Answer Summarization for Technical Queries: Benchmark and New Approach

Chengran Yang[†], Bowen Xu[†*], Ferdian Thung[†], Yucen Shi[†], Ting Zhang[†], Zhou Yang[†], Xin Zhou[†], Jieke Shi[†], Junda He[†], DongGyun Han[‡], David Lo[†]
[†]School of Computing and Information Systems, Singapore Management University
{cryang,bowenxu.2017,ferdianthung,ycshi,tingzhang.2019,zyang,xinzhou.2020,jiekeshi,jundahe,davidlo}@smu.edu.sg
[‡]Royal Holloway, University of London
donggyun.han@rhul.ac.uk

ABSTRACT

Prior studies have demonstrated that approaches to generate an answer summary for a given technical query in Software Question and Answer (SQA) sites are desired. We find that existing approaches are assessed solely through user studies. Hence, a new user study needs to be performed every time a new approach is introduced; this is time-consuming, slows down the development of the new approach, and results from different user studies may not be comparable to each other. There is a need for a benchmark with ground truth summaries as a complement assessment through user studies. Unfortunately, such a benchmark is non-existent for answer summarization for technical queries from SQA sites.

To fill the gap, we manually construct a high-quality benchmark to enable automatic evaluation of answer summarization for the technical queries for SQA sites. It contains 111 query-summary pairs extracted from 382 Stack Overflow answers with 2,014 sentence candidates. Using the benchmark, we comprehensively evaluate the performance of existing approaches and find that there is still a big room for improvements.

Motivated by the results, we propose a new approach TechSumBot with three key modules:1) *Usefulness Ranking* module; 2) *Centrality Estimation* module; and 3) *Redundancy Removal* module. We evaluate TechSumBot in both automatic (i.e., using our benchmark) and manual (i.e., via a user study) manners. The results from both evaluations consistently demonstrate that TechSumBot outperforms the best performing baseline approaches from both SE and NLP domains by a large margin, i.e., 10.83%–14.90%, 32.75%–36.59%, and 12.61%–17.54%, in terms of ROUGE-1, ROUGE-2, and ROUGE-L on automatic evaluation, and 5.79%–9.23% and 17.03%–17.68%, in terms of average usefulness and diversity score on human evaluation. This highlights that automatic evaluation on our benchmark can uncover findings similar to the ones found through user studies. More importantly, the automatic evaluation has a much lower cost, especially when it is used to assess a new approach. Additionally, we also conducted an ablation study, which demonstrates that each module in TechSumBot contributes to boosting the overall performance of TechSumBot. We release the benchmark as well as the replication package of our experiment at https://github.com/TechSumBot/TechSumBot.

CCS CONCEPTS

• **Software and its engineering** → **Software libraries and repositories**; • **Computing methodologies** → **Artificial intelligence**.

KEYWORDS

Summarization, Question Retrieval, Pre-Trained Models

ACM Reference Format:
Chengran Yang[†], Bowen Xu[†*], Ferdian Thung[†], Yucen Shi[†], Ting Zhang[†], Zhou Yang[†], Xin Zhou[†], Jieke Shi[†], Junda He[†], DongGyun Han[‡], David Lo[†]. 2022. Answer Summarization for Technical Queries: Benchmark and New Approach. In *37th IEEE/ACM International Conference on Automated Software Engineering (ASE '22), October 10–14, 2022, Rochester, MI, USA.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3551349.3560421

1 INTRODUCTION

Online software question and answer (SQA) forums are becoming an integral part of software engineering. As the Stack Overflow community flourishes, the platform has around 23 million existing answers as of May 2022, making it a valuable software engineering resource.[1] Developers often rely on Stack Overflow to acquire software knowledge, such as learning API usage, fixing bugs, and discovering trends in development technologies [24, 28, 36, 38, 41, 43] by searching from a massive collection of existing answers. Unfortunately, multiple works (e.g., [21, 41]) have demonstrated that developers suffer from the ineffective and time-consuming answer searching process for their technical queries due to useless, redundant, and incomplete information returned by existing search engines.

Several studies have been proposed to help developers capture desired online information more accurately and succinctly by proposing query-focused summarization approaches [28, 36, 37, 39, 41] – this task is often referred to as *answer summarization for technical questions*. The closest work is proposed by Xu et al. [41], who proposed the latest Stack Overflow answer summarization approach named AnswerBot for technical queries. The problem is formulated as a query-focused extractive summarization problem, i.e., selecting

∗ Corresponding author

ASE '22, October 10–14, 2022, Rochester, MI, USA

ACM ISBN 978-1-4503-9475-8/22/10…$15.00
https://doi.org/10.1145/3551349.3560421

[1] https://stackexchange.com/sites?view=list#traffic

an optimal subset of k sentences from multiple answers to form a summary to answer the target technical query.

