JOIV : Int. J. Inform. Visualization, 8(1) - March 2024 425-433

INTERNATIONAL JOURNAL ON INFORMATICS VISUALIZATION

journal homepage : www.joiv.org/index.php/joiv

Forum Text Processing and Summarization

Yen-Wei Mak [a], Hui-Ngo Goh [a,*], Amy Hui-Lan Lim [a]

[a] *Faculty of Computing and Informatics, Multimedia University, Cyberjaya, 63100, Malaysia*

Corresponding author: [*]hngoh@mmu.edu.my

Abstract— Frequently Asked Questions (FAQs) are extensively studied in general domains like the medical field, but such frameworks are lacking in domains such as software engineering **and open-source communities. This research aims to bridge this gap by establishing the foundations of an automated FAQ Generation and Retrieval framework specifically tailored to the software engineering domain. The framework involves analyzing, ranking, performing sentiment analysis, and summarization techniques on open forums like StackOverflow and GitHub issues. A corpus of Stack Overflow post data is collected to evaluate the proposed framework and the selected models. Integrating state-of-the-art models of string-matching models, sentiment analysis models, summarization models, and the proprietary ranking formula proposed in this paper forms a robust Automatic FAQ Generation and Retrieval framework to facilitate developers' work. String matching, sentiment analysis, and summarization models are evaluated, and F1 scores of 71.31%, 74.90%, and 53.4% were achieved. Given the subjective nature of evaluations in this context, a human review is used to further validate the effectiveness of the overall framework, with assessments made on relevancy, preferred ranking, and preferred summarization. Future work includes improving summarization models by incorporating text classification and summarizing them individually (Kou et al, 2023), as well as proposing feedback loop systems based on human reinforcement learning. Furthermore, efforts will be made to optimize the framework by utilizing knowledge graphs for dimension reduction, enabling it to handle larger corpora effectively.**

Keywords— **Deep learning; forum processing; natural language processing; summarization; sentiment analysis.**

Manuscript received 5 Dec. 2022; revised 9 Jul. 2023; accepted 26 Nov. 2023. Date of publication 31 Mar. 2024.

I. INTRODUCTION

Many individuals have long considered finding answers from online forums time-consuming and arduous. This is primarily due to the scattered nature of the answers across various forums, making it challenging to locate relevant information. In the domain of software engineering, this issue becomes even more prominent. According to Microsoft CEO Satya Nadella, approximately 50 percent of searches fail to yield sufficient results [1]. This problem is particularly pronounced in software engineering, where identifying the precise and optimal solution for a given problem can be highly challenging.

While Automatic FAQ Generation, Question Generation, and Answer Retrieval techniques have emerged, most studies have focused on broad topics such as banking, healthcare, and others. Additionally, because there are typically several relevant posts to evaluate, and some articles might be lengthy, identifying crucial information in online posts can be time-consuming. As a result, this research aims to build upon prior investigations and adapt proven methodologies from other domains to the specific context of software engineering. Leveraging various natural language processing (NLP) techniques, including approaches like n-grams, this study presents a comprehensive framework for achieving Automatic FAQ Generation and Retrieval tailored explicitly to the software engineering domain.

Existing work in the domain of FAQ processing has predominantly centered on non-engineering fields, particularly the medical domain [2]. However, the focus of this research is primarily directed towards the software engineering field. By exploring the existing findings, this paper aims to transfer any applicable insights derived from previous studies. Consequently, extensive reading has been undertaken to ensure proficiency in handling technical languages and terminologies. FAQ retrieval plays a pivotal role in ranking question-answer pairs [3]. It involves retrieving the most relevant answers from an extensive collection based on a user's query. However, traditional methods for FAQ generation heavily rely on extensive manual classification and software engineering techniques [3]. Such approaches demand significant time and effort to be executed. This concern has been underscored by previous studies conducted by [4], [5], [6], [7]. Moreover,

manual classification and software engineering quality also impact the framework's performance.

Various attempts and research have been undertaken to address the issues and enhance existing methods. One promising approach involves automating the process of FAQ generation using NLP techniques[8]–[10]. [1], [3] have made notable advancements in this area by leveraging deep learning methods, specifically by combining Deep Matching Networks (DMN) and Multihop Attention Networks for FAQ retrieval. Deep Matching Network (DMN) is a deep learning model that utilizes two matrix inputs to generate matching scores. These matrices are constructed by computing the dot product of word embeddings from both the questions and answers. The DMN has shown effectiveness in capturing semantic relationships between questions and answers. On the other hand, Multihop Attention Networks have demonstrated their efficacy in reasoning tasks such as answering questions, which aligns with the focus of this paper. This network incorporates multiple "hops" of attention to gather information from the input and make predictions. It involves an encoding step to encode the input and a decoder network that iteratively attends to different parts of the input, ultimately generating an output.

[4] and [11] have proposed an approach encompassing the entire architecture for achieving Auto-Faq-Gen. This architecture includes web scraping, question construction, a ranking algorithm, and question generation. [5] and [12] have made notable strides in selecting, weighing, clustering, and ranking contextual keywords. These advancements aim to achieve question abstraction, thereby facilitating locating pertinent questions and their corresponding answers within open-source forums. The authors subsequently proposed a solution in the form of semi-automatic FAQ generation, which allows for improved organization and retrieval of information. A study by [13] looked into the usage of knowledge graph extra domain knowledge in generating a comprehensive list of FAQs.

