MULTIMEDIA UNIVERSITY

Final Year Project

Time Series Based Summarization

Mak Yen Wei

Bachelor of Computer Science

(Data Science)

Jan 2023

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

Google occupies over ninety percent of people's working hours when it comes to engineering. A large portion of a developer's responsibility is devoted to the removal of errors. Even though it is essential, it might be a tedious waste of time to look through a number of online forums and stack exchanges in order to solve an easy linear equation in Python(?, ?).It is not unusual for engineers to explore Stack Overflow for a number of hours in an effort to find an answer that is definite.

It is infamously difficult to address bugs in software due to the vast number of interdependent libraries and packages that are utilised by developers working in a wide variety of programming languages, frameworks, and platforms. Sometimes the problem is caused by a fresh new bug that was included in the most current version of the application. There is also the possibility that a mistake was made in the coding. In rare instances, the problem may also lie inside the local environment of the developer. There might be an infinite number of contributing elements here.

By utilising text processing techniques, ranking approaches, and summarization models, the work of this thesis is to locate the best solution to the problem faced by engineers. With premilinary research, stitching together context from online forums such as stackoverflow with code sharing platforms such as github in order to compile the best possible solution to the problem at hand. This was particularly highlighted in

(?, ?).

However, one of the major issue with proposing such framework is the data gathering process. The data is not readily available and hence data scraping is required. To find the best solution, it's benifical to have as much dataset as possible. Throughout the course of this investigation, a framework to address this issue by using cleverly timed cronjobs to scrape data from online forums as well as a FAQ generation process will be presented.

1.1 Problem Statement

FAQ Generation has been a long history, manually summarizing and cherry picking the best anwers to a question was the framework used for decades. The drawback of this method is that it's time consuming and not scalable(). With the advent of deep learning, the problem of FAQ generation has been addressed by using summarization models. **However the research done in the software engineering area is very limited**, most researches have made effort in the banks, insurance FAQ generation ().

Furthermore, **FAQ Generation on technical forums is a challenging task** as the corpus used will differ from general datasets such as the medical field. The terminilogies used differs drastically and hence special care must be taken to ensure the faq generated is accurate. (?, ?) (?, ?).

In addition, **keywords used by developers are not always accurate enough to find the best solution**. While google searching does a great job at classifying keywords to finding the matching solution, the same cannot be said for technical forums. Developers are not always able to find the best solution to their problem. (?, ?) because the amount of factors that contributes to a problem can be overwhelming.

1.2 Hypothesis

With the advent research gone into summarization, topic modelling, sentiment analysis models. A framework can be developed to address the problem of FAQ generation on technical forums. The framework will be able to generate a FAQ based on the keywords asked by the developer and also be able to rank the best solution to the problem.

1.3 Objective

Through the utilisation of text processing techniques, ranking approaches, and summarization models, providing assistance to developers in their search for answers is worth investigating. An automation towards faq generation in the software engineering domain is the primary aim of this thesis

The project objectives are as follows:

- **To gather** dataset from online forums in the software engineering domain.
- **To propose** a scalable, modular framework for FAQ generation on online forums in the software engineering domain.
- **To evaluate** multiple architectures for FAQ generation.

1.4 Scope

Platform

The stackoverflow forums may be scraped, ranked in terms of their usefulness to the developer's use case, and the top-ranked posts' comments, answers, and responses' comments can be summarised using the framework. As an added bonus, a web app will be built so programmers may use the framework to find the best solution to their problem.

Technologies

- **Web Application**: Python, FastAPI, AWS, ReactJS
- **Scraping**: Selenium
- **Text Processing**: NLTK, Spacy
- **Summarizing**: Tensorflow, Keras

Language

In the context of this thesis, the language used is English. The reason for this is because the majority of the online forums are in English.

CHAPTER 2

LITERATURE REVIEW

The following literature review section highlights the existing work that is related to our research. The work reviewed mostly consists of FAQ Processing and NLP techniques.

One of the caveats of the existing work is that, most of FAQ processing related work are based on the use on non-engineering fields such as the medical field. Likewise, since our research primarily focuses on the engineering field, we hope to migrate any intuition that we can get from the existing work to our research. Therefore a certain degree of reading has been done in terms of handling technical languages and terminologies.

2.1 FAQ Processing

FAQ retrieval is a crucial task when it comes to question answer pairs rankings (?, ?). It is a process of retrieving the most relevant questions from a large collection of questions based on a user query. Achieving excellent performance in this task will solve many problems that we've discussed in the previous chapter 1.1. However, most of the traditional methods for FAQ generation are based on the use of extensive manual classification and engineering (?, ?), this method takes tremendous amount of time and effort to be done. This was particularly highlighted by (?, ?) (?, ?) (?, ?) (?, ?). In addition, the performance of the framework is also affected by the quality of the manual classification and engineering.

To address the issues brought-up, there are attempts and researches that have done to improve the predecessor. One of the approaches on solving the problem, is to automate the process of FAQ generation using NLP techniques. (?, ?) has made progress on using deep learning methods such as combining both Deep Matching Networks and Multihop Attention Networks in performing FAQ retrieval. Deep Matching Network also known as (DMN) is a deep learning model that generates a matching scores based of 2 matrices inputs. The matrices are made of the dot product of embeddings of every word of question with every word of answers.While Multihop Attention Networks on the other hand is proven to be effective for reasonings tasks like question answering, which is our focus in this paper (?, ?). The network uses multiple "hops" of attention to gather information from a given input and make a prediction. It starts by encoding the input and then uses a decoder network that iteratively attends to different parts of the input, in order to generate an output.

(?, ?) (?, ?) takes an approach where a whole architecture is designed to achieve Auto-Faq-Gen which includes scraping, question construction, a ranking algorithm, and the question generation. The architecture is shown in Figure 2.1.

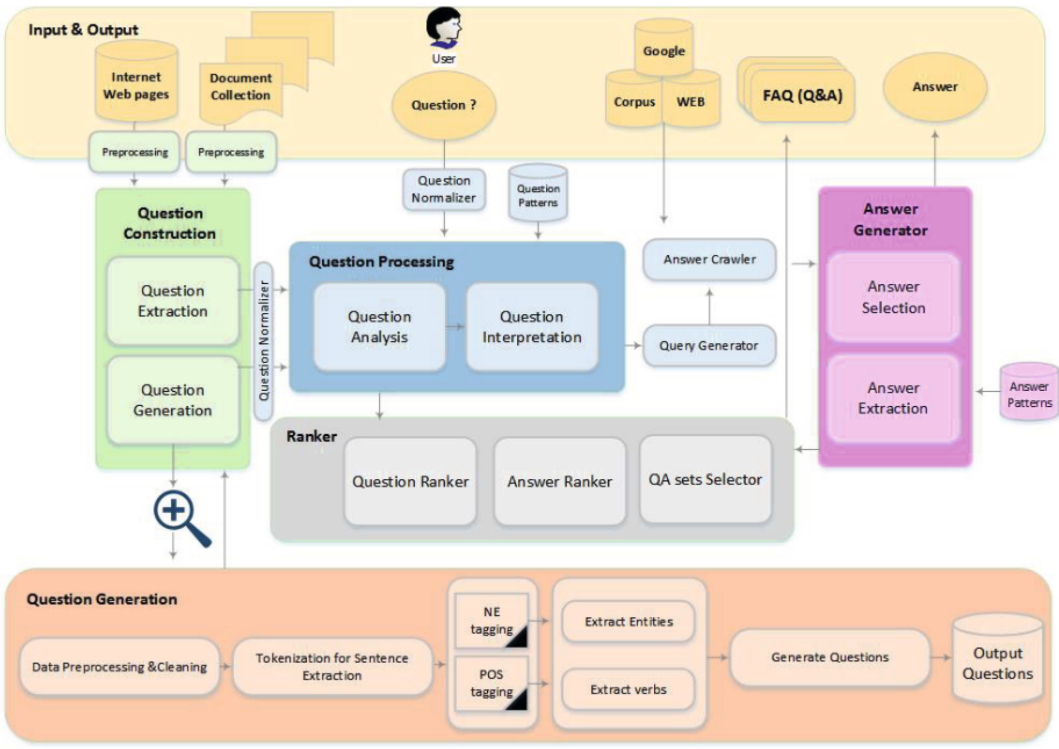


Figure 2.1: Auto FAQ Gen Architecture by ? (?)

The architecture presented was shown to be able to make Question and Answer Generation, Question Processing and a Ranker to rank answers.

2.1.1 Open Source Forums

Open source forums are great sources of information for users to find answers to their questions. However, over time, most forums are bloated with a large number of posts, it's always easier if some form of ranking can be performed on these forums to improve user experience.

(?, ?) (?, ?) has made progress in selecting, weighing, clustering and ranking contextual keywords in order to achieve questions abstraction and ultimately enhance the process of finding questions and related answers in open source forums. The author then proposed the solution to be in a format of semi-automatic FAQ generation.

2.1.2 Neural Networks

Neural Networks are a collection of algorithms modelled after the human brain. It is a collection of algorithms used to identify patterns. It is able to learn independently by evaluating unstructured material with the capabilities of self-learning and adjusting weights to reach the optimal outcome.

Convolutional neural networks (CNN) and RNN has paved it's way towards text processing tasks for the past decade. In (?, ?), CNN was proposed as a method to handle questions generation based on a given passage, it was done in a retrieval based format while RNN was propose to perform questions generation with a generation-based method. Both neural network were able to pull off question pattern prediction, question topic selection, question ranking and lastly question generation. The proposed model is shown to be able to outperform the traditional methods such as MAP or MRR.

2.1.3 Context Matching

Another issue arises, where most of the time, the questions asked by the user might not be easily classified to fit with the existing questions in the FAQ database. This is because the questions asked by the user might be in a different form or context (?, ?). For example, the question asked by the user might be in a different language, or the question asked by the user might be in a different form.

Grammatically, misspellings are another set of issue that we can forsee when it comes to building automated questions answerings, highlighted by (?, ?).

Context Matching is important when it comes to ranking the answers to the user query. The traditional way to score similarity is naive, where the score is calculated by levenshtein distance. (?, ?) However, this method is not effective when it comes to ranking the answers. This is because the levenshtein distance is not able to capture the semantic meaning of the words. Another concept comes to mind which is BOW, the bag of words model, (?, ?) highlighted to us where, BOWs similarity matching algorithm can be used on processed text (stop words removal, stemming, etc) to calculate the similarity between the query and the answer, however similarly to levenshtein distance, BOWs is not able to capture the semantic meaning of the words, which will bring false conceptual similarity between the query and the answer.

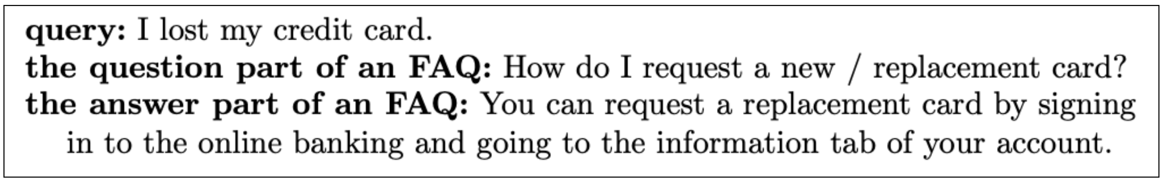
Word knowledge or Word Embedding is one of the solution to the traditional similarity matching algorithm. (?, ?) proposed a method that uses word knowledge to improve the similarity matching algorithm. The model stitch together a knowledge base to a word, which then constructs a knowledge table which consists of the raw words, Hypernyms, Synonyms and Associative concept of words. This solution breaks the traditional similarity matching algorithm, where the similarity is calculated by the semantic meaning of the words, instead of the raw words. (?, ?) has made progress on proposing a method called "WEKOS (Word Embedding, Kmeans and COSine based method)". WEKOS (Word Embedding, Kmeans and COSine) is a new method for question retrieval. The word embeddings of a question are weighted and averaged to get an overall representation of the question. The continuous word representations are learned in advance using the continuous bag-of-words (CBOW) model. (?, ?) The cosine similarity is used to calculate the similarity between the average of the word vectors corresponding to the question and that of each existing question.

2.1.4 Rankings

Deep Matching Network also known as (DMN) is a deep learning model that generates a matching scores based of 2 matrices inputs. The matrices are made of the dot product of embeddings of every word of question with every word of answers.

Multi-hop Attention Network on the other hand is proven to be effective for reasonings tasks like question answering, which is our focus in this paper (?, ?). The network uses multiple "hops" of attention to gather information from a given input and make a prediction. It starts by encoding the input and then uses a decoder network that iteratively attends to different parts of the input, in order to generate an output.

In order to solve the issue of context misconception between the query and the faq. We can highlight the issue below.



query: I lost my credit card.
the question part of an FAQ: How do I request a new / replacement card?
the answer part of an FAQ: You can request a replacement card by signing in to the online banking and going to the information tab of your account.

Figure 2.2: Misconception

In the above example 2.2, the query only matched with the correct output with 2 characters "I" and "Card", that way, the resulting score will be very low, even though the FAQ is relevant to the query. To counter this issue, the authors of (?, ?) proposed a method that predicts if the query corresponds to a FAQ using document classifier. The paper begins to highlight that the proposed document classifier uses binary classifiers to classifier each faq to the query. The sentences are broken into unigrams, bigrams to learn the dependency relation between the query and the faq.

2.1.5 Semantic Networks

Automatically Generated Semantic Networks are networks that are generated by machine learning algorithms based on text or other forms of data. These networks are used to represent the relationships and connections between different concepts, entities, or topics in a structured manner, in order to facilitate various NLP or information retrieval tasks, such as classification, summarization, or question answering. They can be thought of as knowledge graphs constructed automatically from text.

Traditionally, QA (Question Answering) are managed using syntatic, semantic methods. However these methods are not pratical to perform well on low resource languages like Swahili. Semantic networks can be used to langauges like Swahili, where the language generally follows a subject-verb-object structure. (?, ?).

2.1.6 Topic Modeling

Topic Modeling is another great way to find Question and Answers Pairings. Topic modeling is a type of statistical modeling for discovering the abstract "topics" that occur in a collection of documents. It is a frequently used text-mining tool for discovery of hidden semantic structures in a text body. Latent Dirichlet Allocation (LDA) is a popular topic modeling technique that is used to discover topics in a collection of documents. LDA is a generative probabilistic model that assumes each document is a mixture of topics, and each topic is a mixture of words. The model is trained on a corpus of documents, and the topics discovered by the model can be used to label new, unseen documents. (?, ?).

An architecture brought up by (?, ?) highlights the pipeline on leveraging topic modelling to perform prediction on the question and answer pairings. The pipeline proposed extracts the topics from the corpus, then the topics are used to label the questions and answers, which then extracts answers that closely relates to the topic mined by the model.

2.2 Stop Words

Stop words are words that are filtered out before or after processing of natural language data (text). Though "stop words" usually refers to the most common words in a language, there is no single universal list of stop words used by all natural language processing tools. The stopwords used in technical languages differs from the general stopwords list used by general application such as the NLTK library (?, ?) (?, ?).

The properties of stopwords can be defined as follows (?, ?):

- Words used for connecting important words.
- Words that are not important to the meaning of the sentence.
- Words that occurs frequently in the text.

While it might be wise to perform stopwords removal tasks on tasks such as Information Retrieval (IR), classification, it is not advisable to perform stopwords removal on tasks like summarization. (?, ?).

(?, ?) addresses the gap in identifying stopwords for texts in engineering fields which are not covered by the general stopwords list. The concluded stopwords list is statistically identified and evaluated by experts.

Phrases can be detected by using the algorithm of Mikolov at al (?, ?). The algorithm works by finding words that are frequently used together.

$$score(wi,wj) = ((count(wi*wj)-S)|N|/count(wi)*count(wj))$$

Term frequency-inverse-document-frequency (TFIDF) is a numerical statistic that is intended to reflect how important a word is to a document in a collection or corpus. Many information retrieval systems use TFIDF as a central tool.

TF (Term Frequency) is the number of times a word appears in a document, divided by the total number of words in the document.

$$TF = \frac{Number of times term t appears in a document}{Total number of terms in the document}$$

IDF (Inverse Document Frequency) is the log of the number of the documents in the corpus divided by the number of documents that contain the word w.

$$IDF = log_e(\frac{Total number of documents}{Number of documents with term t in it})$$

Term frequency-inverse-document-frequency (TFIDF) is then the product of TF and IDF.

$$TFIDF = TF * IDF$$

This method particularly favours words that appear many times in a document. Both of this methods are used to score and rank words in a document, and potentially form a stopwords lists. An interesting fact is that with using the two methods mentioned above, the stopwords list can be generated for a specific domain.