However, we find that the work carries a major limitation on the evaluation. Both AnswerBot and its considered baseline approaches are evaluated only in a manual manner through user studies. Although user studies can simulate the realistic usage environment of the approaches, it carries two drawbacks at the same time. First, user studies are usually expensive in terms of time cost and human resources. It requires a long period of time for preparation, labeling, and human feedback collection. Second, the user study needs to be repeated every time a new approach is introduced. Aside from slowing down the development of new approaches, it also raises fairness issues. To ensure a fair comparison, a user study on all the considered approaches requires to be done under a similar evaluation setting. However, this fairness constraint is hard to enforce for a series of user studies that evaluate different newly proposed approaches. Such user studies are often performed with different groups of participants and different sets of technical questions.

To mitigate the aforementioned limitations of user studies, automatic evaluation through a benchmark is in need as it can be a good complement. After the initial investment to set up a benchmark, future studies can reuse it to evaluate new approaches in an identical evaluation setting to produce easily repeatable results. For the answer summarization for technical questions task, a benchmark ideally consists of a set of triplets in which each triplet contains a technical query Q, a set of answers Ans, and ground truth answer summaries S with respect to the query, i.e., $Bench = \{\langle Q, S, Ans \rangle\}$. Such a benchmark is often constructed for various summarization tasks in the Natural Language Processing (NLP) field, e.g., CNN/Daily Mail[30], GigaWord [31] and DUC–2005[7]. However, no such benchmark exists for our task considering SQA sites.

Motivated by this, we conduct the first Stack Overflow query-focused multi-answer summarization benchmark TechSumBench (described in Section 2). Table 1 demonstrates an example pair of a technical query and its corresponding answer summary.

Table 1: Example answer summary in TechSumBench with respect to a technical query.

Technical Query: Difference between Spring MVC and Spring Boot
Ground Truth Summary in TechSumBench: 1. ⟨ Spring MVC is a sub-project of the Spring Framework, targeting design and development of applications that use the MVC (Model-View-Controller) pattern.⟩ 2. ⟨ Spring boot is a utility for setting up applications quickly, offering an out of the box configuration in order to build Spring-powered applications.⟩ 3. ⟨ Spring boot = Spring MVC + Auto Configuration(Don't need to write spring.xml file for configurations) + Server(You can have embedded Tomcat, Netty, Jetty server).⟩ 4. ⟨ Spring MVC framework is module of spring which provide facility HTTP oriented web application development.⟩ 5. ⟨ So, Spring MVC is a framework to be used in web applications and Spring Boot is a Spring based production-ready project initializer.⟩

TechSumBench enables us to fairly compare query-focused summarization approaches proposed in the software engineering and NLP fields in an identical evaluation setting and in a repeatable manner. We have used TechSumBench to investigate four approaches (i.e., AnswerBot [41], LexRank [9], Biased-TextRank [17], QuerySum [42]) and find that QuerySum performs the best, but there is still a big room for improvement. Furthermore, we performed a qualitative analysis to seek the reason for the poor performance of these approaches (described in Section 6.1) and summarized two key findings. First, the handcrafted features considered in the existing approaches have limitations. For example, one handcrafted feature in AnswerBot (e.g., *semantic patterns*) uses 12 patterns to indicate the importance of answer sentences, e.g., check if a sentence contains the string "I suggest that...". Such a semantic pattern is not versatile enough to capture the semantics of questions and answer sentences. Second, when removing redundancy from a set of sentences, we find that sentence representation plays an important role in determining the semantic relatedness between sentences.

To better tackle the problem, we propose TechSumBot (described in Section 3), a new query-focused answer summarization approach with three core modules: 1) Usefulness Ranking; 2) Centrality Estimation; 3) Redundancy Removal. Correspondingly, they take into consideration 1) the usefulness of each answer sentence with respect to the query, 2) sentence centrality among all candidates, and 3) semantic similarity between sentences. In the Usefulness Ranking module, we rank the *usefulness* of each sentence to the query by leveraging a transfer learning-based pre-trained Transformer model. In the Centrality Estimation module, we re-rank the sentence by estimating the central sentences among all candidates. Finally, in the Redundancy Removal module, we remove redundant sentences by applying a greedy selection mechanism, in which we leverage a SE domain sentence representation model to calculate the similarity between sentences. Particularly, as no existing dataset in SE domain can be directly used for supervising the sentence representation model, we carefully observe the characteristic of Stack Overflow data and create an in-domain sentence relationship dataset by leveraging the semantics inferred from duplicate question links and tags information; both of them are manually labeled by Stack Overflow users in a high-standard mechanism. Then we use our in-domain sentence relationship dataset to enhance the state-of-the-art contrastive learning-based sentence representation model SIMCSE [12].

To facilitate a comprehensive evaluation of TechSumBot, we conduct an automatic evaluation using TechSumBench and a user study. The automatic evaluation results show that TechSumBot substantially outperforms the best performing baselines in both SE and NLP fields by 10.83%–14.90%, 32.75%–36.59%, and 12.61%–17.54% in terms of ROUGE-1, ROUGE-2, and ROUGE-L, respectively. The result of the user study is consistent with the result of the automatic evaluation: TechSumBot substantially outperforms the best performing baseline by 5.79%–9.23%, 17.03%–17.68% in terms of average usefulness and diversity scores. Besides, an ablation study is performed and its result demonstrates that each module complementarily contributes to the overall performance.