Another challenge arises when the questions posed by users need help to easily be classified to align with the existing questions in the FAQ database. This is often due to differences in form or context between the user's questions and those already in the database [12]. For instance, the user's question may be in a different language or may be expressed in a different format. In addition to contextual variations, grammatical errors, and misspellings pose further obstacles when developing automated question-answering systems, as emphasized by [14]. These linguistic challenges necessitate robust techniques to handle diverse language usage and to interpret and respond to questions accurately. Context matching plays a crucial role in ranking the answers to user queries. However, the traditional approach of scoring similarity based on Levenshtein distance, as highlighted by [15], has limitations in effectively ranking answers. This is because Levenshtein distance fails to capture the semantic meaning of words, which is essential for accurate ranking.

Another concept worth considering is the Bag-of-Words (BOW) model, as discussed by [16]. BOW similarity matching algorithms can be applied to processed text, including steps such as stop word removal and stemming, calculating the similarity between the query and the answer. However, like Levenshtein distance, BOW cannot capture the semantic meaning of words. Consequently, it can lead to false conceptual similarity between the query and the answer. To overcome these limitations, more advanced techniques that consider semantic meaning, such as semantic matching models or neural network-based approaches, have been proposed in recent research to improve the accuracy of answer ranking in the context of user queries. Word knowledge or word embedding offers a potential solution to enhance traditional similarity-matching algorithms. [16] have proposed a method that incorporates word knowledge to enhance similarity matching. Their model connects a knowledge base to individual words, constructing a knowledge table encompassing raw words, hypernyms, synonyms, and associative concepts. This approach deviates from traditional similarity-matching algorithms by considering the semantic meaning of words rather than just the raw words themselves.

Stop words are commonly encountered in natural language data and are typically filtered out during or after text processing. However, the specific set of stop words used can vary across natural language processing tools, and no universal list applies to all applications. Technical languages have their own unique set of stop words, which differs from the general stop words list used in applications like the NLTK library [17], [18], [19]. To address the need for specific stop words for software engineering texts, [17] have developed a list tailored to this domain. The list is created using statistical identification techniques and evaluated by domain experts. The detection of phrases can be achieved using the algorithm proposed by [20]. This algorithm identifies frequently co-occurring words, allowing the detection of meaningful phrases within the text.

Sentiment analysis, or opinion mining, is a process used to determine the sentiment expressed in a piece of writing, classifying it as positive, negative, or neutral. It is commonly employed to gain insights into people's attitudes and opinions about various topics. For example, sentiment analysis can be applied to assess public sentiment toward a new movie or to understand the overall sentiment toward a newly launched product. While social media platforms like Twitter[21], Facebook[22], and Instagram[23] are frequently used as sources for sentiment analysis, this technique can be applied to any text data. In the context of government entities, [24] utilize sentiment analysis to investigate further and comprehend the needs and preferences of customers. Sentiment analysis can be performed at different levels of granularity, which refers to the level of detail at which sentiment is expressed. The three primary levels of granularity are sentence-level, document-level, and aspect-level. Considering the appropriate level of granularity is crucial as it impacts the specific type of sentiment analysis that can be conducted and the resulting insights that will be obtained [24].

Abstractive summarization involves generating new sentences that capture the original text's meaning. One common approach to abstractive summarization is using a sequence-to-sequence neural network, such as RNN or LSTM, which are well-suited for processing sequential data like text. The recurrent connections in these models enable them to maintain a hidden state that retains information from previous steps in the sequence, allowing them to capture contextual information as they read the input text. This enables the model to generate

concise and semantically relevant summaries [25]. Abstract Meaning Representation (AMR) is an RNN-based method introduced by [26]. It utilizes a neural network model that produces a single graph representing time series information in the text to generate abstractive summaries. Another approach is the Abstractive Text Summarization with Dual Learning (ATSDL) model, which combines the advantages of extractive and abstractive summarization. In this model, a sequence-to-sequence neural network is used to generate the summary. Then a sentence ranking model is employed to rank the sentences in the generated summary.

In this work, we focus on answering the following research questions:

- How effective is our question matching, sentiment analysis and summarization model contributing for FAQ in software engineering domains?
- Is the fine-tuned summarization model better than the base BART model?
- How do humans perceive the answer given by our proposed framework?

II. Materials and Method

The proposed framework is formulated based on a fundamental assumption. It is observed that post titles generally possess brevity and conciseness, serving as the primary focal point for individuals seeking solutions to their inquiries. Hence, it is reasonable to postulate that the title effectively captures the essence of the corresponding post. Given this perspective, it is crucial to acknowledge that all comments and answers should be confined to the body of the post itself, as it constitutes a comprehensive representation of the information. When deploying a ranking system, it becomes imperative to assess the post as a cohesive entity, considering both its content and the accompanying responses. Fig. 1 visually represents the proposed framework, illustrating its components and their interactions.