2.3 Sentiment Analysis

Sentiment analysis is the process of determining whether a piece of writing is positive, negative, or neutral. It's also known as opinion mining, deriving the opinion or attitude of a speaker. A common use case of sentiment analysis is to discover how people feel about a particular topic. For example, sentiment analysis can be used to discover how people feel about a new movie, or to learn about the general sentiment towards a new product.

While most sentiment analysis work are based of current social medias such as Twitter, Facebook, and Instagram, sentiment analysis can be applied to any text. For instance, (?, ?) uses sentiment analysis to further investigate and understand the customers needs and wants for goverment entities.

2.3.1 Grananularity

Sentiment analysis can be performed on different levels of granularity. The level of granularity refers to the level of detail at which the sentiment is expressed. There are three levels of granularity: sentence-level, document-level, and aspect-level. Taking into account the level of granularity is important because it affects the type of sentiment analysis that can be performed, and the type of results that can be obtained (?, ?).

2.3.2 Aspect Based

Aspect-Based Sentiment Analysis (ABSA) is a subfield of sentiment analysis that focuses on identifying and extracting opinions towards specific aspects or features of an entity such as a product, a service, or a person. It involves analyzing a piece of text to determine the sentiment expressed towards a particular aspect, rather than the overall sentiment of the text. This type of analysis is useful in understanding the specific opinions and attitudes of customers and users towards a particular product or service, and can be used in a variety of applications such as customer service, market research, and product development (?, ?) (?, ?) (?, ?) (?, ?).

The goal of aspect-based sentiment analysis is to identify the sentiment expressed towards a particular aspect of an entity. For example, consider the following sentence:

```
"The food at this restaurant is great, but the service is terrible."
```

Listing 2.1: Example of Aspect-based Sentiment Analysis

In this sentence, the overall sentiment might be positive, but in actuality the sentiment towards the food is positive, while the sentiment towards the service is negative. Therefore by using Aspect-based sentiment analysis can be used to identify the sentiment towards the food and the service separately. (?, ?) took this intuition and applied it to the government sector. They used the ABSA to identify the sentiment towards the government services and the government entities.

In our research, we'll be using the ABSA model in the hopes where it'll improve the

FAQ Ranker in terms of ranking their sentiment to potentially indicate the quality of the answer.

2.3.3 Traditional Methods

Lexicon and Corpus based approaches are unsupervised learning methods to understand the sentiment of a text. The lexicon based approach uses a predefined list of words coupled with the annotated polarity values of the word's sentiment. This approach has a major downside where there might be numerous words and expressions that are not included in the lexicons.

Corpus based approach however focuses more in predicting the word to be a prefixer or suffixer of the sentiment word. This approach is more accurate than the lexicon based approach, but it is still not perfect. (?, ?)

2.3.4 Machine Learning and Deep Learning

Multiple researches has proven that machine learning can make promising results in terms of ABSA specifically polarity classification. It was highlighted that attention-based or non attention based LSTM models can be used to achieve promising results (?, ?) (?, ?)

SVM had received the highest popularity in terms of polarity classification, it was often used with association rules and feature selection such as PCA to achieve promising results (?, ?) (?, ?) (?, ?)

Convolutional Neural Network were also used to predict polarity of a sentence, there were a handfull of attempts in using CNN coupled with Word2Vec that had been proven to be effective (?, ?) (?, ?) (?, ?) (?, ?)

To learn long term associations and dependencies between texts in a sentence or a document. Long Short Term Memory (LSTM) was a popular choice in learning information dependencies across the history of sentences in a document. Multiple atempts had been made to take advantage of LSTM models in buildling polarity classification models (?, ?) (?, ?) (?, ?)

2.4 Summarization

Text summarization is the process of automatically generating a shorter version of a text that preserves the most important information. There are two main types of text summarization: extractive and abstractive. Extractive summarization involves selecting and concatenating important sentences from the original text, while abstractive summarization involves generating new sentences that summarize the meaning of the original text (?, ?) (?, ?).

In contrast to general text summarization, summarizing software engineering related texts differs itself from the general text summarization. The main difference is that the software engineering related texts are often more technical and contain more technical terms, not to mention that at times, processing raw source codes can be a challenge (?, ?).

The evaluation metrics used generally is to calcualate the compression rate denoted by the ratio of the length of the summary to the length of the original text.(?, ?) The formula is as follows:

$$CompressionRate = \frac{LengthofSummary}{LengthofOriginalText}$$

2.4.1 Extractive Summarization

Extractive summarization involves selecting and concatenating important sentences from the original text. The most common approach to extractive summarization is to use a sentence ranking model to rank sentences in the original text, and then select the top ranked sentences to form the summary. The sentence ranking model can be trained using a variety of features, such as the number of named entities in the sentence, the number of words in the sentence which closely relates to TFIDF, the number of words in the sentence that are also in the original text, and the number of words in the sentence that are also in the summary (?, ?).

CN-Summ is one of the notable extractive summarization model that was introduced by (?, ?). CN-Summ is a neural network based model that uses a graph-like architecture to connect sentences that shared common significant nouns. The model consist preprocesses, a graph construction, and finally selecting the first x sentences as the summarization.

Similarly to using LDA in sentiment analysis (?, ?), LDA has also shed light in the field of extractive summarization. (?, ?) proposed the use of LDA to extractive summarization. The model uses LDA to extract the most important topics in the original text, and then uses the topics to rank sentences in the original text.

2.4.2 Abstractive Summarization

Abstractive summarization involves generating new sentences that summarize the meaning of the original text. The most common approach to abstractive summarization is to use a sequence to sequence neural network to generate the summary. A sequence to sequence neural network such as RNN, LSTM is used because they are well-suited for processing sequences of data, such as text. The recurrent connections in RNNs allow them to maintain a hidden state that can capture information from previous steps in the sequence, which is useful for keeping track of contextual information as the model reads the input text. This allows the model to generate a summary that is both concise and semantically relevant to the input text (?, ?).

AMR (Abstract Meaning Representation) was one of the RNN based methods that was introduced by (?, ?). AMR is a neural network based model produces a single graph to represent time series information in the text to form an abstractive summary.

ATSDL is another attempt where the paper introduce a use of combining advantages of extractive and abstractive summazy to form a hybrid model. The model uses a sequence to sequence neural network to generate the summary, and then uses a sentence ranking model to rank the sentences in the summary. The top ranked sentences are then selected to form the final summary (?, ?). A phrase collection model called MOSP was introduced to generate and learn phrase representations and their relationships in a document, the model solves the problem of unusual terms, which abstrative models faced most of the time.

2.5 Evaluation Metrics

Evaluating the success rate of extracting FAQ can be subjective, not all datasets coincide with the same domain. Therefore, a manual evaluation approach are mostly used to evaluate the success rate of a question finding process (?, ?). Manual classification of the questions pairs are carried out and the results are compared with the results of the question finding process.

However there are semi-automated frameworks that can potentially be used to evaluate the success rate of a question finding process. Alternatively, a user feedback framework can be constructed, where a user can provide feedback on the accuracy of the question finding process. The feedback can be used to evaluate the success rate of the question finding process (?, ?).

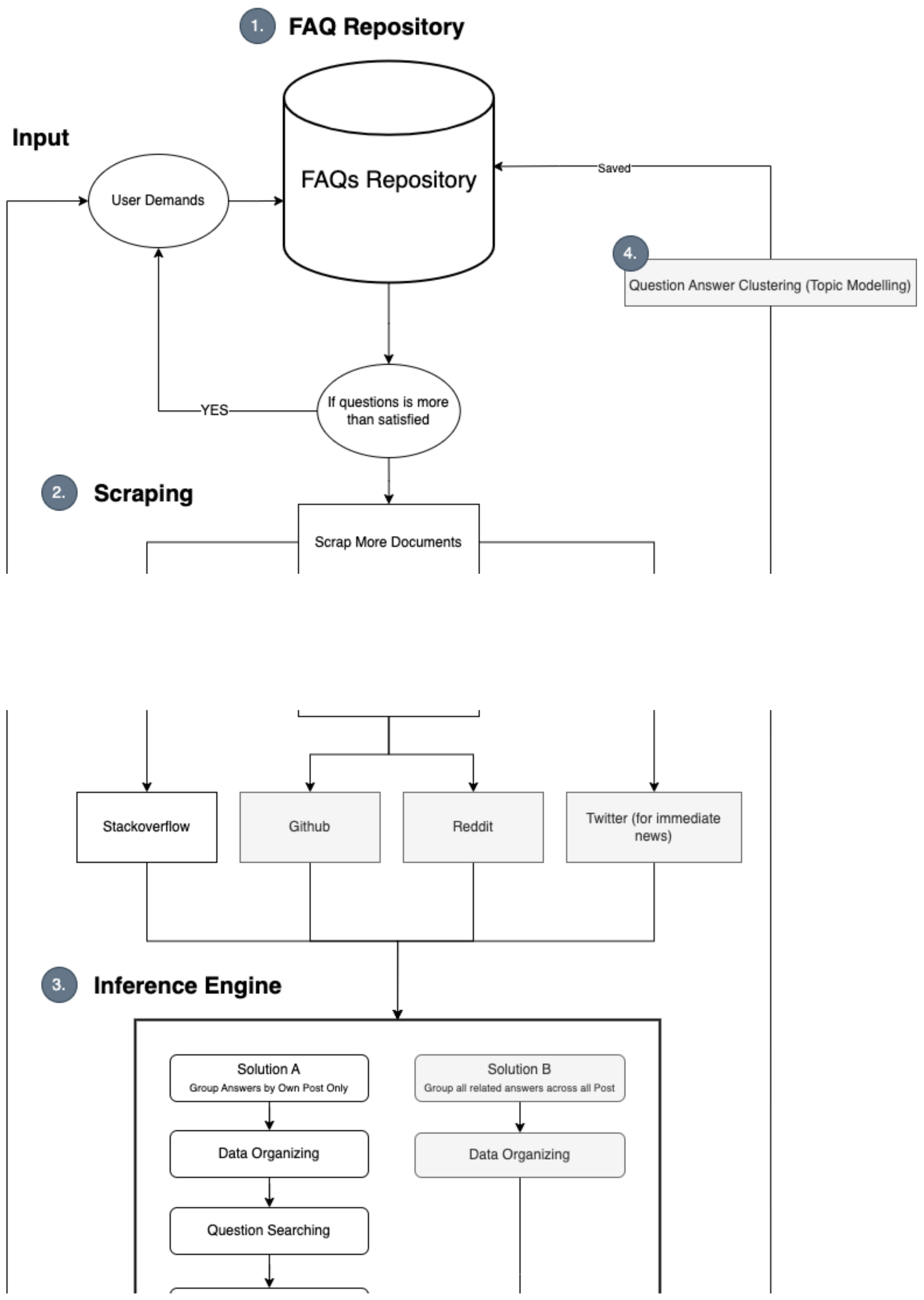
CHAPTER 3

PROPOSED FRAMEWORK

3.1 Introduction to the framework

The following framework, which comprises a series of operations to be done on the data. Figure 3.1 is an illustration of the entire framework.

The pipeline proposed includes all the essential components to comprehensively handle the neccessary procedures to solve the problem. The framework was designed to be modular, therefore each component is replaceable with a different implementations if necessary. A distributed design was used in the development of the framework so that it may be utilised on a broad range of computer systems. The framework's ability to readily include more procedures as they become necessary is made possible by the modular nature it possesses.


1. FAQ Repository
FAQs Repository
Input
User Demands
Saved
4. Question Answer Clustering (Topic Modelling)
If questions is more than satisfied
YES
2. Scraping
Scrap More Documents
Stackoverflow
Github
Reddit
Twitter (for immediate news)
3. Inference Engine
Solution A
Group Answers by Own Post Only
Solution B
Group all related answers across all Post
Data Organizing
Data Organizing
Question Searching

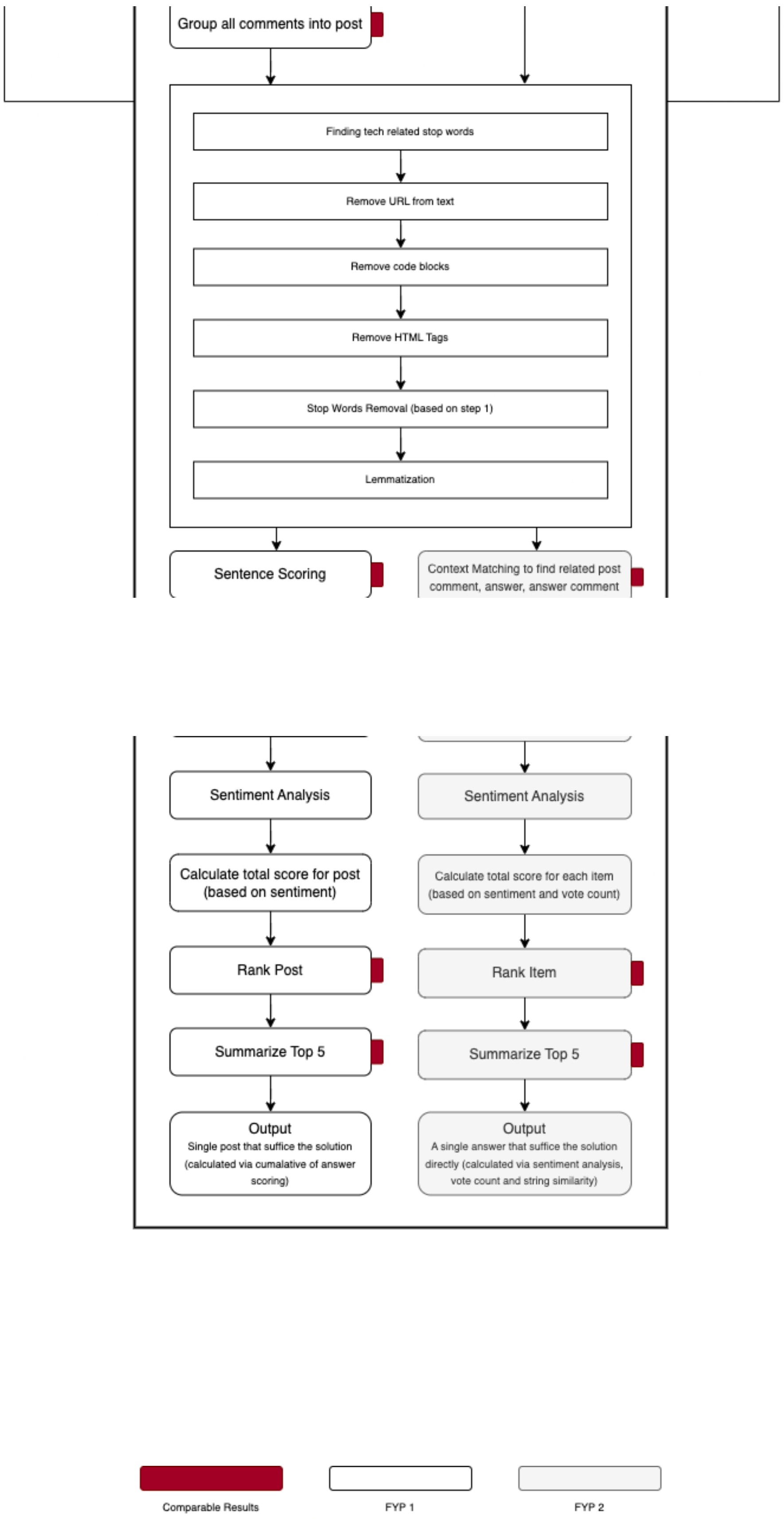

Group all comments into post
Finding tech related stop words
Remove URL from text
Remove code blocks
Remove HTML Tags
Stop Words Removal (based on step 1)
Lemmatization
Sentence Scoring
Context Matching to find related post comment, answer, answer comment
Sentiment Analysis
Sentiment Analysis
Calculate total score for post (based on sentiment)
Calculate total score for each item (based on sentiment and vote count)
Rank Post
Rank Item
Summarize Top 5
Summarize Top 5
Output
Single post that suffice the solution (calculated via cumalative of answer scoring)
Output
A single answer that suffice the solution directly (calculated via sentiment analysis, vote count and string similarity)
Comparable Results
FYP 1
FYP 2

Figure 3.1: Overall proposed framework

Note: The background color of each cell in the framework denotes to:

- Red - *Comparable Results*: Stages where evaluation is done to compare results between each stages.
- White - *FYP 1*: Stages where the main focus is on FYP 1.
- Grey - *FYP 2*: Stages where the main focus is on FYP 2.