The main contributions of this paper are the following:

- We construct the first summarization benchmark named TechSumBench for answer summarization of technical queries in SQA sites. The benchmark can be used to automatically evaluate future proposed summarization methods in an identical setting and a repeatable manner.

- We comprehensively evaluate four query-focused summarization approaches from both SE and NLP fields using our benchmark. The experiments highlight the limitations of the approaches and indicate a room for improvement.
- We propose TechSumBot, a novel query-focused answer summarization approach with three modules to better solve the problem.
- We evaluate the performance of TechSumBot via both automatic evaluation and user study. The results of both evaluations demonstrate that TechSumBot outperforms the best performing baselines by a large margin.
- We release the benchmark, as well as the replication package of TechSumBot to facilitate future work.

The rest of this paper is structured as follows. Section 2 describes the benchmark construction. Section 3 presents the framework of our proposed TechSumBot. Section 4 introduces the baselines and metrics for automatic evaluation. Section 5 analyzes the experiment result. Section 6 discusses the qualitative analysis and threats to validity. Section 7 surveys the related work. Section 8 concludes our work and presents our future plan.

2 BENCHMARK

We present TechSumBench, the first Stack Overflow multi-answer summarization benchmark for answering a specific technical question. Figure 1 describes our benchmark construction process. We firstly automatically collect *annotation units*, which is then followed by the data cleaning process described in Section 2.1. An annotation unit consists of a technical question and multiple relevant answers. Next, six annotators perform a two-phase labeling process: useful sentence selection and summary generation as explained in Section 2.2. Note that we perform an iterative guideline refinement for the useful sentence selection task until the inter-annotator agreement achieves a certain level as described in Section 2.2.1.

2.1 Data Collection

We prepare 50 annotation units (i.e., technical query and multiple relevant answers) by performing automatic data extraction and data cleaning.

2.1.1 ***Automatic Data Preparation***. The PostLinks table in Stack Overflow data dump[2] contains the information of user-voted duplicated question links, in which each original question is linked to their duplicated questions. Specifically, an original question refers to the earliest published Stack Overflow question that is a duplicate of newer questions. The duplicate questions[3] are manually marked by following a strict mechanism that one moderator or at least three active users (i.e., users who have over 3,000 reputation score) vote the questions to be a duplicate to the original one. In particular, we treat the title of the original question as the technical question of each annotation unit in TechSumBench. As the Stack Overflow questions with 'duplicate' links are labeled by reputable Stack Overflow users, we regard all the answers to the original question and their duplicates as the relevant answers of the technical question in the corresponding annotation unit.

To prepare the labeling materials, we randomly extract 50 annotation units from the PostLinks table in Stack Overflow data dump. We focus on questions related to two popular programming languages, Python and Java, by checking if they are tagged with 'Java' or 'Python'. Following AnswerBot [41], we only consider the technical questions in which the sum of the number of answers is between 10 to 15. We discard answers with no vote as their usefulness to the technical question is unclear or considered as bad by Stack Overflow users. As we focus on the text summarization, answers that only contain code snippets without text content are also dropped.

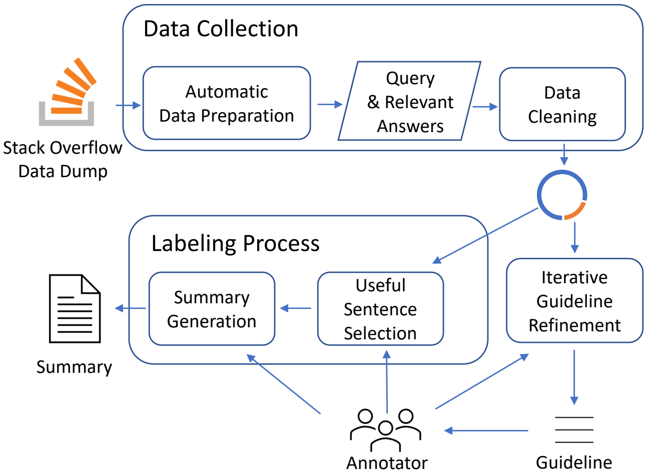

Figure 1: Benchmark Construction

2.1.2 ***Data Cleaning***. We pre-process the 50 annotation units. We use NLTK[4] to break each text content into sentences. We then apply some heuristics to reduce noise. We exclude sentences that consist of only an external hyperlink since the hyperlink content is not directly visible. For sentences that consist of both hyperlinks and text content, we keep the text content and replace the hyperlinks with a place holder (i.e., '[external-link]'). We also replace the code snippet, table and pictures in answers with place holders (i.e., '[code-snippet]', '[table]', '[figure]'). Additionally, we observe that the inline code is important for annotators to understand the sentence meanings. Therefore, we keep the inline code. Furthermore, we treat multi-level headings which is more than five words in answers as individual sentences as we observe that they are frequently summative sentences. Finally, each annotation unit after the above pre-processing serves as the labeling materials.