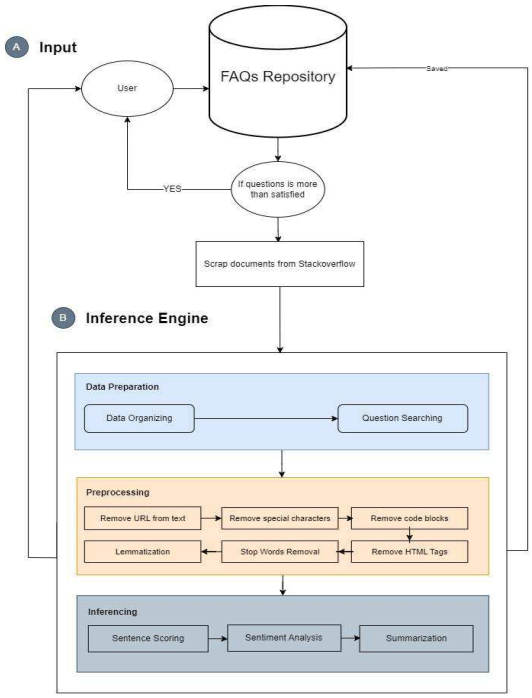

Fig 1 Proposed Framework

A. Input

The framework initiates with an input layer, which involves a user query. Subsequently, the system examines whether the FAQ repository contains pertinent information. If relevant information is available in the FAQ repository, it is directly dispatched to the user. However, if such information is not found, the entire cycle commences. A data gathering process is initially executed, whereby relevant Stack Overflow sites are scraped utilizing the Selenium tool and stored. The gathered data is then passed on to the subsequent step, the Inference Engine (B).

B. Inference Engine

1) Data Preparation:

In data organization, the information will be categorized into distinct entities based on their types, namely posts, comments, answers, and answer comments. This categorization serves the purpose of enhancing data management and facilitating subsequent analyses. It is necessary to undertake this step due to the inherent lack of organization in the provided CSV dataset, which hinders efficient data handling. Fig. 2 and 3 show the comparison between the unorganized and organized data.

```
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')
```

Fig. 2 Unorganized data

```
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')
```

Fig. 3 Organized data

Within the framework of string matching, three widely adopted methods are FuzzyWuzzy, spaCy and roberta-large-mnli model. FuzzyWuzzy is a battle-tested and state-of-the-art model renowned for its exceptional performance in approximate and partial string-matching tasks. It employs sophisticated techniques, such as Levenshtein distance calculations, to achieve remarkable results.

FuzzyWuzzy encompasses two distinct methods, partial ratio, and token sort ratio. The partial ratio method quantifies the similarity between two strings by assessing the ratio of the longest contiguous matching substrings. It effectively handles scenarios where matching substrings, rather than individual characters, holds significant relevance. Conversely, the token sort ratio method sorts the tokens within each string alphabetically and computes the similarity ratio based on the sorted token lists. This approach proves advantageous when comparing strings with different word orders, as it captures similarities that variations in word arrangement may obscure.

On the other hand, spaCy's similarity feature goes beyond surface-level textual matching by capturing the semantic nuances of words, phrases, and sentences. By leveraging comprehensive word vectors derived from extensive text corpora, spaCy deeply understands the intricate relationships and meanings between words. This enhanced comprehension significantly improves the accuracy and relevance of similarity scores, thus providing a robust foundation for a wide range of natural language processing tasks.

Another notable model in the realm of contextual string similarity is RoBERTa. It is a powerful pre-trained model that has been fine-tuned on the Multi-Genre Natural Language Inference (MNLI) dataset. In string matching, entailment refers to the logical relationship between two texts, where one text logically follows from or can be inferred from the other. Using RoBERTa, we can leverage its fine-tuned knowledge to assess the likelihood of one string entailing or implying the other. This approach enables us to capture surface-level similarities and the underlying meaning and context of the compared strings.

We employ the roberta-large-mnli pre-trained model, specifically trained to perform natural language inference tasks. Given two input texts, we encode them using the model's tokenizer. Subsequently, the encoded texts are processed through the RoBERTa model to obtain logits, representing the probabilities of different entailment labels. Applying a SoftMax function to these logits yields a probability distribution over the entailment labels. Finally, we extract the entailment probability associated with the "ENTAILMENT" label, which indicates the likelihood of the two texts being entailed.

2) Preprocessing:

To leverage the valuable information embedded in URLs and understand the context of the data, it is crucial to incorporate them into the dataset. URLs can provide additional insights and references related to the data being analyzed. Since the data is scraped and contains HTML tags, extracting the URLs can be easily achieved using the BeautifulSoup package.

By employing the capabilities of the BeautifulSoup package, the URLs present within the data can be extracted effectively. This allows for isolating the URLs from the rest of the text, enabling their separate storage in a dedicated column within the dataset. This organization facilitates easy access to the URLs for future reference and analysis. By storing the URLs in a separate column, the dataset maintains its structural integrity and allows for the establishment of connections between the data and the associated URLs. This integration of URLs allows researchers to explore additional information and enrich the understanding of the underlying context within the dataset. Table I shows the chronology of before and after of the URL removal.