3.2 Data Collection (Scraping) – 1.0

According to the suggested framework 3.1, four primary data sources are identified in this work : Stack Overflow, GitHub (Issues), Reddit, and Twitter. Each dataset exhibits different characteristic. The data on Stackoverflow is in the form of questions and answers, but the data on GitHub is in the form of code majority speaking. Reddit data is a forum, whereas Twitter data is more likely to have rapid, real-time updates.

In terms of FYP1, Stackoverflow will be our primary data source because it makes it much easier for us to demonstrate whether or not our method works. As a result, the dataset was scraped from stackoverflow.

The stackoverflow api is utilized to locate relevant topics on stackexchange depending on the developer's query. An api token is generated on the user dashboard.

The Stack Exchange API is a RESTful API that allows one to access data from Stack Exchange sites. It is a read-only API, which means that one can only retrieve data from the sites, not modify it. The API is available at api.stackexchange.com/docs. One can use the API to retrieve data from Stack Exchange sites in a variety of formats, including JSON, XML, and JSONP. We'll be quering the data in JSON format.

Example of a query to the Stack Exchange API:

```
{
  "tags": [
  "node.js",
  "reactjs",
  "react-hooks"
  ],
  "owner": {
  "account_id": 26363344,
  "reputation": 9,
  "user_id": 20019220,
  "user_type": "registered",
  "profile_image": "https://lh3.googleusercontent.com/...",
  "display_name": "George Prethesh",
  "link":
     "https://stackoverflow.com/users/20019220/george-prethesh"
  },
  "is_answered": false,
  "view_count": 27,
  "answer_count": 3,
  "score": 0,
  "last_activity_date": 1674042867,
  "creation_date": 1674039763,
  "question_id": 75158231,
  "content_license": "CC BY-SA 4.0",
  "link": "https://stackoverflow.com/questions/75158231/..",
  "title": "React useEffect OnSubmit Rendering Post api
     multiple times"
  },
```

Figure 3.2: Stack Exchange API response

Figure 3.2 displays the response from the Stack Exchange API call with a number of useful pieces of data, including the tags, owner, title, link, and many more. In order to automatically scrap the necessary information from the relevant web sites. Selenium is used in this work.

Selenium is a free (open source) automated testing suite for web applications across different browsers and platforms. It is used to automate web applications for testing purposes, but is certainly not limited to just that. Bots that perform web scraping, data mining, load testing, network traffic recording, and screen scraping are all common uses for the tool.

However, there is a problem since most websites nowadays uses anti-bot procedures to block automated programmes. Since the scraper may be blocked from accessing a Cloudflare-protected website, this complicates web scraping.

Therefore, A combinition of selenium with the a python package named undetected chromedriver that is created and maintained by *ultrafunkamsterdam*, to properly exploit its potential. This bundle is a selenium wrapper that enables us to utilise selenium without being noticed by the website. The website will deny us access if it discovers that selenium is being used, so knowing this is crucial.

When all of that is complete and in place, scraping on all the posts will be done. The python code is designed to be error fail save, with the bulk of the code wrapped inside try catches blocks, so that if an error does occur, it will not stop the current process. This is critical since we'll be extracting more than 200-300 posts most of the time.

Below is the list of features scraped from stackoverflow. It contains 21 features in total. Post id, Post link, Post Title, Post Body, Post date, Post Votecounts, Comment id, Comment score, Comment username, Comment text, Comment Date Time, Answer id, Answer Text, Answer Body, Answer Date Time, Answer Votecounts, Answer Comment id, Answer Comment Text, Answer Comment Body, Answer Comment Date time, Answer Comment Votecounts,

Because Selenium has a capability called "Find Element," it is quite easy to zero in on all of the components that are of interest. The information is structured in a manner similar to that of a dictionary, with the post id acting as the key and a list of all objects of interest performing the function of the value. After that, the data with the modifications are saved as a csv file. It is important to take note that the CSV file has a separate row for each comment, response, and response comment.

In this work, the scraper scrape articles that include the word "React UseEffect" in order to acquire the necessary data. As a result of this, a.csv file that has 3336 rows and 22 columns is saved and ready for use.

Notably, the web scraper is programmed to run once per day in order to provide us with the most recent data possible. This is necessary since data is always shifting, the framework aims to compile the most up-to-date information possible. In addition to this, the scraper will be housed on a server so that it may expand as the needs of the business dictate.

3.3 Inferencing Engine – 2.0

Figure 3.1 shows the proposed inference engine which will handle the core parts of the system. From preprocessing, sentence scoring, and sentence selection. Every nitty-gritty wil be explained in detail in the following sections.

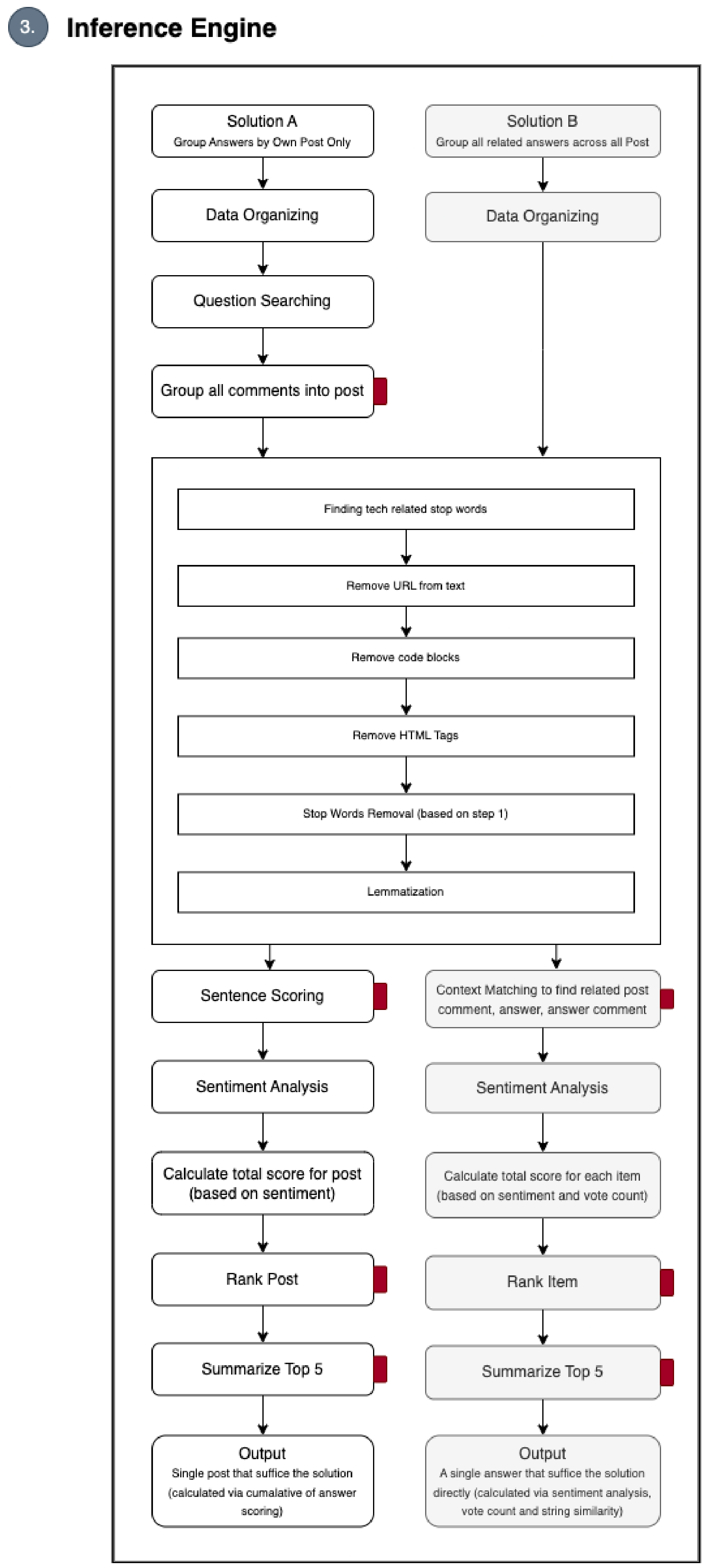

3.
Inference Engine
Solution A
Group Answers by Own Post Only
Solution B
Group all related answers across all Post
Data Organizing
Data Organizing
Question Searching
Group all comments into post
Finding tech related stop words
Remove URL from text
Remove code blocks
Remove HTML Tags
Stop Words Removal (based on step 1)
Lemmatization
Sentence Scoring
Context Matching to find related post comment, answer, answer comment
Sentiment Analysis
Sentiment Analysis
Calculate total score for post (based on sentiment)
Calculate total score for each item (based on sentiment and vote count)
Rank Post
Rank Item
Summarize Top 5
Summarize Top 5
Output
Single post that suffice the solution (calculated via cumalative of answer scoring)
Output
A single answer that suffice the solution directly (calculated via sentiment analysis, vote count and string similarity)

Figure 3.3: Proposed Inference Engine

Note: The background color of each cell in the framework denotes to:

- Red - *Comparable Results*: Stages where evaluation is done to compare results between each stages.
- White - *FYP 1*: Stages where the main focus is on FYP 1.
- Grey - *FYP 2*: Stages where the main focus is on FYP 2.

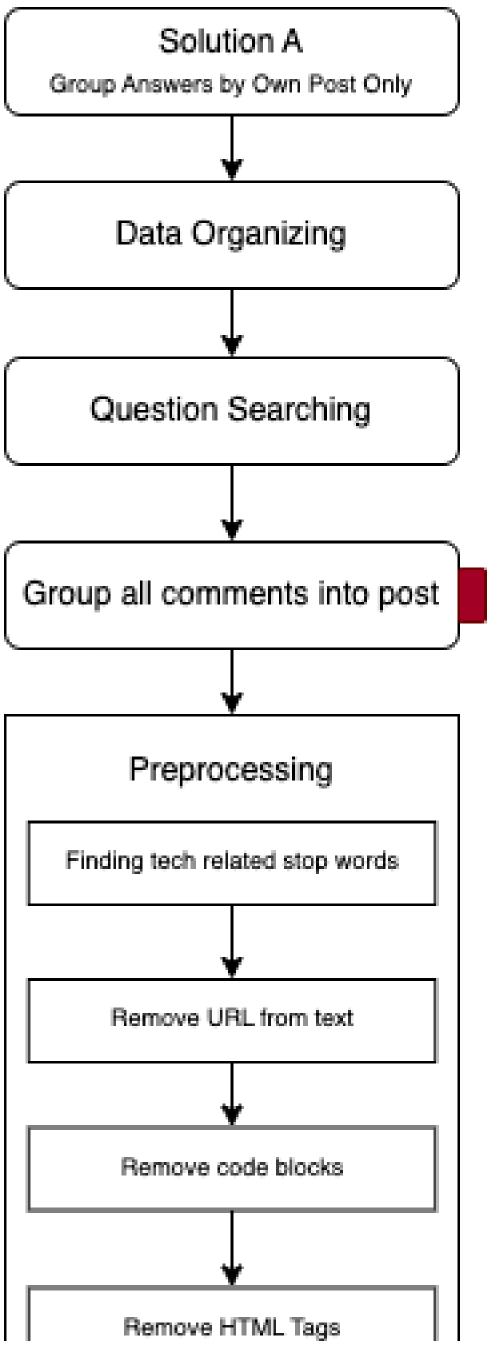
In the proposed inference engine system, 2 ideas had be possed to work on the problem, they are named as solution A and solution B. While the work on FYP 1 only focuses on solution A, the work on FYP 2 focuses on testing both solutions, comparing the results and finding the best approach.

3.3.1 Solution A

Posts titles are usually short and concise, and they are usually the first thing that people search for when they're looking for a solution to their problem. Therefore, it is intuitive to assume that the title of the post tells us what the post is about. With that being said, all comments and answers should be incapsulate in the post itself, and the post should be view as a whole when ranking is performed. This is the idea behind solution A. Essentially, solutions A focuses on ranking the posts incorporating the comments and answers into the post itself.

3.3.2 Solution A - Architecture

A close up view of the architecture of solution A can be seen in the following figure 3.4.


Solution A
Group Answers by Own Post Only
Data Organizing
Question Searching
Group all comments into post
Preprocessing
Finding tech related stop words
Remove URL from text
Remove code blocks
Remove HTML Tags

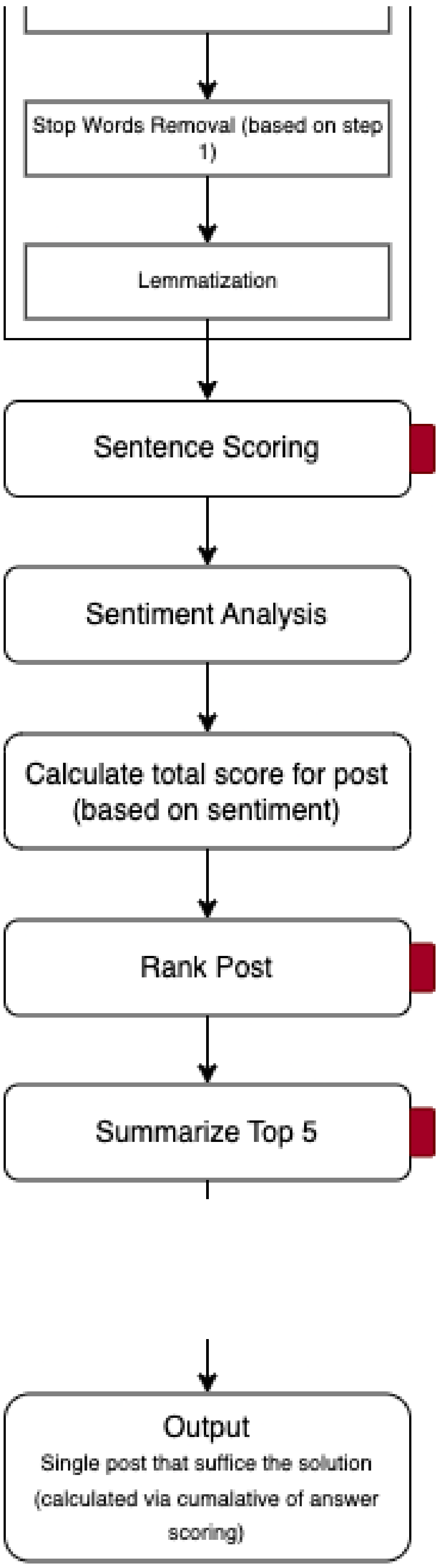
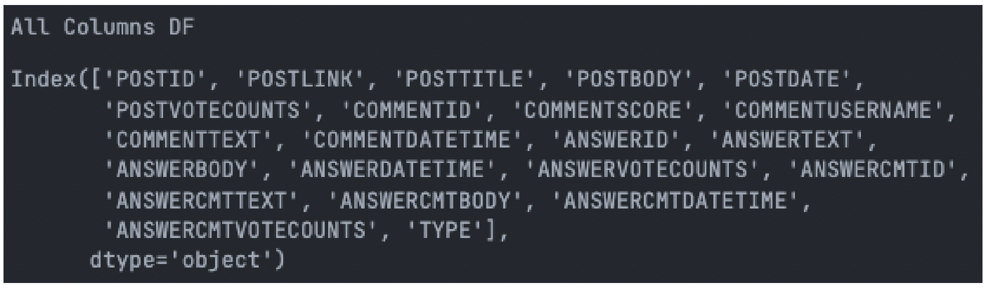

Stop Words Removal (based on step 1)
Lemmatization
Sentence Scoring
Sentiment Analysis
Calculate total score for post (based on sentiment)
Rank Post
Summarize Top 5
Output
Single post that suffice the solution (calculated via cumalative of answer scoring)

Figure 3.4: Solution A Architecture

3.3.2 (a) Data Organizing

First, The information will be arranged into the appropriate buckets. Posts, comments, answers and answers comments are the four primary types of information that'll be separate apart. The goal here is to facilitate our work with the data. The reason for this is that the data in the csv is not organized in a way that is conducive to our work. The comparison of the data before and after the organization is shown in the following figure.