2.2 Labeling Process

Firstly we prepare labeling materials for annotators to get ready for the labeling process. We present the content of each annotation unit into a text file that contains the technical question, sentences from each answer, Stack Overflow hyperlinks to each answer, and a blank section for labeling. Each answer sentence is assigned with a unique identifier based on its source answer and the sentence order in the source answer. For example, the fourth sentence in the

[2] https://archive.org/details/stackexchange
[3] https://stackoverflow.com/help/duplicates
[4] https://www.nltk.org/api/nltk.tokenize.html

third answer of the annotation unit is assigned with the identifier #03_04. We also provide the guideline.

Then we follow a two-phase labeling process that is commonly used in NLP studies [1, 2]. Six annotators are involved in the labeling process. All participants are PhD students and have at least two years of development experience in Java and Python programming languages. In each phase, annotators and labeling materials are divided equally into two groups. Three participants in one group label the same group of labeling materials. The goal of the labeling process is to select summative sentences from multiple answers to form a brief summary of answers to a technical question. To achieve this, in the first phase (i.e., *useful sentence selection*), three annotators select the useful sentences from all relevant answers. The sentences that all annotators agreed to be useful for answering the technical question serve as the input of the second phase (i.e., *summary generation*). In the *summary generation* phase, three annotators are asked to select five sentences that will form an extractive summary based on the sentences' clarity, redundancy and importance. In prior summarization works from NLP field, it is common to select a fixed number of sentences [22, 32, 47] , e.g., five sentences [32, 47], as the summary.

2.2.1 *Phase 1: Useful Sentence Selection.*

In this labeling stage, annotators are required to identify the useful sentences for answering the technical question from multiple Stack Overflow answers. The labeling objective is formulated as a binary classification that specifies sentence usefulness to the technical question. Annotators label the sentences for answering the technical question as either 'useful' or 'not-useful'. Sentences are labeled as '?' when annotators are unsure whether the sentences are useful to answer the question. Note that we do not consider the redundancy among sentences in this phase. In particular, we use the Kappa coefficient [6] to measure the inter-annotator agreement. A higher kappa value illustrates a higher agreement level among annotators. We follow prior studies [16, 19, 26] to interpret the meaning of Kappa value: kappa values 0 as poor agreement, 0.01–0.20 as slight agreement, 0.21–0.40 as fair, 0.41– 0.60 as moderate, 0.61–0.80 as substantial, and 0.81–1.00 as almost perfect agreement.

Iterative Guideline Refinement To ensure the quality of the selected useful sentences, we performed an iterative guideline refinement to improve the inter-annotator agreement, which is widely adopted in the literature [5, 35]. Specifically, six annotators label the same annotation units in each iteration. In each round, we polish the guideline and update the labeling materials according to the annotator feedback and automatic evaluation result. We end the guideline refinement process when the inter-annotator agreement level is *Moderate*. The annotation units used in the guideline refinement process are discarded. Specifically, we discard 10 annotation units in the guideline refinement process and have 40 annotation units for full labeling.

Full Labeling Process Firstly we arrange meetings with the annotators to align the understanding of the guideline. We share the labeling rules that are obtained while observing the guideline refinement process with annotators. In particular, annotators should identify the usefulness of each sentence independently of its context (e.g., surrounding sentences). Additionally, general comments about the question itself or its solutions should be avoided (e.g., "I hate Java!"), as they do not provide useful information to answer the query. Finally, for incomplete sentences, annotators should not assume the usefulness of its hidden part (e.g., "the content in '[code snippet]' place holder").

After the labeling process is finished, one author deals with the labeling results. In each annotation unit, all the sentences that three annotators agree to be useful to that question, coupled with the technical query, serve as the input to the second phase. Note that, there are three annotation units whose inter-annotator agreement is *Slight* (i.e., the Kappa value k is $0.00 \leq k \geq 0.20$) in the full labeling process. We consider a good agreement to be at least *Moderate* (i.e., the Kappa value k is $0.40 \leq k \geq 0.60$). Therefore, we discard these annotation units as the labels may not be of good quality. We have 37 sets of ⟨ question, useful answers ⟩ pairs in total.

Inter-Annotator Agreement of Phase 1 The average kappa value of the *useful sentence selection* phase is 0.43, indicating the annotators achieve *Moderate* agreement. Previous work shows that the human agreements on sentence selection task for constructing news-domain summarization benchmarks are usually lower than 0.3 [33]. Additionally, the kappa value of a recent opinion summarization benchmark is 0.36 [1]. It indicates that the inter-annotators agreement on sentence selection task in our TechSumBench is comparable to those of common NLP community benchmarks.

2.2.2 *Phase 2: Summary Generation.*

In the second phase, given a set of useful sentences to answer the technical question, annotators are required to generate a final extractive summary under a budget of five sentences by selecting sentences based on three key factors, i.e.,clarity of each sentence, redundancy among sentences, and the importance for each sentence to answer the question.

To let annotators equip a consensus of the goal, we arrange a meeting with all the annotators for two things. First, we provide an example (excluded from the data used from constructing the benchmark) to demonstrate the materials and the brief overall labeling process. Second, we let them align their understanding based on the materials by discussing with each other.