TABLE I
ILLUSTRATION OF URL REMOVAL

Before URL Removal
’\n You have to convert the response to json Please Look at this link with await await response.json();\nand then use setState. \n\n<preclass="lang-js s-code-block"><code class="hljslanguage-javascript">useEffect(()=> { \n console.log("useEffectTopTen has been called!"); \n <spanclass="hljs-keyword">const</spanclass=> fetchdata = <spanclass="hljs-keyword">async</spanclass=> (<spanclass="hljs-params"></spanclass=>) => {\n <spanclass="hljs-keyword">const</spanclass=> response = <spanclass="hljs-keyword">await</spanclass=> api.topTen(); //this calls axios(url)\n <spanclass="hljs-keyword">const</spanclass=> responseData = await response.<span 83 class="hljs-title function_">json();\n <spanclass="hljs-title function_">setLoading</spanclass=>(<spanclass="hljs-literal">false</spanclass=>);\n setTopten(responseData.<spanclass="hljs-property">data</spanclass=>); \n setError(responseData.<spanclass="hljs-property">error</spanclass=>); \n };\n\n fetchdata ();\n}, []);\n</code></preclass=>\n ’
After URL Removal
’\n You have to convert the response to json with await response.json();\nand then usesetState. \n\n<preclass="lang-js s-code-block"><code class="hljslanguage-javascript">useEffect(()=> { \n console.log("useEffectTopTen has been called!"); \n <spanclass="hljs-keyword">const fetchdata = <spanclass="hljs-keyword">async (<spanclass="hljs-params">) => {\n <spanclass="hljs-keyword">const response = <spanclass="hljs-keyword">await api.topTen(); //this calls axios(url)\n <spanclass="hljs-keyword">const responseData = <spanclass="hljs-keyword">await response.<span83class="hljs-title function_">json();\n <spanclass="hljs-title function_">setLoading(<spanclass="hljs-literal">false);\n setTopten(responseData.<spanclass="hljs-property">data); \n <spanclass="hljs-titlefunction_">setError(responseData.<spanclass="hljs-property">error); \n };\n\n fetchdata ();\n},]);\n</code></pre>\n ’

Following the identification of special characters present in the dataset that is scraped from Stack Overflow, the subsequent step involves their removal. The dataset contains a variety of special characters, and it is crucial to address this issue as these characters can lead to undesired complications during analysis. The list of special characters identified within the dataset is [“(“, “)”,”,”, , License: CC BY-SA 4.0, segFault].

Removing special characters ensures data cleanliness and facilitates subsequent processing and analysis tasks. By eliminating these characters, the dataset becomes more standardized and amenable to further analysis. The removal of special characters is typically achieved through text preprocessing techniques, such as pattern matching and substitution. The framework recognizes the significance of removing code blocks from the data. While code blocks may not be a major concern in typical natural language processing tasks, they are prevalent in Stack Overflow, which serves as a platform for developers to seek assistance and share their programming knowledge. Given the specific focus on Stack Overflow data, the removal of code blocks becomes a crucial step in the preprocessing process to minimize noise and enhance the quality of the dataset as can be seen in Table II.

Code blocks within the data are typically enclosed within triple quotes ("```" or """"). This distinctive pattern simplifies the identification of code blocks within the pipeline, making it straightforward to recognize and subsequently eliminate them from the dataset. By removing code blocks, the framework aims to refine the dataset and ensure that the presence of code snippets does not influence the subsequent analyses and modeling efforts.

TABLE II
CODEBLOCKS REMOVAL

Before Codeblocks Removal
’\n You have to convert the response to json Please Look at this link withawait <a Ref="https://google.com">awaitresponse.json();\nand then use setState. \n\n<preclass="lang-js s-code-block"><code class="hljslanguage-javascript">useEffect(()=> { \n console.log("useEffectTopTen has been called!"); \n <spanclass="hljs-keyword">const fetchdata = <spanclass="hljs-keyword">async (<spanclass="hljs-params">) => {\n <spanclass="hljs-keyword">const response = <spanclass="hljs-keyword">await api.topTen(); //this calls axios(url)\n <spanclass="hljs-keyword">const responseData = <spanclass="hljs-keyword">await response.<span83class="hljs-title function_">json();\n <spanclass="hljs-title function_">setLoading(<spanclass="hljs-literal">false);\n setTopten(responseData.<spanclass="hljs-property">data); \n setError(responseData.error); \n };\n\n fetchdata ();\n}, []);\n</code></pre>\n ’

After Codeblocks Removal
’\nYou have to convert the response to json Please Look at this link with await await response.json();\nand then use setState.\n\nuseEffect(() => { \n console.log("useEffect TopTen has been called!"); \n const fetchdata = async () => {\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 ’

The presence of HTML tags in the data is a significant concern that requires careful attention and removal during the preprocessing phase. The scraping process involved in collecting the data may inadvertently result in the inclusion of HTML tags within the textual content. By leveraging the distinctive pattern of HTML tags enclosed within angle brackets ("<" and ">"), the framework can readily identify and remove these tags. This step is essential to eliminate potential interference, improve data integrity, and enhance readability for subsequent text processing and analysis tasks as shown in Table III.

TABLE III
COMPARISON OF HTML TAGS REMOVAL

Before HTML Tags Removal
’\n You have to convert the response to json Please Look at this link withawait await response.json();\nand then use setState. \n\n<preclass="lang-js s-code-block"><code class="hljslanguage-javascript">useEffect(()=> { \n console.log("useEffectTopTen has been called!"); \n <spanclass="hljs-keyword">const fetchdata = <spanclass="hljs-keyword">async (<spanclass="hljs-params">) => {\n <spanclass="hljs-keyword">const response = <spanclass="hljs-keyword">await api.topTen(); //this calls axios(url)\n <spanclass="hljs-keyword">const responseData = <spanclass="hljs-keyword">await response.<span83class="hljs-title function_">json();\n <spanclass="hljs-title function_">setLoading(<spanclass="hljs-literal">false);\n setTopten(responseData.<spanclass="hljs-property">data); \n setError(responseData.error); \n };\n\n fetchdata (); \n}, []);\n</code></pre>\n ’

After HTML Tags Removal
You have to convert the response to json Please Look at this link with await await response.json();\nand then use setState.\n\n useEffect(() => { \n console.log("useEffect TopTen has been called!"); \n const fetchdata = async () => {\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 ();

The subsequent step entails identifying technology-related stop words to facilitate the removal of these words from the data. Stop words are commonly occurring words that do not hold significant importance for the analysis and are frequently used in the text.