Essentially the data is organized from this

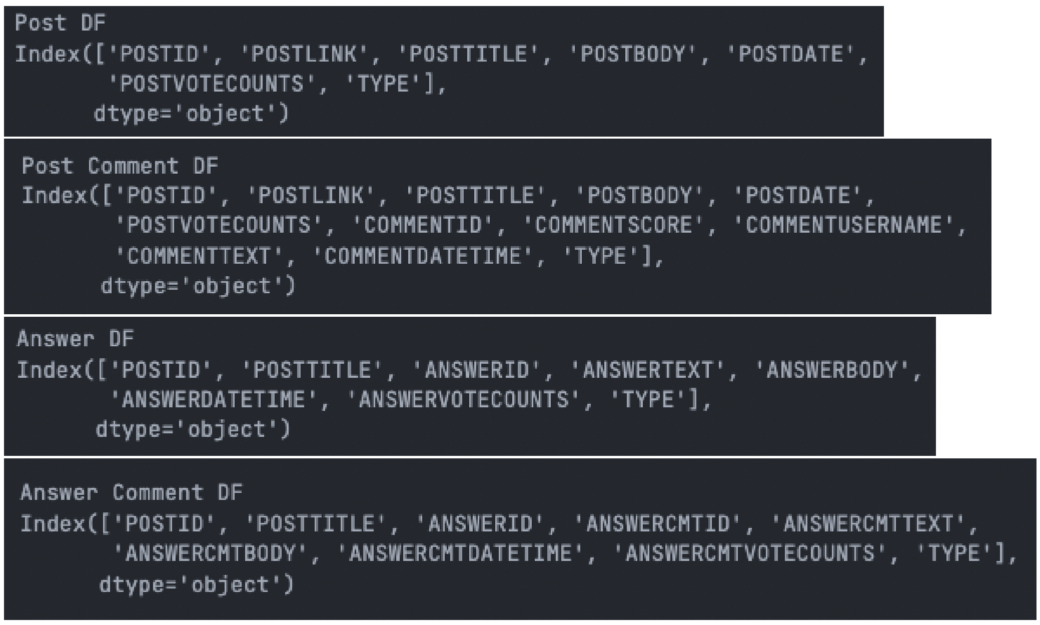


```
All Columns DF

Index(['POSTID', 'POSTLINK', 'POSTTITLE', 'POSTBODY', 'POSTDATE',
       'POSTVOTECOUNTS', 'COMMENTID', 'COMMENTSCORE', 'COMMENTUSERNAME',
       'COMMENTTEXT', 'COMMENTDATETIME', 'ANSWERID', 'ANSWERTEXT',
       'ANSWERBODY', 'ANSWERDATETIME', 'ANSWERVOTECOUNTS', 'ANSWERCMTID',
       'ANSWERCMTTEXT', 'ANSWERCMTBODY', 'ANSWERCMTDATETIME',
       'ANSWERCMTVOTECOUNTS', 'TYPE'],
      dtype='object')
```

Figure 3.5: Unorganized Data

To this:



```
Post DF
Index(['POSTID', 'POSTLINK', 'POSTTITLE', 'POSTBODY', 'POSTDATE',
       'POSTVOTECOUNTS', 'TYPE'],
      dtype='object')
```

```
Post Comment DF
Index(['POSTID', 'POSTLINK', 'POSTTITLE', 'POSTBODY', 'POSTDATE',
       'POSTVOTECOUNTS', 'COMMENTID', 'COMMENTSCORE', 'COMMENTUSERNAME',
       'COMMENTTEXT', 'COMMENTDATETIME', 'TYPE'],
      dtype='object')
```

```
Answer DF
Index(['POSTID', 'POSTTITLE', 'ANSWERID', 'ANSWERTEXT', 'ANSWERBODY',
       'ANSWERDATETIME', 'ANSWERVOTECOUNTS', 'TYPE'],
      dtype='object')
```

```
Answer Comment DF
Index(['POSTID', 'POSTTITLE', 'ANSWERID', 'ANSWERCMTID', 'ANSWERCMTTEXT',
       'ANSWERCMTBODY', 'ANSWERCMTDATETIME', 'ANSWERCMTVOTECOUNTS', 'TYPE'],
      dtype='object')
```

Figure 3.6: Orginized Data

3.3.2 (b) Matching Posts based on String Similarity

The first step is to match the posts based on string similarity. This is done by using the python package fuzzywuzzy. Fuzzy Wuzzy is a Python library for doing approximate and partial string matching using Levenshtein distance.

There are two methods the work was used in fuzzy wuzzy, partial ratio is the first method, which is used to compare the similarity of two strings. The partial ratio method is a measure of the similarity between two strings, which is used in the fuzzy-wuzzy library. The partial ratio method calculates the ratio of the longest contiguous matching substrings between two strings.

Partial Ratio Flow:

- Split the longer string into all possible substrings of equal length to the shorter string.

- Calculate the ratio of similarity between each substring and the shorter string.
- Return the highest ratio of similarity.

This method can be more effective than other similarity measures, such as the Levenshtein distance, in certain situations where it's important to match substrings rather than individual characters. The partial ratio method can handle differences in the length of the two strings better than the full ratio method, which compares the two strings in their entirety.

The second method is token sort ratio. The token sort ratio method is a measure of the similarity between two strings, which is used in the fuzzywuzzy library. This method splits each string into a list of tokens (i.e., individual words or substrings) and sorts these tokens alphabetically. Then, it calculates the ratio of similarity between the sorted lists of tokens.

Token Sort Ratio Flow:

- Split each string into a list of tokens, using a specified delimiter (e.g., spaces).
- Sort the list of tokens in each string alphabetically.
- Calculate the ratio of similarity between the sorted lists of tokens.

The token sort ratio method is more robust to differences in word order than other similarity measures, such as the Levenshtein distance. For example, the token sort ratio between the strings "apple pear" and "pear apple" would be 100 percent, as the sorted lists of tokens are the same in both strings. This method is particularly useful when comparing strings that represent names or other similar data, where differences in word order can result in significant differences in the similarity of the strings.

In short, partial_ratio checks for longest matching substrings and token_sort_ratio checks for matching tokens after sorting them alphabetically. However, while both methods are robust enough to handle most common cases, they are not contextual and do not take into account the meaning of the strings.

Another string similarity method is then proposed, namely spaCy, which is a free, open-source library for advanced Natural Language Processing (NLP) in Python. It is designed specifically for production use and helps to build applications that process and "understand" large volumes of text.

The notable differences between FuzzyWuzzy and spaCy are:

- FuzzyWuzzy is based on Levenshtein distance, which is a metric for measuring the difference between two sequences. The levenshtein distance between two words is the minimum number of single-character edits (i.e. insertions, deletions or substitutions) required to change one word into the other.
- spaCy is based on word embeddings, which is a type of word representation that allows words with similar meaning to have a similar representation. Word embeddings are a type of word representation that allows words with similar meaning to have a similar representation. It is a distributed representation for the text that is perhaps one of the key breakthroughs for the impressive performance of deep learning methods on challenging natural language processing problems.
- FuzzyWuzzy is faster than spaCy, but spaCy is more accurate.
- FuzzyWuzzy is more suitable for matching short strings, while spaCy is more suitable for matching long strings.

Results are filtered to remain only those that have a score of 50 and above, as the results with a score below 50 are not relevant to the developer requirements. To crown a winner of two of this method in the work, the efficacy of both methods will be examined in the upcoming *chapter 4*.

3.3.2 (c) Group all answers, comments, and answer comments by post_id

The next step is to group all answers, comments, and answer comments by post_id, so this way, all answers, comments, and answer comments can be concatinated into a single dictionary with the post_id as the key, and thus all answers, comments, and answer comments can be viewed as a single entity, and can be compiled to a single string. Viewing everything as a single string is crucial for us as it's way easier to do sentence scording, sentiment analysis and other NLP tasks on a single string than on a list of strings. For that to happen a class called **GroupedComments** is created, which is responsible for grouping all answers, comments, and answer comments by post_id.

```
class GroupedComments:
  def __init__(self, title, post, post_comments, answers,
      answer_comments):
       self.title = title
       self.post = post
       self.post_comments = post_comments
       self.answers = answers
       self.answer_comments = answer_comments
```

Figure 3.7: GroupedComments Class

A simple python script that iterates over the data and groups all answers, comments, and answer comments by post_id. The disctionary can be visualized as follows:

```
"post_id": {
  "title": "title",
  "post": "post",
  "post_comments": "post_comments",
  "answers": "answers",
  "answer_comments": "answer_comments"
}
```

Figure 3.8: GroupedComments Dictionary Example

Since now that the data has been formatted nicely it is very easy for us to track the context of the data and to do further analysis on the data.

3.3.2 (d) Preprocessing - Finding tech related stop words

3.3.2 (d) The next step is to find tech related stop words. This is done in order to remove the stop words from the data. Stop words are words that are not important to the data, and they are usually words that are used frequently in the data.

However, finding stop words is a very tedious task, in the previous yeaers, most of the time one would use manually curated stopwords lists to remove stop words from the data (?, ?),the stop words list can come from a variety of sources, such as a list of stop words from a specific domain, or a list of stop words from a specific language. In our case, we're dealing with a very specific domain, which is the engineering domain, normal stop words lists are not suitable for our case, as they are not specific to the engineering domain. Noises in the dataset will be resulted if a normal stop words lists is used to remove stop words from the data.

Stop words list can be found by using a wordcloud to find the most frequent words in the data, and then removing the most frequent words from the data. However, this method is not very accurate, as it does not take into account the context of the data, and it does not take into account the importance of the words in the data.

(?, ?) has made tremendous effort on curating the stopword list for the engineering domain, they've done it by analyzing the data found in patent documents which mostly describes domains related to engineering. The notable techniques they've used are minimally describe in the Figure 3.9 below. In short, the author of (?, ?) had propose a procedure which carefully picks stopwords from patented documents with using prep-rocessing techniques, a ranking framework based of terms statistics and finally with an evaluation carried out by experts based on a term-by-term basis. With that being said, the stopwords list curaed by them are choosed to be used in this work.

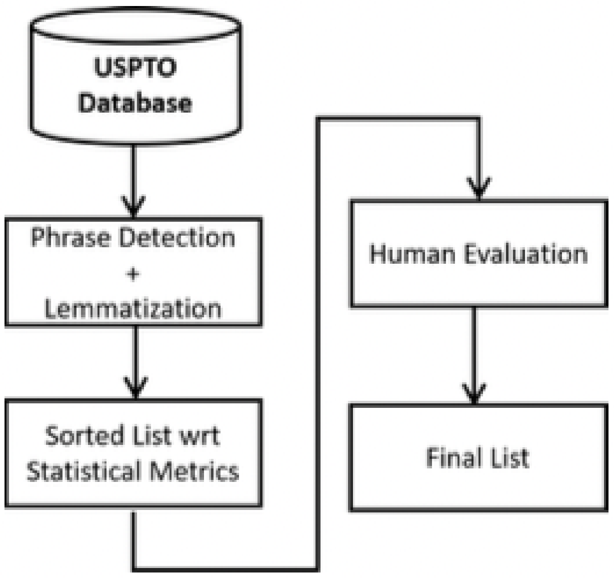

USPTO Database
Phrase Detection + Lemmatization
Sorted List wrt Statistical Metrics
Human Evaluation
Final List

Figure 3.9: Overall Procedure by ? (?)

3.3.2 (e) Remove Special Characters

The next step is to remove special characters from the data. The data scraped from StackOverflow consists all sorts of special characters, the data is not clean at all. Having special characters in the dataset might causes a lot of unwanted trouble, therefore special characters needed to be treated in our dataset. The table below **??** shows all the special characters that are identified in the dataset.

Case	VoteCount	Dates	All Columns
1	(	, License: CC BY-SA 4.0	segFault
2	)	(	
3	,	)	
5	'	,	

Table 3.1: Special Character

3.3.2 (f) Preprocessing - Extract URL from text

It is essential to incorporate URLs into the data, as these may be mined for information on the context of the data. The url can be easily extracted from the data as it is scraped since the data contains HTML tags. The BeautifulSoup package was utilised to access the URLs included inside the data. As a consequence of this, the url can be stored in a separate column of the dataset so that it can be referred to in the future.

In FYP 2, an effort will be made to put in place a framework that can autonomously determine the context relationships between retrieved URLs and extract more information from the URLs that have been obtained. Applying a score framework is one way to determine whether or not the URL in question is relevant to the data. If the scoring method wasn’t implemented, the process of the framework going to each of the urls would take a very significant amount of time.

3.3.2 (g) Preprocessing - Remove code blocks

The framework focuses on removing code blocks next, code blocks are not generally found in normal use cases and are not a huge concern on normal NLP tasks, however, by nature, this work is revolving around stackoverflow, a platform where developers post their questions and answers, and code blocks are a very common occurrence in stackoverflow. Therefore, removing code blocks is a very important step in the preprocessing process to reduce noises in our dataset. Code blocks are usually surrounded by “ “ “ or “ “ “ in the data, therefore it’s is effortless to identify code blocks in the pipeline and to remove them.

3.3.2 (h) Preprocessing - Remove HTML Tags

HTML Tags are another concerning factors in our work, where the scraping process might have caused some HTML tags to be included in the data. Therefore, special attention is needed to be given to remove the HTML tags. HTML tags are usually surrounded by < and > in the data.

3.3.2 (i) Preprocessing - Stop Words Removal

In the previous step 3.3.2 (d), we’ve found a list of stop words that are related to the technology domain. The removal is not done on the previous step because the text might contain noisy data. At this current stage, the data has been cleaned and this suggests that the removal of stop words can be done.

3.3.2 (j) Preprocessing - Lemmatization/Stemming

The process of truncating a word to it’s root or base unit is called stemming or lemmatization.

In the discipline of Natural Language Processing, text normalisation techniques like stemming and lemmatization are employed to get sentences, words, and documents ready for analysis. For instance, the terms "kick" and "kicked" are both forms of the verb "to kick," thus it’s possible that you’ll want your natural language processing application to recognise this. This is the idea of stripping down many uses of a term to its essential meaning.

Techniques like lemmatization or stemming can be achieved by using the famous Python NLTK package.

However, whether to utilise stemming or lemmatization is a contentious issue. Because stemming is a primitive heuristic procedure that cuts off the ends of words in the aim of reaching this objective properly most of the time, it often involves the removal of derivational affixes.

Lemmatization, on the other hand, is a more complex technique that takes into consideration morphological examination of the words. It does this by the use of a lexicon and morphological analysis of words, with the goal of removing only inflectional ends and returning the base or dictionary form of a word, known as the lemma.

The decision on using stemming or lemmatization will be later defined in chapter 4.

3.3.2 (k) Sentence Scoring

Consequently, all of our sentences should be clean and ready to be scored. The scoring process is done by using the TF-IDF algorithm. The TF-IDF algorithm is a statistical measure that evaluates how relevant a word is to a document in a collection of documents. The algorithm is composed of two terms: the first computes the normalized Term Frequency (TF), to further clarity, the number of times a word appears in a document, divided by the total number of words in that document; the second term is the Inverse Document Frequency (IDF), computed as the logarithm of the number of the documents in the corpus divided by the number of documents where the specific term appears.

The TF-IDF score is the product of these two terms. The TF-IDF score increases proportionally to the number of times a word appears in the document and is offset by the number of documents in the corpus that contain the word, which helps to adjust for the fact that some words appear more frequently in general.

The TF and IDF scores are defined as follows:

$$\text{TF} = \frac{\text{Number of times term t appears in a document}}{\text{Total number of terms in the document}} \tag{3.1}$$

$$\text{IDF} = \log \frac{\text{Total number of documents}}{\text{Number of documents with term t in it}} \tag{3.2}$$

At last, The TF-IDF scoring can be simplified into a single formula as shown below:

$$\text{TF-IDF} = \text{TF} \times \text{IDF} \tag{3.3}$$

Each sentences are then scored using the TF-IDF algorithm, the scored are then averaged and the average score is used to rank the associated post. This suggests that the post with the highest score will be the most relevant post to the question.

3.3.2 (l) Sentiment Analysis

Sentiment analysis is the process of determining whether a piece of writing is positive, negative, or neutral. It's also known as opinion mining, deriving the opinion or attitude of a speaker. A common use case of sentiment analysis is to discover how people feel about a particular topic.

Sentiment analysis is usually performed on textual data to help businesses monitor brand and product sentiment in customer feedback, and understand customer needs. It can also be used to gauge the sentiment of the general public to a particular event.

But how does sentiment analysis take place in our framework?

For context, each posts are scored based on the TF-IDF algorithm. The score is then used to rank the post. However the score itself is not telling enough to determine whether the post is useful or not. We've made an assumption that the post with a positive sentiment is more likely to be useful than the post with a negative sentiment. Hence, sentiment analysis is used to determine the sentiment of the post and gives a telling factor of how well "ranked" the post is. Ultimately, it is to add another layer of filtering on the ranking aspect of the framework.

3.3.2 (m) Sentiment Analysis – Model

The sentiment model used in this work is the famous twitter roberta base sentiment analysis model, which has been used over 2 million times this month. It is a RoBERTa base model trained on approximately 124M tweets from January 2018 to December 2021, and finetuned for sentiment analysis with the TweetEval benchmark. The model is able to predict and inference a probability score for each of the 3 classes: positive, negative, and neutral.