After that, annotators select sentences by following the process: first, all sentences are clustered into different RCs (Redundant Cluster) in which each sentence is semantically similar; then the annotator should select the most summative sentence with high clarity from each RC into the candidate list; finally the annotators are asked to delete sentences from the candidate list until only five sentences are left, by taking into account how well each sentence answers the question. Note that, for each group of sentences in our benchmark, there are always more than five RCs clustered by annotators. Finally, the top five sentences serve as the summary of the answers to the technical question.

2.3 Benchmark Statistic

In TechSumBench, there are 37 technical questions, each of which corresponds to three ground truth answer summaries that are independently labeled by three annotators. Each summary contains five sentences extracted from corresponding relevant answers. In total, we have 111 query-summary pairs. The average number of words in ground-truth summaries is 106.45. The scale of TechSumBench is comparable with two commonly used query-focused multi-documentation summarization benchmarks in the NLP community [7, 12], i.e., DUC-2005 and DUC-2007, which consist of 50 and 45 annotation units, respectively.

3 APPROACH

This section presents our proposed approach TechSumBot to tackle the problem of answer summarization for technical queries.

3.1 Overview

As shown in Figure 2, TechSumBot takes a technical query and a set of relevant answers as input and outputs an extractive answer summary. The framework of TechSumBot contains three core modules: Usefulness Ranking, Centrality Estimation, and Redundancy Removal. Correspondingly, they tackle the problem from three different perspectives, 1) the degree of usefulness carried by an answer sentence to answer the query; 2) the centrality (importance) level carried by a sentence among answer sentences; 3) the extent of redundant information carried by two sentences, respectively.

TechSumBot firstly decomposes all the relevant answers into a list of sentences in the pre-processing step. In the first Usefulness Ranking module, we leverage a transfer learning-based approach to capture the *usefulness* of each sentence to the given query. Usefulness Ranking module produces a ranked list of answer sentences ordered by the predicted usefulness. Next, the second module, named Centrality Estimation, measures the centrality (i.e., the importance) of a sentence among the set of useful sentences by TextRank [27], a simple yet effective approach for the problem. Centrality Estimation module outputs a rank list of sentences in terms of sentence centrality. The third module Redundancy Removal aims to reduce redundant sentences among candidates. To achieve the goal, we propose a greedy algorithm-based selection mechanism, in which we integrate the state-of-the-art sentence representation model. The Redundancy Removal module also outputs a rank list of sentences without redundant sentences. In the end, the top-5 answer sentences are used to form an answer summary to the target technical query.

3.2 Module I: Usefulness Ranking

Measuring the usefulness of sentences with respect to a given query is a common step for the technical query-focused answer summarization. For example, AnswerBot [41] assesses the usefulness by using handcrafted features which carry certain limitations. Taking one of the features named *semantic patterns* as an example, it uses a static set with 12 manually summarized strings (e.g., "*I'd recommend...*") as an indicator of usefulness. However, such heuristic-based features require great human effort to maintain.

3.2.1 *Transfer Learning-based Ranking Approach*.

Inspired by a recent work [42], we leverage BERT, a transfer learning-based model [15] that has achieved great success in many Stack Overflow-based tasks [43, 45], to model the extent of the usefulness of each answer sentence with respect to the query. Specifically, we first train a BERT-based classifier that predicts the likelihood of an answer sentence's usefulness score with respect to the target query. And then we rank the usefulness of each answer sentence by extracting the intermediate score (i.e., the probability of positive class) predicted by the classifier. We leverage a large-scale QA dataset named ASNQ [13] (short for Answer Sentence Natural Questions) from the general domain to fine-tune the BERT-based classifier. ASNQ has around 19 million pairs of data, in which each data is in the form of two sets of query-sentence pairs, i.e., $D = \{\{\langle Q_i \rightarrow PS_j\rangle\}, \{\langle Q_i \rightarrow NS_k\rangle\}\}$. Q_i denotes each query in ASNQ data, which is the real anonymized query issued to the Google search engine. PS_j and NS_k denote corresponding useful and useless Wikipedia sentence for answering the query. Garg et al. [13] argued that the BERT model trained with ASQN dataset can produce promising performance on NLP domain sentence usefulness classification tasks. By doing so, our model can benefit from the existing large-scale dataset.

3.2.2 *Model Training*.

Firstly we train a BERT model for the sentence usefulness classification task by feeding the ASNQ dataset [13]. For each of the pairs $\langle Q, S\rangle$ in the ASNQ dataset, we convert it into a format acceptable by BERT, i.e., $\{[CLS], Q, [SEP], S, [SEP]\}$. Both $[CLS]$ and $[SEP]$ are special tokens used to form the input of BERT. $[CLS]$ stands for classification and $[SEP]$ stands for separator used to separate the Q and S. We follow the standard padding (i.e., adding $[PAD]$ token) to the sequence to fix the length of each input instance to 512. Next, the $[CLS]$ vectors extracted from encoder output serves as input to the final classification layer (i.e., a single neural network layer) and Sigmoid function to produce the likelihood of the input instance to be predicted as positive or negative class. By following the standard practice, we use cross entropy as the loss function:

$$\mathcal{L} = -\sum_{i=1}^{N} (y \log(p_{pos}) + (1-y) \log p_{neg}) \tag{1}$$

where N denotes the number of training instances; p_{pos} and p_{neg} denotes the probability of the positive and negative classes.