Traditionally, the process of finding stop words has been a laborious task involving the utilization of manually curated stop word lists [17]. These lists can be sourced from various origins, such as domain-specific or language-specific stop word lists. However, in the case of our study, which focuses on the software engineering domain, conventional stop word lists need to be revised as they need more specificity to this domain. Consequently, employing such general stop word lists could introduce noise into the dataset.

To address this challenge, [18] has significantly contributed by curating a stop word list explicitly tailored to the software engineering domain. Their approach involved thorough analysis of data extracted from patent documents, which predominantly describe domains related to software engineering. Notably, they employed a range of techniques, including preprocessing methods, a ranking framework based on term statistics, and an evaluation conducted by domain experts on a term-by-term basis. The meticulously curated stopword list developed by [18] is utilized in this study, ensuring its relevance and suitability for our specific research objectives.

It is essential to acknowledge that alternative methods exist for deriving stop word lists, such as employing word clouds to identify the most frequent words in the data and subsequently removing them. However, such approaches may need more precision as they do not consider the contextual nuances of the data or the significance of individual words within the dataset. Therefore, the comprehensive approach proposed by [18] emerges as a more effective solution, incorporating domain-specific considerations and expert evaluation to curate the appropriate stopword list for the software engineering domain. Table IV shows a sample of the stop words list curated by the paper.

TABLE IV
A SAMPLE OF STOP WORDS LIST CURATED BY [13]

able	above-	mentioned	accordingly	across
along	already	alternatively	always	among
and/or	anything	anywhere	better	disclosure

In the field of NLP, text normalization techniques such as lemmatization are employed to prepare sentences, words, and documents for analysis. These techniques aim to reduce words to their root or base form. For example, the terms "kick" and "kicked" both stem from the verb "to kick," and it is desirable for a NLP application to recognize this relationship.

3) Inferencing

To improve the effectiveness of information retrieval systems, we propose a hybrid scoring mechanism that incorporates TF-IDF (1), time relevance (2), and vote count (3). The goal is to rank titles based on their relevance to a given query while considering both textual similarity and temporal proximity.

The mechanism begins by utilizing the TF-IDF method, a statistical measure that evaluates how relevant a word is to a document in a collection of documents. In addition to the TF-IDF score, the mechanism incorporates the time relevance of each title. The created date of each title is compared to the current time, and a time difference score is calculated. This score represents the temporal proximity of the title to the present moment. The closer the created date is to the current time, the higher the time score assigned to the title.

The formula for TF-IDF Score is as follows:

$$w_{ij} = tf_{ij} \times \log(N/df_i) \quad (1)$$

where
tf_{ij} : number of occurrences of i in
df_i : number of documents containing i
N = total number of documents

The formula for Time Score is as follows:

$$\text{Time Score} = 100 - (r-n/t) * 100 \quad (2)$$

where
r: current time
n: created date
t: max time difference

Furthermore, the mechanism considers the vote count of each title as an indicator of its popularity or relevance. The vote count is transformed into a vote count score, considering the minimum and maximum vote counts in the dataset. The score is calculated as a percentage of the vote count's position within the vote count range, ensuring that higher vote counts receive higher scores.

The formula for Vote Count Score is as follows:

$$\text{Vote Count Score} = 100 - (n - s / y - s) \times 100 \quad (3)$$

where n is created date, y is vote count, and s is min vote count.

To achieve a balanced ranking, weightages are assigned to each score component. In our approach, the TF-IDF matching score carries a weightage of 80%, reflecting its primary importance in capturing textual similarity. The time and vote count scores contribute with weightages of 10% each, acknowledging their relevance but to a lesser degree than textual similarity. It is observable based on our experimental session, where the ratio of 80:10:10 is the best setting we've settled in.

The formula to Calculate the Weighted Score is as follows:

$$\text{Weighted Score} = (w*s) + (x*\text{Time Score}) + (z*\text{Vote Count Score}) \quad (4)$$

where s is min vote count, w is TF-IDF weightage, x is time weightage, and z is vote weightage.

The final weighted score for each title is obtained by combining the TD-IDF matching score, time score, and vote count score according to their respective weightages. The mechanism sorts the titles based on the weighted scores in descending order, ensuring that titles with higher overall scores are ranked higher in the retrieval results.

Sentiment analysis, also known as opinion mining, is a crucial process in determining the sentiment or attitude expressed in a piece of writing, whether positive, negative, or neutral. In the context of our framework, sentiment analysis plays a pivotal role in enhancing the ranking of posts. Upon scoring the posts using the sentence scoring method and assessing their relevance, it becomes apparent that the scores alone may not offer sufficient information to judge the usefulness of a post. To overcome this limitation, sentiment

analysis is introduced as an additional factor in the ranking process.

By conducting sentiment analysis, the assumption is made that posts with a positive sentiment are more likely to be useful than those with a negative sentiment. This assumption allows for incorporating sentiment analysis as an additional layer of filtering within the ranking aspect of the framework. Initially, the Twitter RoBERTa base sentiment analysis model is employed, given its popularity with over 2 million applications this month. This model is based on RoBERTa, a widely utilized transformer-based model, and has been trained on an extensive dataset encompassing approximately 124 million tweets from January 2018 to December 2021. However, subsequent research revealed a superior alternative.