While twitter roberta based sentiment analysis is a normal sentiment analysis model, multiple research has shown in *chapter 2* that aspect-based sentiment analysis is a better approach to sentiment analysis as it is able to identify the sentiment of a specific aspect of a sentence. An aspect-based sentiment analysis approach will be examined in FYP2.

3.3.2 (n) Summarization

Finally as the framework approached to the final cell of the pipeline 3.4, in order to fulfill a better user experience, the engine will then summarize the top 5 sentiment ranked posts.

Summarization is the process of shortening a text document with software, in order to create a summary with the major points of the original document. As the name suggests, it is a technique to reduce the size of the given text while trying to retain the most important information. A summarization is crucial in our framework as it will help the user to understand the post better, faster and easier.

3.3.2 (o) Summarization – Model

The model we've used is the very well known bart large cnn model, which has been used over 1 million times this month. BART model pre-trained on English language, and fine-tuned on CNN Daily Mail. It was introduced in the paper BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation

BART is a transformer encoder-encoder (seq2seq) model with a bidirectional (BERT-like) encoder and an autoregressive (GPT-like) decoder. BART is pre-trained by (1) corrupting text with an arbitrary noising function, and (2) learning a model to reconstruct the original text. BART is particularly effective when fine-tuned for text generation (e.g. summarization, translation) but also works well for comprehension tasks (e.g. text classification, question answering). This particular checkpoint has been fine-tuned on CNN Daily Mail, a large collection of text-summary pairs.

Summarization will be done to each of the top 5 sentiment ranked posts. To clarify, the summarization summarizes all of the texts from the post, including the post, post comments, answer and lastly answer comments. In addition, the sentences are sorted based on time, this is to ensure that the summarization is in chronological order, where BART will be able to pickup any time series related information about the context of the on-going conversation. Namely, user mentioning each others in the comments section.

3.3.3 Solution B

The first solution focuses on ranking posts, whereby all comments, answers and answers comments are grouped together with the post and is been seen as a whole entity. Closed loop context, is the issue that we'll be addressing in this solution. We'll be explaining the architecture, the issue of solution A in detail, and the approach that we'll be taking to solve the issue.

3.3.3 (a) Solution B - Issue of Solution A

The issue of solution A is the situation – closed loop context, meaning that the ranking of the posts solely depends on the numbers, interactivity and the sentiment of the comments, answers and answers comments of the post.

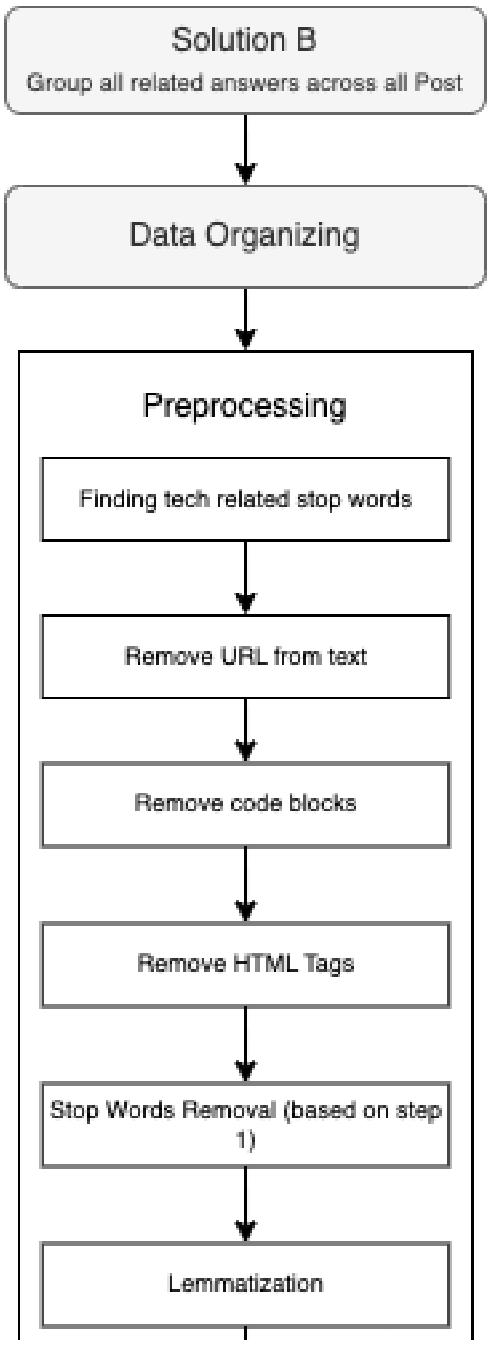
This is not a serious problem, however there is room for improvement. Jumping on another point of view, let's zoom out from the post context, let's view all of the comments, answers, and answer comments all together from every posts. The argument that we're trying to define here is, is there possibility that the answers from other posts are more relevant to the query than the answers from the post itself? The sole reason where the post that contains the better answers is not ranked is because the title of the post is not relevant to the query.

This is where the second solution comes in. The pipeline uses mostly the same approach as solution A, however we'll be grouping all related answers across all posts and do string similarity, and the rest of the architecture will be more or less similar to solution A.

Note, this is a solution made up with an assumption of the possibility of the answers from other posts are more relevant to the query than the answers from the post itself. The actual implementation will be covered in FYP2.

3.3.3 (b) Solution B - Architecture

The architecture of solution B is very similar to solution A, the only difference is that the framework will be grouping all related answers across all posts, The architecture is shown below 3.10


Solution B
Group all related answers across all Post
Data Organizing
Preprocessing
Finding tech related stop words
Remove URL from text
Remove code blocks
Remove HTML Tags
Stop Words Removal (based on step 1)
Lemmatization

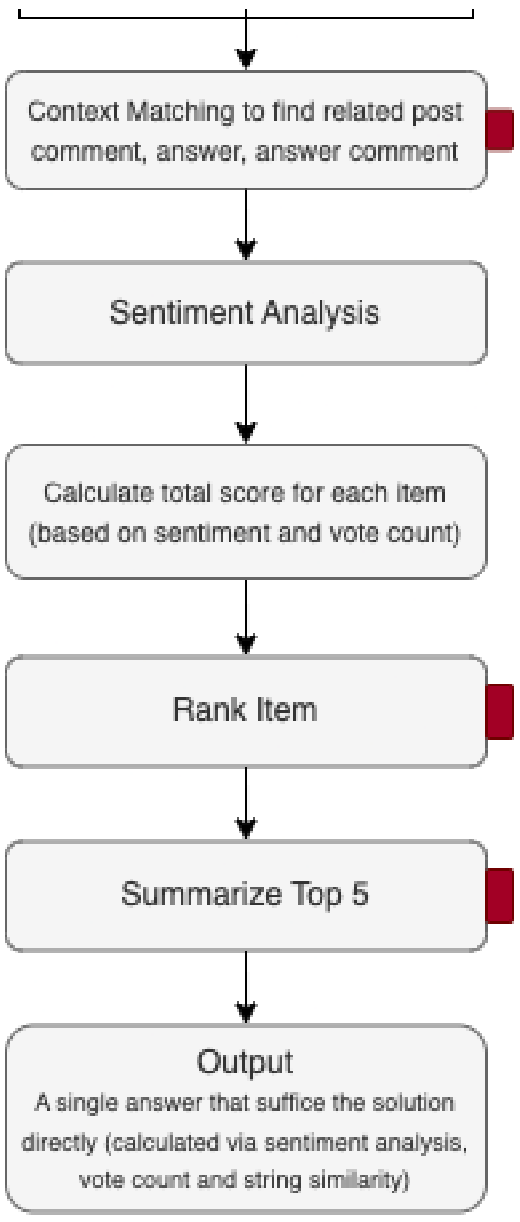

Context Matching to find related post comment, answer, answer comment
Sentiment Analysis
Calculate total score for each item (based on sentiment and vote count)
Rank Item
Summarize Top 5
Output
A single answer that suffice the solution directly (calculated via sentiment analysis, vote count and string similarity)

Figure 3.10: Solution B Architecture

3.3.3 (c) Solution B - Approach

The approach is very similar to solution A, the only differentiating factor is that the pipeline not ony focuses on an individual post when ranking but on a wider context, all of the posts, post comments, answer, answer comments. To summarise, all of the comments should be seen as equal no matter the parent post of the comments itself.

Data cleaning **??**, organizing 3.3.2 (a), pre-processing 3.3.2 (d) will be done in the same way as solution A. Just that the data will not be grouped by post 3.3.2 (c), but rather by all posts.

This way all comments, answers, answers comments take a part in the ranking process. This work believe that this is a better framework as it excludes the bias of the post title, and it helps to rank the individual item that are more relevant to the query, eventually leading to a better user experience.

3.4 Topic Modeling

To further improve the framework, topic modelling is the final pieces to the puzzle. Topic modelling can be used to further mine topics that are hidden in the data, the topic mined can be then used to tag the posts providing extra information when a search is made on our framework.

Topic modeling is a natural language processing technique used to identify and extract the main topics that are discussed in a large collection of text documents. It is a process of automatically discovering the topics that are present in a text corpus by grouping together words that frequently occur together. The resulting topics can then be used for various downstream applications such as text classification, information retrieval, and summarization.

In the context of chatbots like ChatGPT, topic modeling can be used to identify the main topics that are being discussed in a conversation and use that information to generate more relevant and coherent responses. For example, if the conversation is about a particular news article, the topic model can extract the main topics discussed in the article and use that information to generate a summary or to provide additional information on the same topic.

In FYP2, we hope to port over some of the ideas discussed above to further enhance the framework proposed in this work.

3.5 Real Time Capabilities Issue

One of the big issues for this framework to be generalized is the dataset. The dataset has to be generalized enough to be able to answer any query, that is simply impossible considering the wide range of the tehcnical field.

Real time scraping to populate the dataset is not a good idea as it will be very costly, time consuming, and cpu intensive, as our workers have to scale up to the demand of the users.

Therefore, two solutions are created to solve the problem, the first solution is to build a FAQs repository to store FAQs for further usage, and the second solution is to predict the next faq that will be asked.

3.6 FAQs Repository

With the first solution, a FAQs repository is needed to be built. The FAQs repository will be a collection of questions and answers that are related to the technical field. The way it works is that whenever a user asks a question, the framework will check if the question is in the FAQs repository, if it is, the framework will return the answer from the FAQs repository. If it's not, the framework will then scrap all the related posts from stackoverflow, and do inferencing. The summarized result will be stored in the FAQs repository, and the user or anyone will be able to access the answer from the FAQs repository next time they ask the same question.

3.7 FAQs Repository - Architecture

In the below figure, the architecture of the FAQs repository is presented. The FAQs repository is a database that stores the questions and answers. The database is populated by the system, and the framework will check the database whenever a user asks a question.

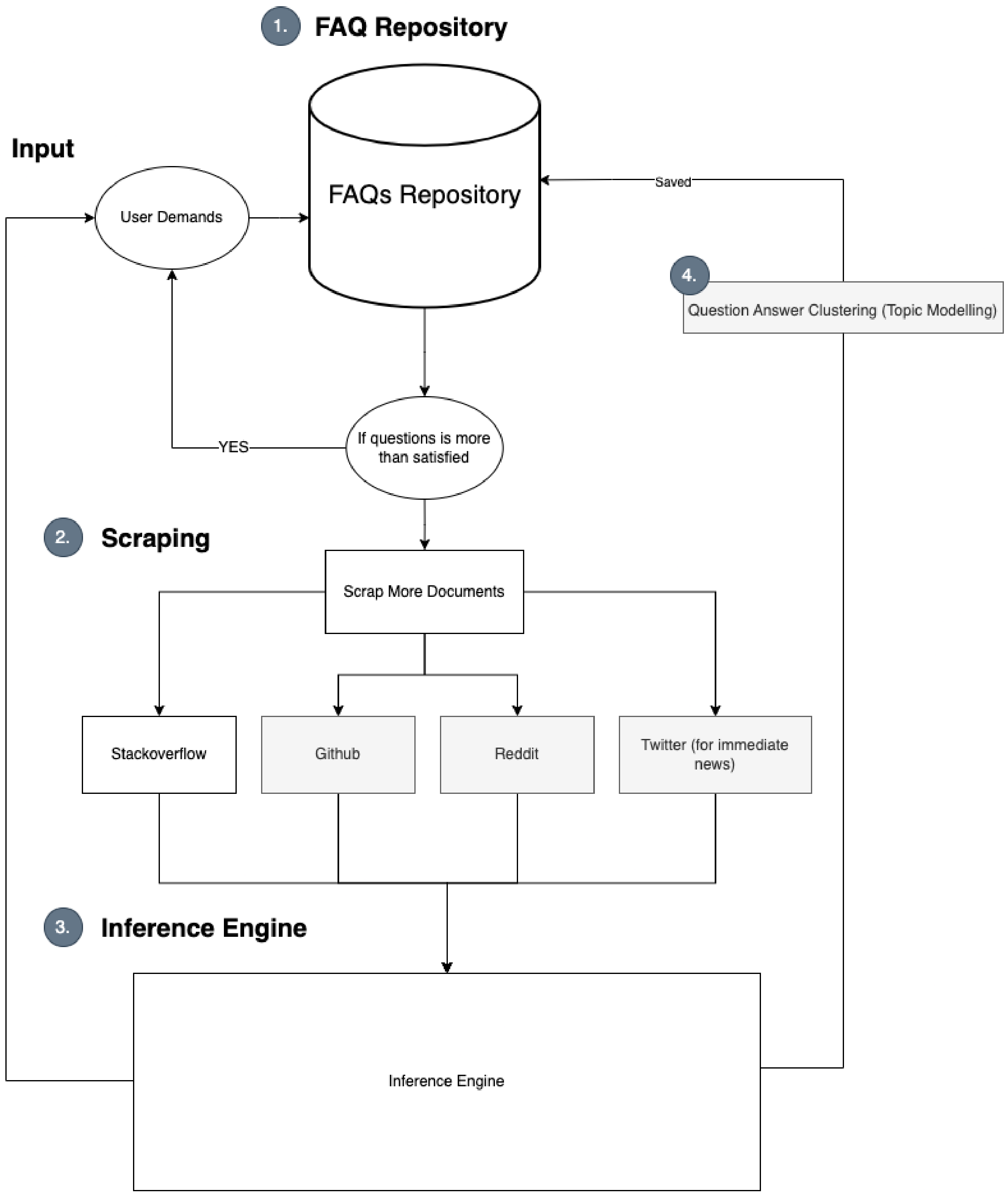


Figure 3.11: FAQ Repository Architecture

3.8 FAQs Repository - Approach

With the above architecture, the framework will be functioning as stated below:

1. The user asks a question

2. The framework will check if the question is in the FAQs repository

3. If the question is in the FAQs repository, the framework will return the answer from the FAQs repository

4. If the question is not in the FAQs repository, the framework will then scrap all the related posts from stackoverflow, and do inferencing

5. The summarized result will be stored in the FAQs repository, and the user or anyone will be able to access the answer from the FAQs repository next time they ask the same question.

With this architecture, the framework ensures that it is not only able to answer the question that is asked before, but also able to answer the question that is not asked before, also it can ensure that the framework will not scrap the same question again and again, as it's already in the FAQs repository.

3.9 FAQs Repository - Database Design

The database of choice is MongoDB, as it's a NoSQL database, and it's very easy to scale up. As a plus point, MongoDB being a NoSQL database gives us the flexibility to store the data in anyway we wanted, and the structure of the data can be altered whenever.