Once the BERT model is trained, for a given query Q and a set of relevant answer sentences $S = \{S_1, S_2, ..., S_i\}$, we utilize the model to predict the likelihood of each of the pairs $\langle Q, S_i\rangle$ as positive class (i.e., $P(class = useful|\langle Q, S\rangle)$) to the query and consider the probability p_{pos} as the usefulness score. At the end, the usefulness score is used to rank all candidate sentences. We select top k answer sentences with the highest usefulness scores as the input of the next module.

3.3 Module II: Centrality Estimation

To select summative sentences, we consider the usefulness of each sentence to the query and the centrality of each sentence among all candidates. To estimate the central sentences from all candidates, we apply a simple yet effective approach, TextRank [27], to extract the representativeness score of each sentence. TextRank is a graph-based sentence ranking approach that essentially aims to quantify the *representativeness* of each sentence in the sentence-graph based on global information recursively drawn from the entire graph. More precisely, a sentence similar to other sentences *recommends* others and thus represents the overall understanding of the complete documentation. Meanwhile, a highly *recommended* sentence by others is likely to be more summative and representative.

To perform TextRank algorithm, each candidate sentence is represented into a node in a sentence-graph with undirected edges. The initial weights of each edge are based on the word overlap between two nodes. Specifically, given two sentences S_i and S_j, each sentence is in form of N word tokens t, $S_i = t_1^i, t_2^i, ..., t_N^i$, the initial edge weight is defined as:

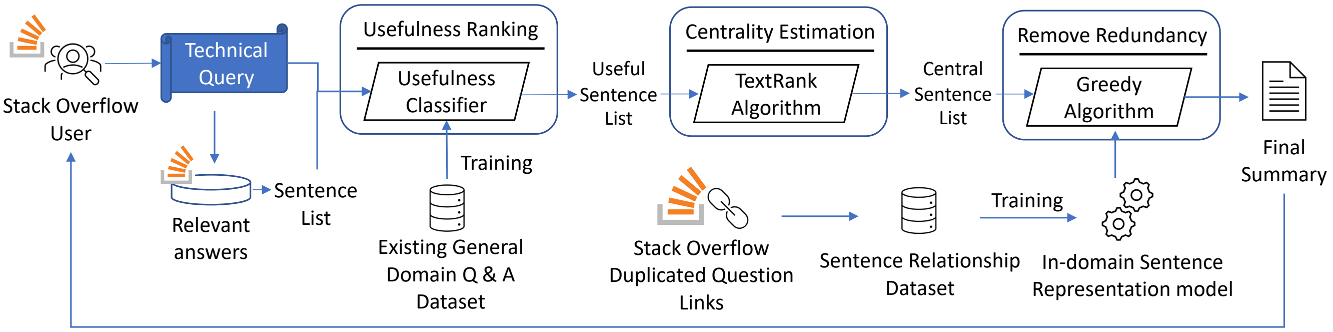

Figure 2: Overview of TechSumBot

$$Edge\_Weight(S_i, S_j) = \frac{\left|\{t_k | t_k \in S_i \& t_k \in S_j\}\right|}{log\left(|S_i|\right) + log\left(|S_j|\right)} \quad (2)$$

Then TextRank recursively calculates the representativeness score $R(s_i)$ of each sentence S_i as follow:

$$R(S_i) = 1 - \phi + \phi * \sum_{j \in In(S_i)} \frac{1}{Out(S_i)} R(S_j) \quad (3)$$

where $In(S_i)$ denotes the list of nodes that point to S_i while $Out(S_i)$ denotes the list of nodes that S_i points to. The $\phi \in (0, 1)$ serves as the dumping factor; it refers to the probability of jumping from a given node to another random node in the graph (i.e., simulate the user behavior of web surfing in PageRank). We set the heuristic value ϕ as 0.85 which is the same as the original paper [27]. We randomly set the initial value of $R(s_i)$ as the final value is not affected by the initial value $R(s_i)$. TextRank stops the iterations when the convergence score C is below a given threshold. Given the iteration k, the convergence score C is defined as follows:

$$C = R^{k+1}(s_i) - R^k(s_i) \quad (4)$$

As Mihalcea and Tarau [27] suggested, we set the threshold of C as 0.0001. Finally, we rank all input sentences s_i by taking the final representativeness score $R(s_i)$ as the ranking score. We feed the ranked sentence list into the final module.

3.4 Module III: Redundancy Removal

According to the survey conducted by Xu et al. [41], redundancy is one of the key factors that developers face in identifying their target information in software questions and answer sites. Thus, redundancy removal should be performed as a part of the answer summarization solution. To remove redundant sentences, an in-domain sentence representation model is desired to measure the similarity between two answer sentences. To achieve this, we create a large-scale SE domain sentence relationship dataset for training the sentence representation model. Each instance in this dataset is automatically extracted from Stack Overflow data by considering the implied semantics in the duplicate question links. We then use our collected dataset to train a contrastive learning-based sentence representation model and transform each sentence into the corresponding representation. Finally, we embed the learned representation into a greedy search algorithm to select sentences iteratively.