In the latest study, the Senti4SD [27] model was utilized, surpassing the performance of the Twitter RoBERTa base sentiment analysis model significantly. Senti4SD is an emotion polarity classifier for sentiment analysis in developers' communication channels. This model was trained and evaluated using a gold standard dataset comprising over 4,000 posts extracted from Stack Overflow. It is a Collab Emotion Mining Toolkit (EMTk) component, catering to sentiment analysis requirements in software development contexts. The Senti4SD model demonstrates remarkable performance by accurately predicting sentiment and providing probability scores for three sentiment classes: positive, negative, and neutral. Leveraging the capabilities of this model allows for the determination of sentiment expressed in each text, with corresponding probability scores assigned to each sentiment class. Training on a domain-specific dataset from Stack Overflow enhances its effectiveness in capturing sentiments prevalent in developers' communication channels.

Incorporating the Senti4SD model into the framework elevates the accuracy and reliability of the sentiment analysis process. A comprehensive understanding of the posts' usefulness and relevance is obtained by considering the sentiment expressed in each post alongside the sentence scores. This refined approach enables a more precise ranking of the posts within the framework, facilitating improved decision-making based on sentiment analysis.

As the framework reaches the final stage of the pipeline, it aims to enhance the user experience by providing a summary of the top 5 ranked posts. Summarization involves condensing a text document to create a concise summary that captures the critical points of the original document. The goal of summarization is to reduce the length of the text while preserving the most essential information. In our framework, summarization is critical as it enables users to comprehend the posts more efficiently, quickly, and effortlessly. By generating summaries of the top 5 sentiment-ranked posts, we aim to facilitate a better understanding of the content and enable users to grasp the essential information more easily.

The chosen model for summarization is the widely used Bidirectional and Autoregressive Transformer (BART) large CNN model [28], which has been utilized more than 1 million times this month. BART is a transformer-based encoder-decoder model that combines bidirectional encoding (similar to BERT) and autoregressive decoding (similar to GPT). It has been pretrained on English language data and fine-tuned on the CNN Daily Mail dataset. To enhance the performance of the BART model for our specific use case, we have conducted fine-tuning using the SOSum dataset [29]. This dataset consists of extractive summaries from 2,278 Stack Overflow posts related to 506 of the most popular Stack Overflow questions. This fine-tuning aims to tailor the BART model to understand better and generate summaries specifically for Stack Overflow posts. In the final section of this paper, we will present and discuss the results of the fine-tuned BART model and compare its performance to the base model. This analysis will demonstrate how the fine-tuned model outperforms the original model, highlighting its effectiveness in generating more accurate and informative summaries.

III. Results and Discussion

We conduct both quantitative experiments and user studies. This section will describe the outcome of our experiments and user studies.

A. Datasets Descriptions

It is important to note that all the selected datasets are closely related to software engineering domains or at least have the exact nature of technicality in online forums. The dataset used to evaluate the **question-matching methods** is the Quora questions pair dataset because it provides pairs of questions, i.e., it contains two questions with the same meaning. The ground truth is the set of labels supplied by human experts. To evaluate the **Sentiment Analysis model**, the dataset chosen was the gold standard Stackoverflow dataset [30]; the data are directly scraped from Stack Overflow, thus making perfect sense to our research work. Lastly, the **summarization evaluation** datasets are from SOSum [29]. This dataset consists of extractive summaries from 2,278 Stack Overflow posts related to 506 of the most popular Stack Overflow questions.

B. Evaluation

1) Question matching models:

Four different methods are tested: fuzzy wuzzy partial ratio, fuzzy wuzzy token sort, spacy similarity, and the roberta-large-mnli model. The four methods are evaluated using the Quora questions pair dataset, confusion matrices, F1, recall, and precision scores, which are noted down to perform further analysis as in Table V.

TABLE V
QUESTION MATCHING MODELS EVALUATION METRICS

Models	Precision	Recall	F1
Fuzzy wuzzy partial ratio	30.03%	52.61%	38.23%
Fuzzy wuzzy token sort	29.10%	56.12%	38.33%
Spacy Similarity	**96.35%**	40.24%	56.77%
roberta-large-mnli-entailment	76.70%	**66.63%**	**71.31%**

The roberta-large-mnli-entailment model shows us that including entailment is crucial in understanding semantic meaning of the sentences when comparing the similarity between two sentences, therefore this model is chosen as our framework.

2) Sentiment analysis models:

The RoBERTa model and Senti4SD model are both tested. Similarly, confusion matrices, F1, recall, and precision scores are detailed in Table VI. Senti4SD is the better choice for our

use case, as the model is trained explicitly on Stackoverflow's data.

TABLE VI
SENTIMENT ANALYSIS MODELS EVALUATION METRICS

Models	Precision	Recall	F1
Roberta	**68%**	69%	68%
Senti4SD	60%	**100%**	**84%**

3) Summarization models

As part of our research work, this paper fine-tunes a base BART model with the SOSum [29] datasets to improve its accuracy further. Both models are evaluated with the SOSum model. It is important to note that the datasets used to fine-tune the BART model are split into training and evaluation datasets, whereby the training data is fed into the model while fine-tuning, and the evaluation datasets are treated as unseen data.