The database will be structured as below:

```
ISent {
  p_score: number;
  n_score: number;
  nu_score: number;
  comparative: number;
  tokens: string[];
  words: string[];
  positive: string[];
  negative: string[];
  neutral: string[];
}

IAns {
  answer: string;
  tags: string[];
  upvotes: number;
  downvotes: number;
  sentiment: ISent;
}

IFaq {
  question: string;
  answer: IAns[];
  tags: string[];
}
```

Figure 3.12: FAQ Database Structure

3.10 FAQs Repository - System Design

The entire backend will be built using NodeJs, as it's a very popular language, and it's very easy to scale. We'll be using a boilerplate created by the author of this thesis, which consists of multi-threading and clustering capabilities. These factors will help us to scale up the framework whenever there's a need to. The boilerplate is available at `https://github.com/Shaunmak1214/cluster-node-auth-sql-boilerplate.git`. A few notable perks that the boilerplate has is that it has a built in authentication system, and it has a built in SQL database, and a whole lot more. All of the important features of the boilerplate will be listed below:

- Clustering in Node.js
- Server with Express.js
- Database schema and models using Sequelize ORM
- User authentication with JWT
- StandardJs for coding standards and styling
- Request validation using Express-validator
- Morgan and Winston for server side logging
- Swagger for API documentation

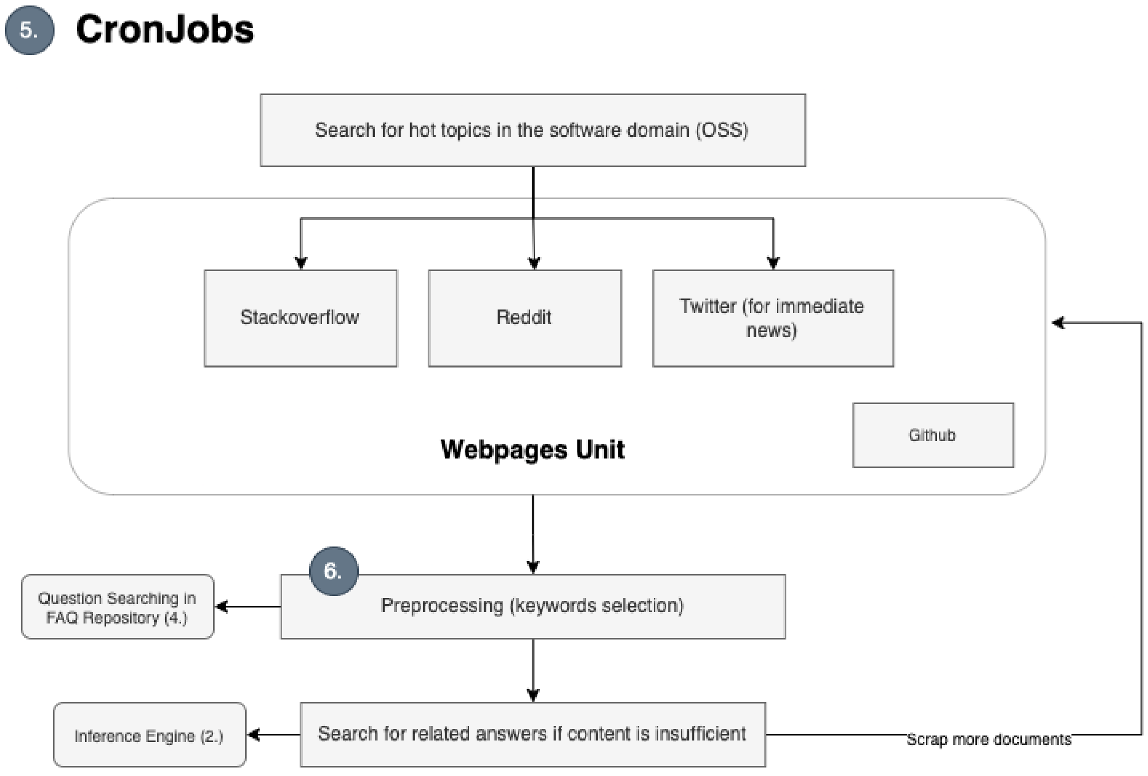
To host the backend, AWS EC2 is used, as it's the best cheap option, and it's very easy to scale according to user demands. Amazon Elastic Compute Cloud (Amazon EC2) offers the broadest and deepest compute platform, with over 500 instances and choice of the latest processor, storage, networking, operating system. Laslty to protect the servers from ddos attacks, AWS WAF is used , which is a web application firewall that helps protect a web applications from common web exploits that could affect application availability, compromise security, or consume excessive resources. AWS WAF gives one control over which traffic to allow or block to a web applications by defining customizable web security rules. One can use AWS WAF to create security rules that block common attack patterns, such as SQL injection or cross-site scripting, and rules that filter specific types of traffic. AWS WAF is integrated with AWS Shield, a managed DDoS protection service that safeguards web applications running on AWS. AWS Shield Standard provides always-on detection and automatic inline mitigations that minimize application downtime and latency, and AWS Shield Advanced provides distributed denial of service (DDoS) attack protection, proactive engagement with AWS support, and usage metrics.

3.11 CronJobs - Predictive FAQ Generation System

While the first solution – FAQ Repository 3.6 is able to solve the problem, there's still a concerning factor. The FAQs repository solution works well if a question is already asked by someone, but what if a question is not asked by anyone before? The framework will not be able to answer the question, and the user will have to wait for the lenghty process for the framework to scrap, process, analyze, rank and summarize the data for the user.

To solve this problem, a cronjob that runs the framework every 24 hours will be implemented, the framework will scrap, process, analyze, rank and summarize the data, and store it in the FAQs repository. This approach, assumes and rightfully predicts the newest, latests tech questions around four of the platforms, Stackoverflow, Reddit, Twitter and Github. The predicted topic is then used to scrap more data from the same platform, and hopefully it suffices the user's needs.

3.12 CronJobs - Predictive FAQ Generation System - Architecture


5.
CronJobs
Search for hot topics in the software domain (OSS)
Stackoverflow
Reddit
Twitter (for immediate news)
Github
Webpages Unit
6.
Preprocessing (keywords selection)
Question Searching in FAQ Repository (4.)
Search for related answers if content is insufficient
Inference Engine (2.)
Scrap more documents
Figure 3.13: CronJobs Architecture

3.13 CronJobs - Predictive FAQ Generation System - Approach

With the above architecture, the framework will be functioning as stated below:

1. The framework will scrap all the posts from stackoverflow, reddit, twitter and github every 24 hours

2. Perform topic modeling on the scrapped data, Extracting keywords

3. If dataset is not sufficient (metrics will be explored), scrap more documents related to the topic.

4. Process, analyze, rank and summarize the data

5. Store the summarized data in the FAQs repository

Which the proposed architecture, the proposed framework will populate itself with the most recent data, and in the hopes where the framework is able to fit with the user's needs.

CHAPTER 4

PRELIMINARY RESULTS AND DISCUSSION

4.1 Data Exploration

Data exploration refers to the process of analyzing, cleaning, transforming, and modeling data with the goal of discovering valuable insights and understanding the underlying structure of the data. It is an essential step in the data science process and involves visualizing and summarizing data, identifying patterns, outliers and anomalies, and checking for missing or duplicate data. Data exploration helps you get a sense of what your data looks like, identify potential problems, and make informed decisions about how to prepare the data for further analysis. By exploring your data, you can gain insights into the relationships and patterns that exist within the data and gain a better understanding of the problem you are trying to solve.

4.1.1 Unique Posts

First we should look at how many unique posts there are. This is important since we want to make sure that we have enough data to work with. After analyzing the data, we can see that there are 660 unique posts. This is a good sign since it means that we have enough data to work with.

4.1.2 Post Time Distribution

In order to analyze the longetivity of the current topic on stackoverflow, we will plot the time span distribution of all postings as a whole.

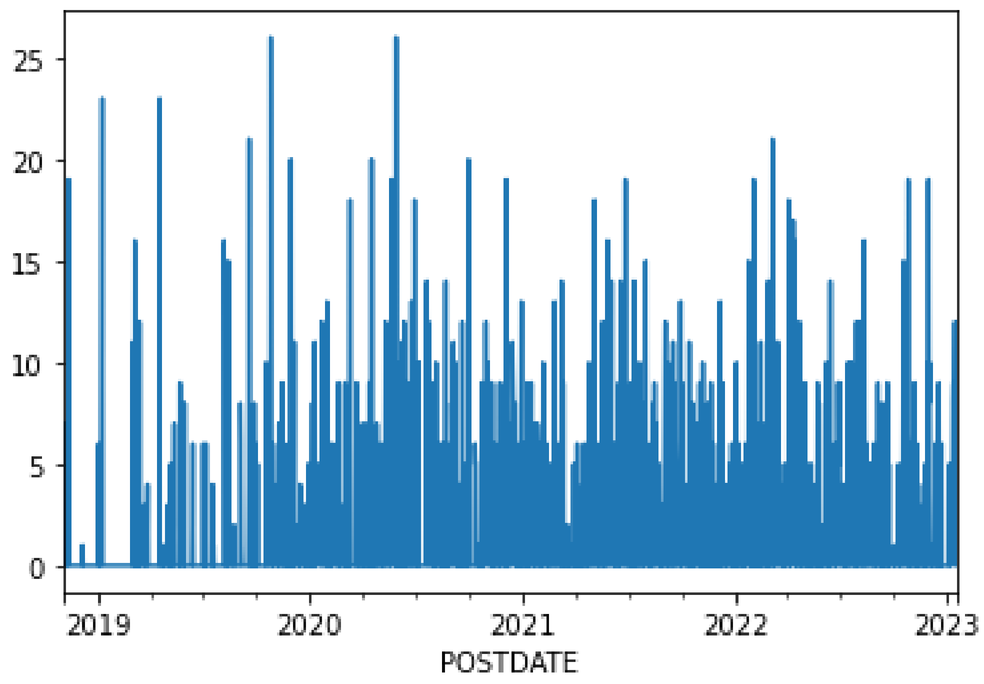


Figure 4.1: Post time distribution

As you can see on the above Figure 4.1, the number of posts per day is pretty constant. This is a good sign since it means that the number of people using the React UseEffect is still there and therefore it might gives us a better result.

4.1.3 Vote Counts Distribution

Next, we will look at the votecounts distribution across all of the dataset as a whole, it is important to note that the votecounts is the number of votes that a post has received, the higher the votecounts, the more popular the post is. We can probably inference that the more popular the post is, the more likely it is to be answered, and the more likely it is to be answered, the more likely it is to be answered correctly.

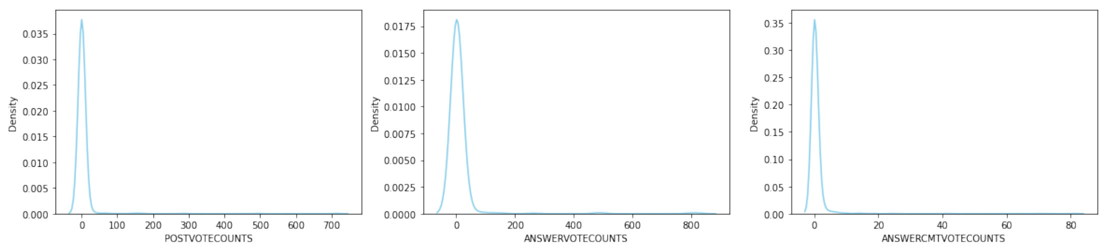


Figure 4.2: Vote counts distribution

As you can tell from the graph in Figure 4.2, the distribution is skewed to the left. This is not a good sign as it means that the majority of the posts have a low votecounts. With that being said, this motivates us to explore more data to be mined from the dataset and facilitate to our work.

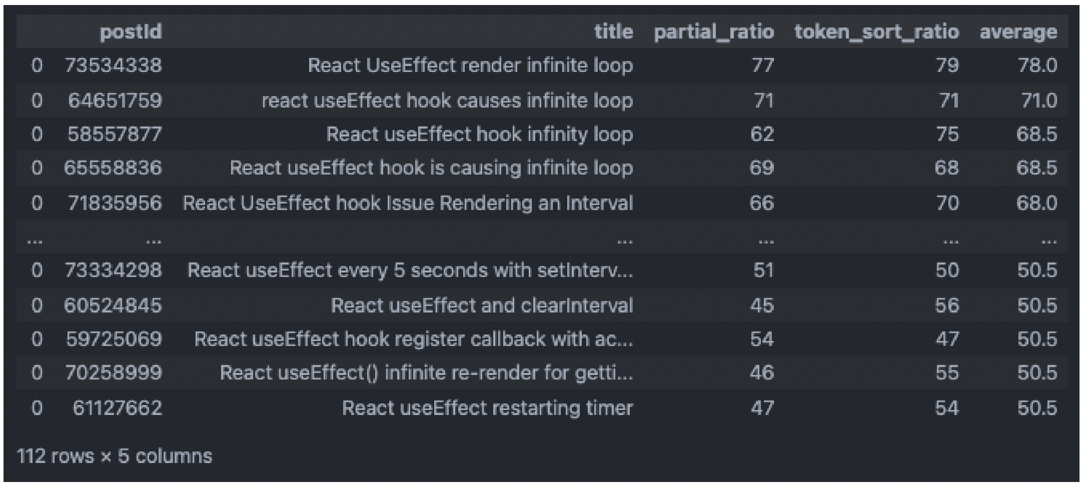
4.2 Post Matching

Post Matching is one of the most important part of this work. It is important to note that the post matching is not a trivial task. There are many ways to do it, in the following section we discuss the results that are obtained from 2 different methods that are noted on the "Proposed Framework" section 3.3.2 (b). In the below discussion, we used the query "How to solve useEffect hook rerenders infinitely?" to test and evaluate the performance of the post matching methods, it is important to note the post's title is the only string that is used to match with the query.

4.2.1 Fuzzy Wuzzy: Partial Ratio and Token Sort Ratio Combined

Fuzzy Wuzzy is a python library that provides string matching algorithms. The library provides a simple and efficient way to perform fuzzy string matching tasks, such as finding the similarity ratio between two strings, and finding the closest match for a given string from a list of potential candidates. A detailed discussion on how partial ratio and token sort ratio differ from each other is further highlighted 3.3.2 (b).

4.2.1 (a) Resutls:



```
      postId                                           title  partial_ratio  token_sort_ratio  average
0   73534338                  React UseEffect render infinite loop             77                79     78.0
0   64651759               react useEffect hook causes infinite loop             71                71     71.0
0   58557877                   React useEffect hook infinity loop             62                75     68.5
0   65558836           React useEffect hook is causing infinite loop             69                68     68.5
0   71835956   React UseEffect hook Issue Rendering an Interval             66                70     68.0
...      ...                                             ...            ...               ...      ...
0   73334298    React useEffect every 5 seconds with setInterv...             51                50     50.5
0   60524845                  React useEffect and clearInterval             45                56     50.5
0   59725069     React useEffect hook register callback with ac...             54                47     50.5
0   70258999      React useEffect() infinite re-render for getti...             46                55     50.5
0   61127662                   React useEffect restarting timer             47                54     50.5

112 rows × 5 columns
```

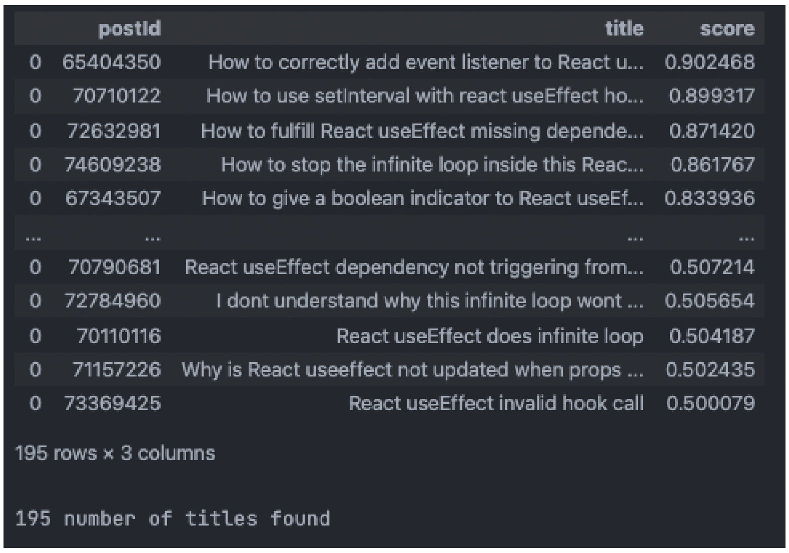
Figure 4.3: Partial Ratio and Token Sort Ratio Combined Results

Using the partial ratio method combined with token sort ratio, it is able to yield 112 posts title to match with the query.

4.2.1 (b) Spacy Similarity

Spacy is a python library that provides a lot of NLP tools. The library provides a simple and efficient way to perform NLP tasks, such as finding the similarity ratio between two strings, and finding the closest match for a given string from a list of potential candidates. A detailed discussion on how spacy similarity differs from other methods is further highlighted 3.3.2 (b).

4.2.1 (c) Results:



```
       postId                                               title     score
0    65404350   How to correctly add event listener to React u...  0.902468
0    70710122   How to use setInterval with react useEffect ho...  0.899317
0    72632981   How to fulfill React useEffect missing depende...  0.871420
0    74609238      How to stop the infinite loop inside this Reac...  0.861767
0    67343507   How to give a boolean indicator to React useEf...  0.833936
...       ...                                                 ...       ...
0    70790681  React useEffect dependency not triggering from...  0.507214
0    72784960     I dont understand why this infinite loop wont ...  0.505654
0    70110116                     React useEffect does infinite loop  0.504187
0    71157226  Why is React useeffect not updated when props ...  0.502435
0    73369425                       React useEffect invalid hook call  0.500079

195 rows × 3 columns

195 number of titles found
```

Figure 4.4: Spacy Results

A total of 195 posts are matched using the spacy similarity method. Indicating a better matching performance than the fuzzy wuzzy method.