3.4.1 ***Dataset Construction for Contrastive Learning.*** To capture the semantic similarity between answer sentences, we train a Stack Overflow-specific sentence representation model based on contrastive learning. The main goal of contrastive representation learning is to learn such an embedding space in which similar sentence pairs stay close to each other while dissimilar ones are far apart [12]. Each input instance for training the approach is in the form of a triplet, $\langle s, s^+, s^- \rangle$, while $\langle s, s^+ \rangle$ are considered as similar pairs and $\langle s, s^- \rangle$ are dissimilar ones.

Unfortunately, none of the existing datasets in SE domain directly serves the purpose. To tackle the problem, we first carefully observe the characteristic of Stack Overflow data and find that the duplicate question pairs (as described in Section 2.1.1) in Stack Overflow naturally carry the semantic relatedness between questions (i.e., describe the same semantic information but in different ways) by its definition.[5] Duplicate questions could be a valuable signal considered as similar sentence pairs to supervise sentence representation, i.e., the representation of duplicate questions is expected to be close in the vector space while the non-duplicate pairs are far from each other. Thus, we propose to create positive (i.e., similar) sentence pairs in the triplet by leveraging duplicate question pairs in Stack Overflow. Specifically, we extract the titles of a pair of duplicate questions as the pair of the original sentence and its similar sentence, i.e., $\langle s, s^+ \rangle$. Besides, we also observe that if two questions are labeled by users with totally different tags, then they are most likely irrelevant. Based on the observation, we randomly select pairs of questions that share no common tag and use their title to form the negative (i.e., dissimilar) sentence pairs (i.e., $\langle s, s^- \rangle$). By following the TechSumBench setting, we focus on Stack Overflow duplicate posts with 'Java' and 'Python' tags. As a result, we built a dataset with 304,046 sentence triplets $\langle s, s^+, s^- \rangle$.

3.4.2 ***Contrastive Learning for Sentence Embedding.*** We apply SIMCSE [12], the state-of-the-art sentence embedding approach based on contrastive learning structure. SIMCSE uses a pre-trained model, RoBERTa [23], as its base model for generating embeddings and add a multilayer perceptron (i.e., MLP) layer on the top of it.

In the training stage of SIMCSE, for each input instance in the form of a triplet $\langle s, s_i^+, s_i^- \rangle$ described in Section 3.4.1, the loss function is defined as:

$$\mathcal{L} = -log \frac{e^{sim(r_i, r_i^+)/\tau}}{\sum_{j=1}^{N}(e^{sim(r_i, r_j^+)/\tau} + e^{sim(r_i, r_j^-)/\tau})} \quad (5)$$

[5] https://stackoverflow.com/help/duplicates

while r_i denotes the representation of each input s_i, N as the number of sentences in a mini-batch, and τ is a temperature hyperparameter, $sim(r_1, r_2)$ is the cosine similarity, respectively. Specifically, all parameters in base model RoBERTa [23] are fine-tuned by minizing the loss function (i.e., Eq. 5). We train the model by leveraging our in-domain dataset described in Section 3.4.1.

3.4.3 *Greedy Algorithm for Redundancy Removal.*

The goal of this step is to select a subset of answer sentences with the minimum redundancy. To achieve this, we apply a greedy algorithm which is originally proposed in the prior extractive summarization study R2N2 [4] for removing redundant sentences. Thus, it perfectly matches our scenario.

Algorithm 1: Greedy Selection for Redundancy Removal

```
   Input: R[]: Ranked sentence list
   Result: summary_list: List of summary sentences
1  embedding_list = [] ;   /* initialize empty embedding list */
2  for sent in R do
3      embed = generate_embedding(sent);
4      embedding_list.append(embed);
5  end
6  redundancy_removed = []; /* initialize empty summary list */
7  redundancy_removed.append(S[0])
8  for embed in embedding_list[1:] do
9      for ground_truth in redundancy_removed do
10         if is_redundant(embed, ground_truth) then
11             continue
12         else
13             redundancy_removed.append(embed)
14         end
15     end
16 end
17 summary_list = select_top_five(redundancy_removed)
18 return summary_list
```

In our case, we first transform all the input (i.e., answer sentences) into sentence embedding by using the sentence embedding model described in Section 3.4.2 (i.e., Lines 2 to 5). Then we pick each sentence for the final summary one-by-one according to the ranking given in the Centrality Estimation module in descending order (i.e., Line 8 to 16). For each selection, if the cosine similarity between the current sentence and others that are already included in the final summary is above a threshold T, the current sentence is regarded as redundant with the final summary and we discard it (i.e., Line 10 to 14). In the end, we pick the top five sentences after greedy selection to form the final summary (i.e., Line 17). If less than five sentences are left after running the greedy selection algorithm, we pick all the remaining sentences as the summary.

3.5 Implementation Details

3.5.1 *Usefulness Ranking Module.*

We implement the BERT model by using a popular Python deep learning library named *Hugging Face*[6]. The model version we select is 'bert-base-uncased'. We tune the same key hyper-parameters (i.e., the learning rate, batch size, and number of epochs) as mentioned in the original BERT paper[15]. We fine tune the model on the training set of ASNQ (around 19 million data instances) and tune the hyper-parameters (i.e., learning rate, batch size, and number of epochs) through the grid searching on the validation dataset of ASNQ.