Three evaluation methods in ROUGE are used in our evaluation process. Rouge-1 evaluates individual words, Rouge-2 assesses word pairs, and Rouge-L considers overall structure and content overlap. Using a combination of metrics provides a comprehensive evaluation. The suitability of each metric varies based on specific evaluation needs. Rouge-1 is helpful for keyword accuracy, Rouge-2 focuses on cohesion and fluency, and Rouge-L allows word order and sentence structure flexibility. By considering multiple metrics, a broader range of summary qualities can be assessed, offering a more holistic view of system performance. Ultimately, the choice should align with the goals and requirements of the summarization task. Approaching a near 60% accuracy in abstractive summarization is reasonable based on state-of-the-art evaluated models [31]. Table VII(A-C) shows the performance results of respective ROUGE.

TABLE VII (A)
SUMMARIZATION MODELS EVALUATION METRICS - ROUGE-1

Models	Precision	Recall	F1
Base BART	22.40%	**83.24%**	34.43%
Fine-tuned BART	**46.01%**	75.36%	**53.47%**

TABLE VII (B)
SUMMARIZATION MODELS EVALUATION METRICS - ROUGE-2

Models	Precision	Recall	F1
Base BART	14.70	**70.95%**	23.69%
Fine-tuned BART	**40.32%**	69.85%	**47.12%**

TABLE VII (C)
SUMMARIZATION MODELS EVALUATION METRICS - ROUGE-L

Models	Precision	Recall	F1
Base BART	21.88%	**81.41%**	33.64%
Fine-tuned BART	**45.72%**	74.96%	**53.16%**

4) Verdict by human evaluation

Apart from individual model evaluations, the paper combines all models and ultimately proposes a framework to help developers filter, analyze, and summarize online forums. 10 pairs of questions and answers denote generating 10 queries and feeding to the model to infer the best Stack Overflow posts. The experiment is set up whereby the original post and the summarized content are presented to the testers. 10 undergraduate students with a computer science background are then picked from our institution to perform human evaluations. A question sheet will then be presented to each tester alongside 10 questions and answers generated by the proposed framework. The question sheet contains 3 checkboxes: relevancy, preferred ranking, and preferred summarization. Relevancy indicates if the answers are relevant to the query, and preferred ranking indicates if the sorting of the questions and answer pairs is correct. Lastly, preferred summarization indicates if the summarization outputs are beneficial in understanding the content of the answers. The results are shown in Table VIII.

TABLE VIII
HUMAN EVALUATION METRICS ON THE OVERALL PROPOSED FRAMEWORK

Question	Relevancy	Preferred Ranking	Preferred Summarization
1	100%	60%	50%
2	100%	70%	10%
3	100%	60%	40%
4	100%	80%	10%
5	100%	50%	30%
6	100%	50%	50%
7	100%	40%	50%
8	100%	80%	20%
9	100%	50%	40%
10	100%	60%	50%
Averaged:	**100%**	**60%**	**35%**

IV. CONCLUSION

This paper presents an Automated Frequently Asked Question Generation and Retrieval framework specifically tailored for the software engineering domain. The framework addresses the challenges software engineers face when seeking solutions on open forums like Stack Overflow. The proposed framework achieves promising results by integrating state-of-the-art models from domains such as string matching, sentiment analysis, and summarization, with F1 scores of 71.31%, 74.90%, and 53.4%, respectively. A user study involving 10 participants was conducted to evaluate the framework, with assessments on relevancy, preferred ranking, and preferred summarization. The results indicate high relevance scores (100%) while ranking and summarization obtained average scores of 60% and 35%, respectively. Thorough evaluations have been conducted on each individual component, confirming their viability within our framework. The fine-tuned BART model has demonstrated superior performance compared to the baseline model, achieving a higher score. However, integrating these components into a combined framework has further enhanced the overall results. Future work includes improving summarization models by incorporating text classification and summarizing them individually [32], as well as proposing feedback loop systems based on human reinforcement learning. Furthermore, efforts will be made to optimize the framework by utilizing knowledge graphs for dimension reduction, enabling it to handle larger corpora effectively.

REFERENCES

[1] S. Gupta and V. R. Carvalho, "FAQ Retrieval Using Attentive Matching," Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval, Jul. 2019, doi: 10.1145/3331184.3331294.

[2] F. Raazaghi, "Auto-FAQ-Gen: Automatic Frequently Asked Questions Generation," Lecture Notes in Computer Science, pp. 334–337, 2015, doi: 10.1007/978-3-319-18356-5_30.

[3] W.-C. Hu, D.-F. Yu, and H. C. Jiau, "A FAQ Finding Process in Open Source Project Forums," 2010 Fifth International Conference on Software Engineering Advances, Aug. 2010, doi:10.1109/icsea.2010.46.

[4] S. Hens, M. Monperrus, and M. Mezini, "Semi-automatically extracting FAQs to improve accessibility of software development knowledge," 2012 34th International Conference on Software Engineering (ICSE), Jun. 2012, doi: 10.1109/icse.2012.6227139.

[5] F. Razzaghi, H. Minaee, and A. A. Ghorbani, "Context Free Frequently Asked Questions Detection Using Machine Learning Techniques," 2016 IEEE/WIC/ACM International Conference on Web Intelligence (WI), Oct. 2016, doi: 10.1109/wi.2016.0095.

[6] A. Virani, R. Yadav, P. Sonawane, and S. Jawale, "Automatic Question Answer Generation using T5 and NLP," 2023 International Conference on Sustainable Computing and Smart Systems (ICSCSS), Jun. 2023, doi: 10.1109/icscss57650.2023.10169726.

[7] S. Gangopadhyay and S. M. Ravikiran, "Focused Questions and Answer Generation by Key Content Selection," 2020 IEEE Sixth International Conference on Multimedia Big Data (BigMM), Sep. 2020, doi: 10.1109/bigmm50055.2020.00017.