4.3 Proprocessing

4.3.1 Evaluate Examples

4.3.1 (a) Example 1

Imagine if that was a Web Socket and we were scheduling a new heartbeat tick every time we remounted; the server would be very angry at our apps heart palpitations.

Figure 4.5: Examples for Evaluation Uses: Example 1

The text above was used to test and evaluate the following sections 4.3.2 4.3.3

4.3.1 (b) Example 2

```
'\n<p>You have to convert the response to json <a
    href="https://google.com">Please Look at this link</a> with
    await <a href="https://google.com">await</a>
    response.json();\nand then use setState.</p>\n\n<pre
    class="lang-js s-code-block"><code class="hljs
    language-javascript"><span class="hljs-title
    function_">useEffect</span>(<span class="hljs-function">()
    =&gt;</span> { \n <span class="hljs-variable
    language_">console</span>.<span class="hljs-title
    function_">log</span>(<span class="hljs-string">"useEffect
    TopTen has been called!"</span>); \n <span
    class="hljs-keyword">const</span> <span class="hljs-title
    function_">fetchdata</span> = <span
    class="hljs-keyword">async</span> (<span
    class="hljs-params"></span>) =&gt; {\n <span
    class="hljs-keyword">const</span> response = <span
    class="hljs-keyword">await</span> api.<span class="hljs-title
    function_">topTen</span>(); <span class="hljs-comment">//
    this calls axios(url)</span>\n <span
    class="hljs-keyword">const</span> responseData = <span
    class="hljs-keyword">await</span> response.<span
    class="hljs-title function_">json</span>();\n <span
    class="hljs-title function_">setLoading</span>(<span
    class="hljs-literal">false</span>);\n <span class="hljs-title
    function_">setTopten</span>(responseData.<span
    class="hljs-property">data</span>); \n <span
    class="hljs-title
    function_">setError</span>(responseData.<span
    class="hljs-property">error</span>); \n };\n\n fetchdata ();
    \n}, []);\n</code></pre>\n '
```

Figure 4.6: Examples for Evaluation Uses: Example 2

The text shown is a non-edited scraped html from stackoverflow. It was used to evaluate the following sections 4.3.4 4.3.5 4.3.6

4.3.2 Stopwords Removal

Stopwords are identified as the common words that exists in a language, they are usually removed from the text to give more importance to other words in the text. Stopwords lists can be subjective, stopwords list differ itself from one language to another, also they are different when treating texts from different domains. A detailed discussion on this domain is further highlighted 3.3.2 (d).

4.3.2 (a) Results:

The results after using the stopwords removal is as follows:

```
imagine web socket scheduling new heartbeat tick every time remounted; server would
angry apps heart palpitations.
```

Figure 4.7: Stopwords Removal Results

4.3.3 Stemming vs Lemmatization

A comparison of the results of stemming and lemmatization is shown below. Stemming is the process of reducing inflected (or sometimes derived) words to their word stem, base or root form—generally a written word form. Lemmatization is the process of grouping together the inflected forms of a word so they can be analysed as a single item, identified by the word’s lemma, or dictionary form. A detailed discussion on this domain is further highlighted 3.3.2 (j).

4.3.3 (a) Stemming Results:

```
imagin if that wa a web socket and we were schedul a new
    heartbeat tick everi time we remount ; the server would be veri
    angri at our app heart palpit .
```

Figure 4.8: Stemming Results

4.3.3 (b) Lemmatization Results:

```
Imagine if that wa a Web Socket and we were scheduling a new
    heartbeat tick every time we remounted ; the server would be
    very angry at our apps heart palpitation .
```

Figure 4.9: Lemmatization Results

In this work, lemmatization is chosen. Lemmatization has proven to be more effective than stemming if context of the text is taken into consideration.

4.3.4 Extracting URL

4.3.4 (a) Results:

```
[('https://google.com', 'Please Look at this link'), ('https://google.com', 'await')]
```

Figure 4.10: Extracting URL Results

The url and the text that is linked to the url is extracted and shown in the results.

4.3.5 Code Blocks Removal

4.3.5 (a) Results:

```
'\n<p>You have to convert the response to json <a
    href="https://google.com">Please Look at this link</a> with
    await <a href="https://google.com">await</a>
    response.json();\nand then use setState.</p>\n\n<pre
    class="lang-js s-code-block"></pre>\n '
```

Figure 4.11: Code Blocks Removal Results

In this step, the codeblocks is removed from the original html by detecting the tags and classes used.

4.3.6 HTML Tags Removal

4.3.6 (a) Results:

```
'\nYou have to convert the response to json Please Look at this
    link with await await response.json();\nand then use
    setState.\n\nuseEffect(() =&gt; { \n console.log("useEffect
    TopTen has been called!"); \n const fetchdata = async ()
    =&gt; {\n const response = await api.topTen(); // this calls
    axios(url)\n const responseData = await response.json();\n
    setLoading(false);\n setTopten(responseData.data); \n
    setError(responseData.error); \n };\n\n fetchdata (); \n},
    []);\n\n '
```

Figure 4.12: HTML Tags Removal Results

In this step, the html tags is removed from the original html by detecting the tags.

4.3.7 Combining all proprocessing techniques

By combining all of the proprocessing techinques used above 4.3.2 4.3.3 4.3.4 4.3.5 4.3.6, an evaluation is made using Example 2 4.3.1 (b)

```
You have to convert the response to json with await
    response.json();and then use setState.
```

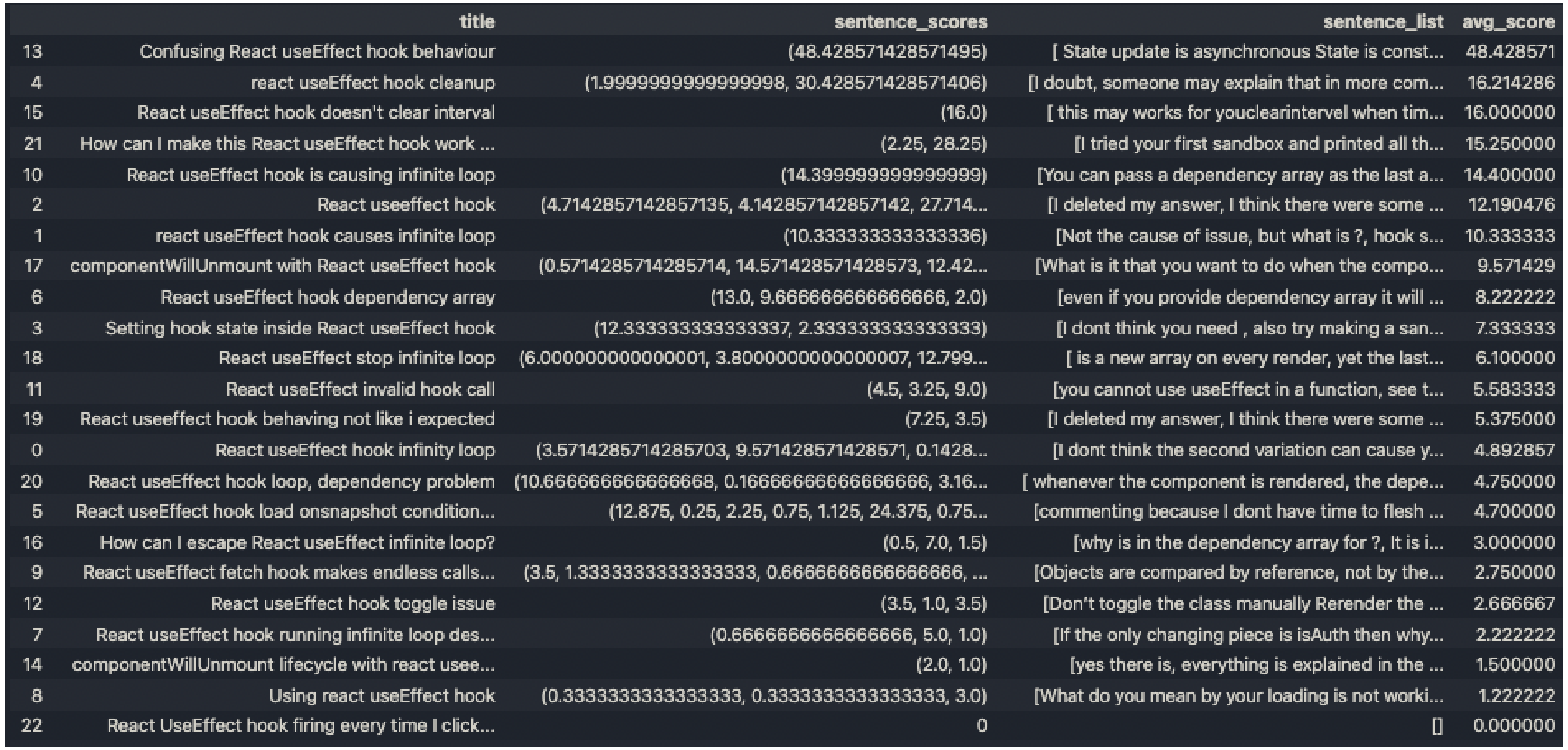
Figure 4.13: Combining all the methods used

The preprocessing techniques used above has removed the html tags, codeblocks, stop-words, and lemmatized the text. The text is now ready to be used for the following pipeline

4.4 Sentence Scoring

In order to rank sentences in a document, sentence scoring is used in this work, the mechanics used to achieve such scoring is highlighted in 3.3.2 (k). The results of the sentence scoring is shown below:

4.4.1 Sentence Scoring Results:



	title	sentence_scores	sentence_list	avg_score
13	Confusing React useEffect hook behaviour	(48.428571428571495)	[State update is asynchronous State is const...	48.428571
4	react useEffect hook cleanup	(1.9999999999999998, 30.428571428571406)	[I doubt, someone may explain that in more com...	16.214286
15	React useEffect hook doesn't clear interval	(16.0)	[this may works for youclearintervel when tim...	16.000000
21	How can I make this React useEffect hook work ...	(2.25, 28.25)	[I tried your first sandbox and printed all th...	15.250000
10	React useEffect hook is causing infinite loop	(14.399999999999999)	[You can pass a dependency array as the last a...	14.400000
2	React useeffect hook	(4.7142857142857135, 4.142857142857142, 27.714...	[I deleted my answer, I think there were some ...	12.190476
1	react useEffect hook causes infinite loop	(10.333333333333336)	[Not the cause of issue, but what is ?, hook s...	10.333333
17	componentWillUnmount with React useEffect hook	(0.5714285714285714, 14.571428571428573, 12.42...	[What is it that you want to do when the compo...	9.571429
6	React useEffect hook dependency array	(13.0, 9.666666666666666, 2.0)	[even if you provide dependency array it will ...	8.222222
3	Setting hook state inside React useEffect hook	(12.333333333333337, 2.3333333333333333)	[I dont think you need , also try making a san...	7.333333
18	React useEffect stop infinite loop	(6.000000000000001, 3.8000000000000007, 12.799...	[is a new array on every render, yet the last...	6.100000
11	React useEffect invalid hook call	(4.5, 3.25, 9.0)	[you cannot use useEffect in a function, see t...	5.583333
19	React useeffect hook behaving not like i expected	(7.25, 3.5)	[I deleted my answer, I think there were some ...	5.375000
0	React useEffect hook infinity loop	(3.5714285714285703, 9.571428571428571, 0.1428...	[I dont think the second variation can cause y...	4.892857
20	React useEffect hook loop, dependency problem	(10.666666666666668, 0.16666666666666666, 3.16...	[whenever the component is rendered, the depe...	4.750000
5	React useEffect hook load onsnapshot condition...	(12.875, 0.25, 2.25, 0.75, 1.125, 24.375, 0.75...	[commenting because I dont have time to flesh ...	4.700000
16	How can I escape React useEffect infinite loop?	(0.5, 7.0, 1.5)	[why is in the dependency array for ?, It is i...	3.000000
9	React useEffect fetch hook makes endless calls...	(3.5, 1.3333333333333333, 0.6666666666666666, ...	[Objects are compared by reference, not by the...	2.750000
12	React useEffect hook toggle issue	(3.5, 1.0, 3.5)	[Don't toggle the class manually Rerender the ...	2.666667
7	React useEffect hook running infinite loop des...	(0.6666666666666666, 5.0, 1.0)	[If the only changing piece is isAuth then why...	2.222222
14	componentWillUnmount lifecycle with react usee...	(2.0, 1.0)	[yes there is, everything is explained in the ...	1.500000
8	Using react useEffect hook	(0.3333333333333333, 0.3333333333333333, 3.0)	[What do you mean by your loading is not worki...	1.222222
22	React UseEffect hook firing every time I click...	0	[]	0.000000

Figure 4.14: Sentence Scoring Results

4.5 Rank Posts: Part 1

According to the framework proposed in 3.1, the rankings can be evaluated after the sentence scoring is done. The results can be evaluated by using a wordcloud to determine the most important words in selected posts. The wordcloud is shown below:

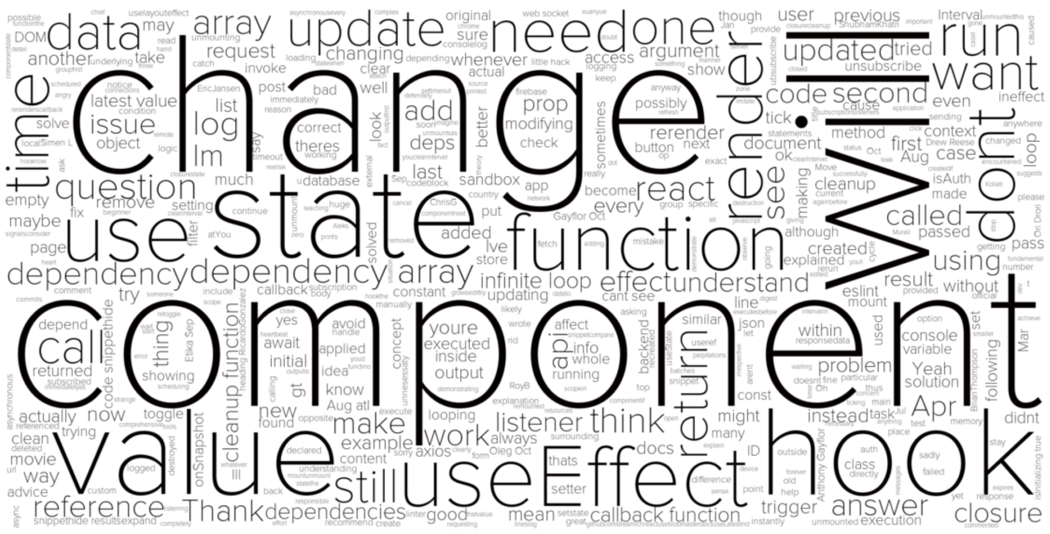


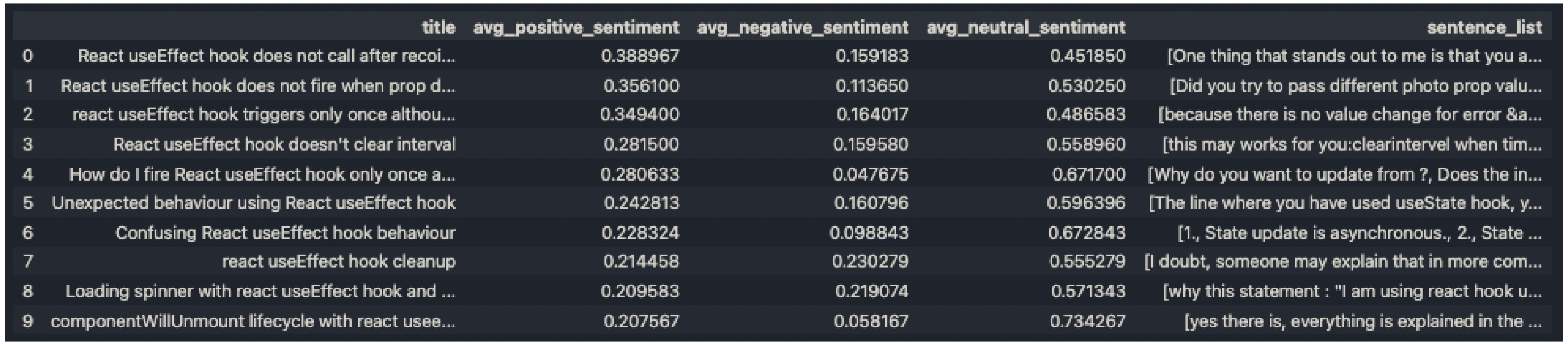
Figure 4.15: Word Cloud 1st Post Ranking Results

The word cloud generated this round is quite messy, the important keywords are quite a few, this indicates that the ranking framework is not effective enough. This is expected as the ranking framework is still in its early stages.

