score and their overall average as a form of output table using '*PrettyTable*' library.
- Also, follow all above steps for measuring the similarity scores of BART generated summaries with user query.
- Additionally, use **GloVe** embedding for measurement of similarity scores both in case of LSA and LLM generated summaries.

Evaluation Metrics for checking the quality of generated summaries.

- Calculate the **BLEU**, **ROUGE** score and also the **METEOR** score to check the quality of LSA and LLM generated summaries with a reference summary of individual sections created by ChatGPT.

Task 3:

Display the output in CSV format.

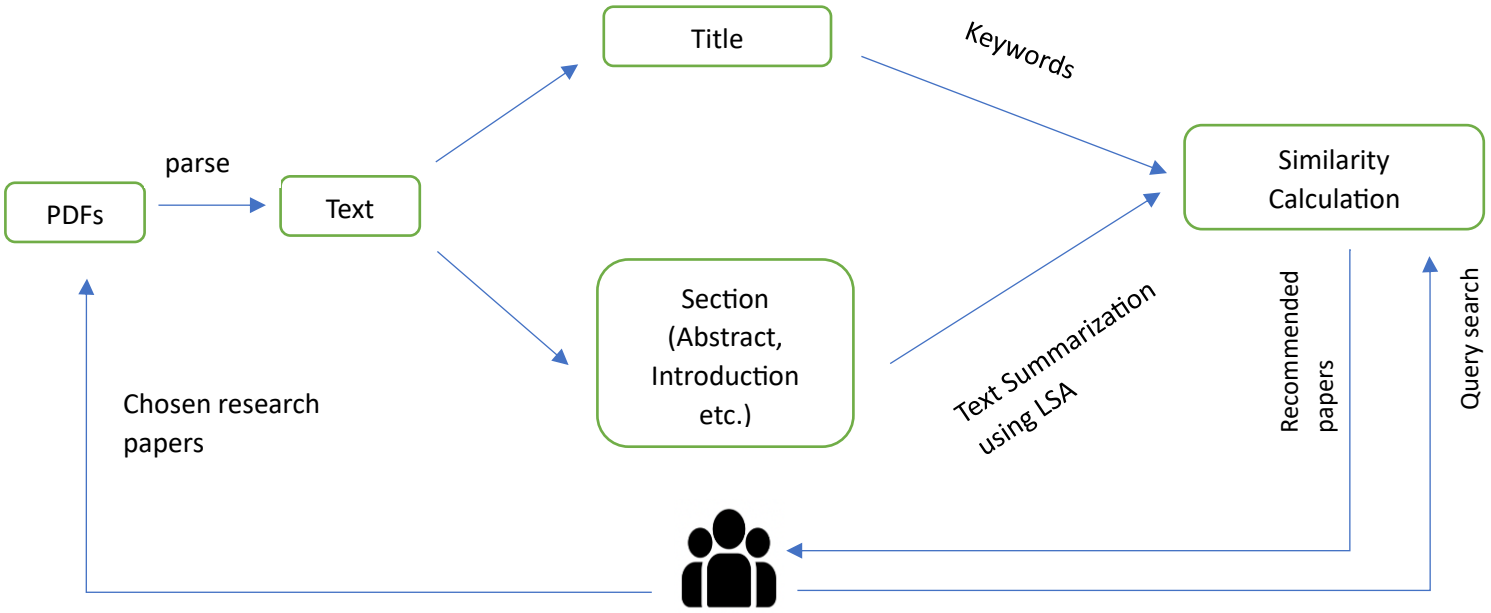
Solution Steps –

- Create a table for similarity scores of LSA Top-Ranked sentences and populate the similarity scores for each section individually.
- Add another row with the average similarity scores for LSA method.
- Create another table for populating the similarity scores of LLM method (**BART**) along with their average similarity.

Observations from output table and Evaluation metrics:

- The individual similarity scores for each section seem to be quite improved in the case of GloVe embedding of summaries than TF-IDF due to its capability of capturing semantic relationships of words.
- Our base LSA model is performing better than pre-trained BART model in terms of finding the similarity between user query and some of the sections of given pdf. However, the average similarity score is little higher for BART model, but it is quite closer to average score of LSA model.
- The evaluation metrics show close competition between LSA and LLM model.

Workflow Diagram:



Enhancements

Our future work will be based on making our solution work for multiple pdf files accurately. We will also try to improve our base model's performance through applying more efficient NLP techniques to capture the semantic relationship of words.

References:

- https://stackoverflow.com
- GitHub
- ChatGPT
- BARD