[8] S. Dutta, H. Assem, and E. Burgin, "Sequence-to-sequence learning on keywords for efficient FAQ retrieval," *arXiv preprint arXiv:2108.10019*, 2021, doi: 10.48550/arXiv.2108.10019.

[9] V. Jijkoun and M. de Rijke, "Retrieving answers from frequently asked questions pages on the web," Proceedings of the 14th ACM international conference on Information and knowledge management, Oct. 2005, doi: 10.1145/1099554.1099571.

[10] T. Makino, T. Noro, and T. Iwakura, "An FAQ Search Method Using a Document Classifier Trained with Automatically Generated Training Data," Lecture Notes in Computer Science, pp. 295–305, 2016, doi:10.1007/978-3-319-42911-3_25.

[11] S. Vasisht, V. Tirthani, A. Eppa, P. Koujalgi, and R. Srinath, "Automatic FAQ Generation Using Text-to-Text Transformer Model," 2022 3rd International Conference for Emerging Technology (INCET), May 2022, doi: 10.1109/incet54531.2022.9823967.

[12] G. Kothari, S. Negi, T. A. Faruquie, V. T. Chakaravarthy, and L. V. Subramaniam, "SMS based interface for FAQ retrieval," in *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*, 2009, pp. 852–860.

[13] S. Zhang, Y. Hu, and G. Bian, "Research on string similarity algorithm based on Levenshtein Distance," 2017 IEEE 2nd Advanced Information Technology, Electronic and Automation Control Conference (IAEAC), Mar. 2017, doi: 10.1109/iaeac.2017.8054419.

[14] G. Zhou, Y. Liu, F. Liu, D. Zeng, and J. Zhao, "Improving question retrieval in community question answering using world knowledge," in *Twenty-third international joint conference on artificial intelligence*, 2013.

[15] M. Gerlach, H. Shi, and L. A. N. Amaral, "A universal information theoretic approach to the identification of stopwords," Nature Machine Intelligence, vol. 1, no. 12, pp. 606–612, Dec. 2019, doi:10.1038/s42256-019-0112-6.

[16] S. Sarica and J. Luo, "Stopwords in technical language processing," PLOS ONE, vol. 16, no. 8, p. e0254937, Aug. 2021, doi:10.1371/journal.pone.0254937.

[17] C. P. Chai, "Comparison of text preprocessing methods," Natural Language Engineering, vol. 29, no. 3, pp. 509–553, Jun. 2022, doi:10.1017/s1351324922000213.

[18] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," *Adv Neural Inf Process Syst*, vol. 26, 2013.

[19] Y. Wang, J. Guo, C. Yuan, and B. Li, "Sentiment analysis of Twitter data," *Applied Sciences*, vol. 12, no. 22, p. 11775, 2022, doi:10.3390/app122211775.

[20] Y.-C. Fung, L.-K. Lee, K. T. Chui, G. H.-K. Cheung, C.-H. Tang, and S.-M. Wong, "Sentiment Analysis and Summarization of Facebook Posts on News Media," Advances in Data Mining and Database Management, pp. 142–154, 2022, doi: 10.4018/978-1-7998-8413-2.ch006.

[21] C.-P. Chan and J.-H. Yang, "Instagram Text Sentiment Analysis Combining Machine Learning and NLP," ACM Transactions on Asian and Low-Resource Language Information Processing, Jul. 2023, doi:10.1145/3606370.

[22] O. Alqaryouti, N. Siyam, A. Abdel Monem, and K. Shaalan, "Aspect-based sentiment analysis using smart government review data," Applied Computing and Informatics, vol. 20, no. 1/2, pp. 142–161, Jul. 2020, doi: 10.1016/j.aci.2019.11.003.

[23] D. Yadav, J. Desai, and A. K. Yadav, "Automatic text summarization methods: A comprehensive review," *arXiv preprint arXiv:2204.01849*, 2022, doi: 10.48550/arXiv.2204.01849.

[24] L. Banarescu *et al.*, "Abstract meaning representation for sembanking," in *Proceedings of the 7th linguistic annotation workshop and interoperability with discourse*, 2013, pp. 178–186.

[25] F. Calefato, F. Lanubile, F. Maiorano, and N. Novielli, "Sentiment polarity detection for software development," Proceedings of the 40th International Conference on Software Engineering, May 2018, doi:10.1145/3180155.3182519.

[26] M. Lewis *et al.*, "Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension," *arXiv preprint arXiv:1910.13461*, 2019, doi:10.48550/arXiv.1910.13461.

[27] B. Kou, Y. Di, M. Chen, and T. Zhang, "SOSum," Proceedings of the 19th International Conference on Mining Software Repositories, May 2022, doi: 10.1145/3524842.3528487.

[28] N. Novielli, F. Calefato, and F. Lanubile, "A gold standard for emotion annotation in stack overflow," Proceedings of the 15th International Conference on Mining Software Repositories, May 2018, doi:10.1145/3196398.3196453.

[29] A. A. Syed, F. L. Gaol, and T. Matsuo, "A Survey of the State-of-the-Art Models in Neural Abstractive Text Summarization," IEEE Access, vol. 9, pp. 13248–13265, 2021, doi: 10.1109/access.2021.3052783.

[30] B. Kou, M. Chen, and T. Zhang, "Automated Summarization of Stack Overflow Posts," *arXiv preprint arXiv:2305.16680*, 2023, doi:10.48550/arXiv.2305.16680.