4.6 Sentiment Analysis

To further rank the sentences, sentiment is another factor that can provide usefull pieces of information to rank the sentences, 3.3.2 (l). Note that the list of posts are being filtered in the previous step 4.4, the filtering basically filters out any posts that have a negative or nan score on the previous step.

4.6.1 Sentiment Analysis Results:



	title	avg_positive_sentiment	avg_negative_sentiment	avg_neutral_sentiment	sentence_list
0	React useEffect hook does not call after recoi...	0.388967	0.159183	0.451850	[One thing that stands out to me is that you a...
1	React useEffect hook does not fire when prop d...	0.356100	0.113650	0.530250	[Did you try to pass different photo prop valu...
2	react useEffect hook triggers only once althou...	0.349400	0.164017	0.486583	[because there is no value change for error &a...
3	React useEffect hook doesn't clear interval	0.281500	0.159580	0.558960	[this may works for you:clearintervel when tim...
4	How do I fire React useEffect hook only once a...	0.280633	0.047675	0.671700	[Why do you want to update from ?, Does the in...
5	Unexpected behaviour using React useEffect hook	0.242813	0.160796	0.596396	[The line where you have used useState hook, y...
6	Confusing React useEffect hook behaviour	0.228324	0.098843	0.672843	[1., State update is asynchronous., 2., State ...
7	react useEffect hook cleanup	0.214458	0.230279	0.555279	[I doubt, someone may explain that in more com...
8	Loading spinner with react useEffect hook and ...	0.209583	0.219074	0.571343	[why this statement : "I am using react hook u...
9	componentWillUnmount lifecycle with react usee...	0.207567	0.058167	0.734267	[yes there is, everything is explained in the ...

Figure 4.16: Sentiment Anlysis Results

4.7 Rank Posts: Part 2

The wordcloud is shown below:

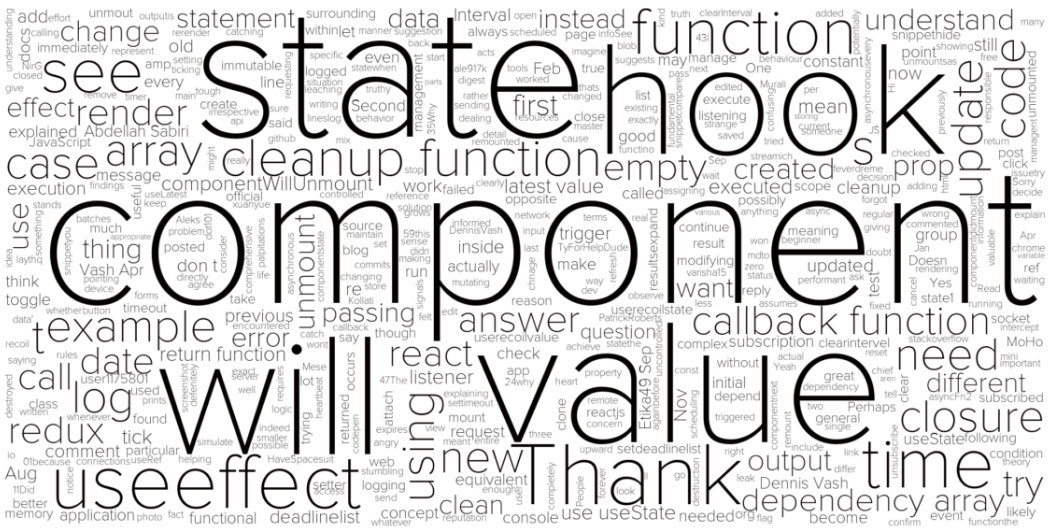


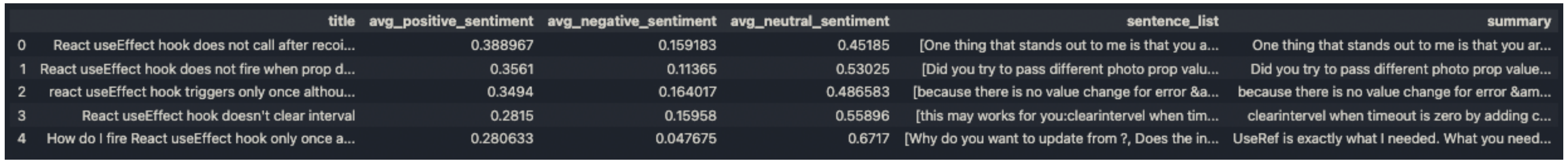
Figure 4.17: Word Cloud 2nd Post Ranking Results

The word cloud generated on this section suggests that the important words from the first posts are slighly more exagerated in the second post. This is a good indication of the effectiveness of the ranking system, because the unrelated posts are removed and the related posts are ranked higher.

4.8 Summarization

The final step is to summarize the top 5 posts ranked by the previous step 4.6. The detailed explaination of the summarization process and the model used is noted in 3.3.2 (n)

4.8.1 Summarization Results:



	title	avg_positive_sentiment	avg_negative_sentiment	avg_neutral_sentiment	sentence_list	summary
0	React useEffect hook does not call after recoi...	0.388967	0.159183	0.45185	[One thing that stands out to me is that you a...	One thing that stands out to me is that you ar...
1	React useEffect hook does not fire when prop d...	0.3561	0.11365	0.53025	[Did you try to pass different photo prop valu...	Did you try to pass different photo prop value...
2	react useEffect hook triggers only once althou...	0.3494	0.164017	0.486583	[because there is no value change for error &a...	because there is no value change for error &am...
3	React useEffect hook doesn't clear interval	0.2815	0.15958	0.55896	[this may works for you:clearintervel when tim...	clearintervel when timeout is zero by adding c...
4	How do I fire React useEffect hook only once a...	0.280633	0.047675	0.6717	[Why do you want to update from ?, Does the in...	UseRef is exactly what I needed. What you need...

Figure 4.18: Sentiment Anlysis Results

4.9 Rank Posts: Part 3

The wordcloud is shown below:

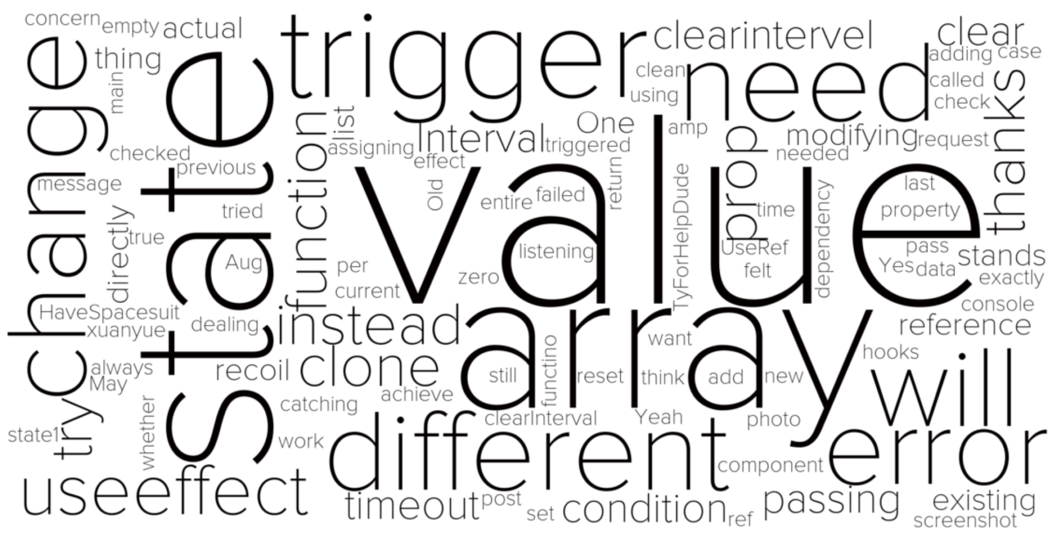


Figure 4.19: Word Cloud 3rd Post Ranking Results

The final word cloud gives us a more clearer view on the most important words in the top 5 posts, the words chosen are definitely less noisy than the previous wordclouds.

4.10 Survey

The survey is conducted to determine the effectiveness of the system, the survey is conducted on 3 people, 3 of them are experts in the field of "react useeffect" which is the topic of the dataset scraped from stackoverflow. The survey is conducted by asking the experts to come up with the query related to the topic "react useeffect" and to determine if the results of the framework is relevant to the topic of the question and if they're useful to the experts. The results of the survey is shown below:

4.10.1 Expert 1

The query asked by expert 1 is:

```
How to fetch data using the useEffect Hook?
```

Figure 4.20: Query asked by expert 1

Feedback from expert 1:

```
The result is relevant to the question, but the summarized output
    is not useful.
```

Figure 4.21: Feedback from expert 1

4.10.2 Expert 2

The query asked by expert 2 is:

```
useEffect hook doesn’t work!
```

Figure 4.22: Query asked by expert 2

Feedback from expert 2:

```
It is able to find the post that are related to my prompt.
```

Figure 4.23: Feedback from expert 2

4.10.3 Expert 3

The query asked by expert 3 is:

```
How do i use the Use effect hook with use state.
```

Figure 4.24: Query asked by expert 3

Feedback from expert 3:

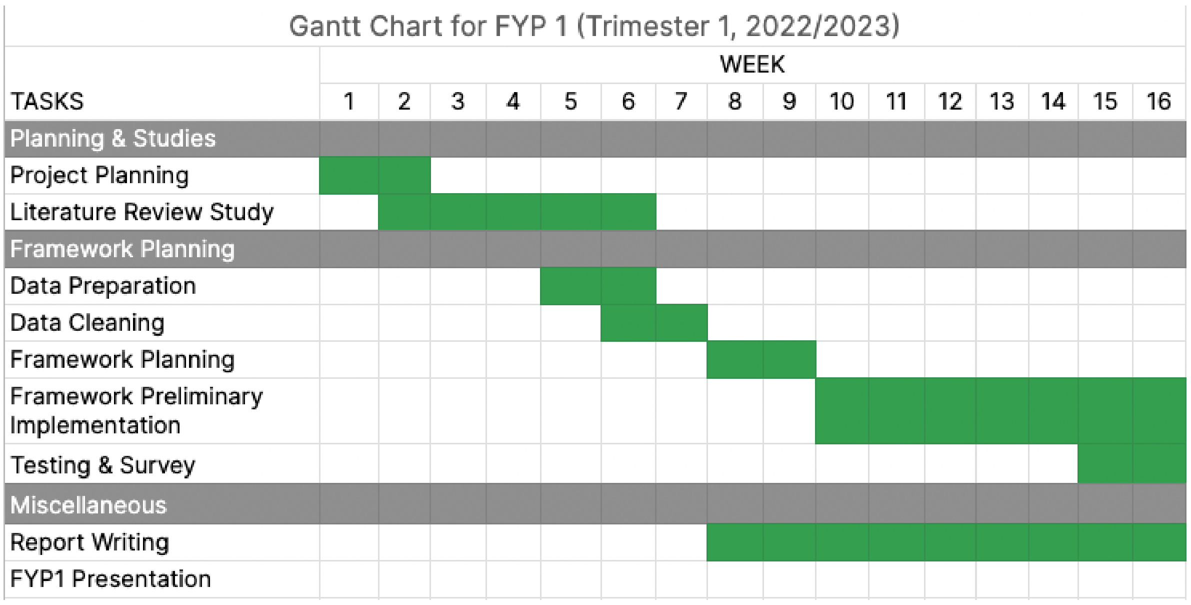
```
Maybe my query is abit out of context because useState is
    involved, but surprisingly the framework is able to find the post.
```

Figure 4.25: Feedback from expert 3

CHAPTER 5

IMPLMENTATION PLAN

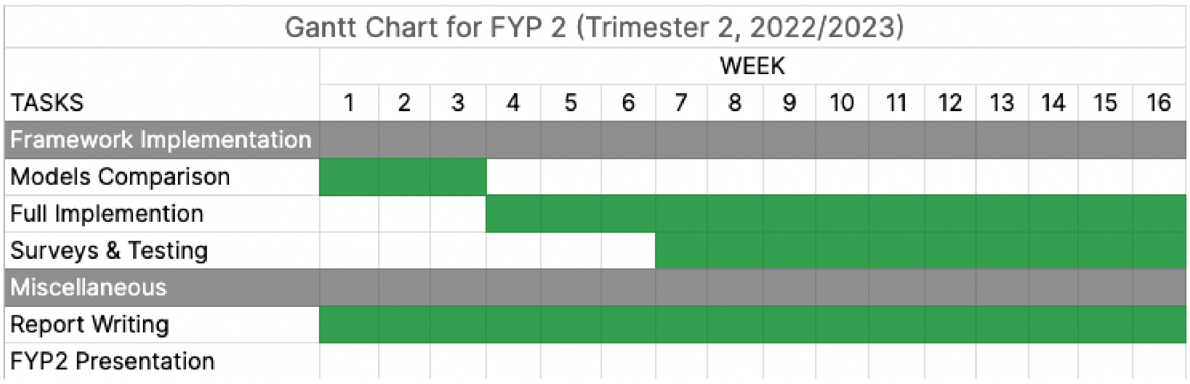
The implementation plan of this work can be divided into two parts, FYP1 and FYP2. The tasks are then spreaded out into the two parts. Both gantt charts indicate the tasks and the time frame for each task. The gantt chart for FYP1 is shown in Figure 5.1 and the gantt chart for FYP2 is shown in Figure 5.2.



Gantt Chart for FYP 1 (Trimester 1, 2022/2023)

	WEEK															
TASKS	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Planning & Studies																
Project Planning	🟩	🟩														
Literature Review Study		🟩	🟩	🟩	🟩	🟩										
Framework Planning																
Data Preparation					🟩	🟩										
Data Cleaning						🟩	🟩									
Framework Planning								🟩	🟩							
Framework Preliminary Implementation										🟩	🟩	🟩	🟩	🟩	🟩	🟩
Testing & Survey															🟩	🟩
Miscellaneous																
Report Writing								🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩
FYP1 Presentation																

Figure 5.1: Gantt Chart for FYP1



Gantt Chart for FYP 2 (Trimester 2, 2022/2023)																
	WEEK															
TASKS	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Framework Implementation																
Models Comparison	🟩	🟩	🟩													
Full Implemention				🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩
Surveys & Testing							🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩
Miscellaneous																
Report Writing	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩
FYP2 Presentation																

Figure 5.2: Gantt Chart for FYP2

The colored architecture framework also indicates the part of the framework that is being developed in FYP1 or FYP2. The architecture framework is shown in 3.1.

CHAPTER 6

CONCLUSIONS

6.1 Summary

Over the course of this research, we have scraped, compiled, cleaned, proprocess the dataset. A framework architecture to facilitate the process of finding the related post as an answer to the user prompt has been proposed. The framework is composed of 3 main components: scraping, proprocess and inferencing. 2 inferencing architecture had been proposed in the work, however only the first solution is built and tested with 3 experts that we've identified. One of the important tasks is to find a match of the question asked by the user with the posts in the forum. As of now, all techniques are based of NLP techniques such as FuzzyWuzzy and Spacy which was discussed in 3.3.2 (b) and tested in 4.2, however deep learning models that are discussed in the literature review section 2.1.3 will be tested in the future.

6.2 Challenges

Computation power is one of the main challenges in this work, scraping, sentiment analysis and summarization are computationally expensive tasks. Scraping takes a lot of time as the process is sequential, sentiment analysis and summarization however is slow because the model we're using is expected to be ran on a gpu-enabled machine.

6.3 Future Work

- The results have shown that the summarization of the texts in posts are possilbe, but the results are not satisfactory. Therefore others summarization algorithms should be tested.
- The framework is slow in general, each process in the pipeline takes time. As planned in the proposed framework an FAQ Repository will be built to store the results of the pipeline. This repository will be used to answer the questions in the future. Therefore the framework should be improved to be able to answer the questions in a reasonable time.
- Solution A is the only architecture we've built and tested in FYP 1, solution B will be built in FYP 2.
- Parralelization of the scraping process is to explored.
- Possibility of migrating to a fully cloud hosted solution is to be explored.
- Deep learning models for matching posts are to be tested.