3.5.2 *Centrality Estimation Module.*

We implement the TextRank algorithm by adapting the python TextRank library.[7]. Unlike the original output of TextRank, which is in the form of a documentation, we extract the intermediate result of TextRank (i.e., the pairs of a sentence and its representativeness score) as the output.

3.5.3 *Redundancy Removal Module.*

We split our in-domain sentence relationship dataset (described in Section 3.4.1) into training and test data with the ratio 9:1. To train our in-domain sentence embedding model, we implement the base model (RoBERTa [23]) by using the same python library *Hugging Face* as Module I. We select the 'roberta-base' verison. We then reuse the RoBERTa model checkpoint provided by SIMCSE[8] that has been fine tuned with general domain dataset and further fine tune it with our training sub-dataset. Meanwhile, we follow the same tuning methodology as tuning the BERT-based model for Usefulness Ranking Module (Section 3.5.1). We found that the performance achieved by tuning with different hyper-parameters does not differ much. Thus we set the learning rate as 5e-5, batch size as 64, and the number of epochs as 3, which are the same as the setting in SIMCSE [12]. We tune the threshold T for redundancy removal by performing a duplicate SO post classification task only using titles, i.e., predict whether a given pair of Stack Overflow title sentences are duplicates or not. This is inspired by prior work [14] that the title of the Stack Overflow posts is the most informative component for expressing the semantics. Specifically, we obtain the representations of both sentences by using our trained sentence embedding model as described in Section 3.4.2. If the cosine similarity of both representations is higher than the threshold T, we consider them duplicates. We empirically investigate the performance with different values of the threshold and obtain the best performing value of 0.8.

4 EXPERIMENT SETTINGS

This section describes the implementation details of all the baselines and the metrics that we used for evaluating the techniques automatically.

4.1 Baselines

By following AnswerBot [41], we position our task as an *extractive query-focused multi-documentation summarization* task. Thus we compare TechSumBot against two groups of baselines: 1) the state-of-the-art answer summarization approach in the SE domain, AnswerBot [41]; and 2) the state-of-the-art extractive query-focused and multi-doc summarization approaches in the NLP domain (i.e., LexRank [9], Biased-TextRank [17], and QuerySum [42]).

AnswerBot. AnswerBot is the state-of-the-art Stack Overflow answer summarization approach. It retrieves relevant questions and then selects useful answer paragraphs. Finally, AnswerBot generates summaries by leveraging MRR algorithm. In relevant question retrieval process of AnswerBot [41], it requires a relevant score,

[6] https://huggingface.com

[7] https://github.com/summanlp/TextRank

[8] https://github.com/princeton-nlp/SimCSE

which represents the extent of the relevance of each answer for the query. In our experiment, we set the relevant scores as 1 because the input answers are selected from the duplicated question posts of the query, which is voted as duplicated by Stack Overflow users as described in Section 2.

LexRank. LexRank [9] is a widely used text summarization approach that is based on sentence-graph. It computes the importance of sentences based on the cosine similarity in the sentence-graph. We implement the LexRank by leveraging python lexrank library.[9]

Biased-TextRank. Biased-TextRank [17] achieves great performance on a query-focused debates summarization dataset in the NLP domain [17]. It works for the query-focused multi-doc summarization task, which matches our task. Biased-TextRank is an unsupervised summarization approach and based on the sentence graph. It also considers the query bias into the sentence-graph and applies an advanced sentence representation approach (i.e., Sentence-BERT [34]) in its pipeline. To adopt Biased-TextRank on TechSumBench, we perform the grid search for its hyperparameters (i.e., damping_factor and similarity_score). We set both damping_factor and similarity_score to 0.7.

QuerySum. To the best of our knowledge, QuerySum [42] is the state-of-the-art summarization approach on query-focused multi-doc summarization task in the NLP domain. It follows coarse-to-fine framework that progressively estimates whether the sentences should be in the summary. We modified the QuerySum [42] pipeline to enable the sentence-level budget of the final summary other than the word number budget used in the original paper.

4.2 Automatic Evaluation Metrics

Following the existing summarization approaches [17, 42], we use ROUGE (**R**ecall-**O**riented **U**nderstudy for **G**isting **E**valuation) [20], a set of evaluation metrics used for evaluationg the automatic summarization approaches, as our automatic evaluation metric. ROUGE is a widely adopted evaluation metric for automatic evaluation for summarization systems [10]. We report ROUGE-N and ROUGE-L. ROUGE-N evaluates the extent of the n-gram overlapping between a new summary and a set of ground-truth summaries [20]:

$$\text{ROUGE-N} = \frac{\sum\limits_{S_i \in S} \sum\limits_{gram_n \in Si} Count_{match}(gram_n)}{\sum\limits_{S_i \in S} \sum\limits_{gram_n \in Si} Count(gram_n)} \tag{6}$$

where S denotes the set of ground-truth summaries, n denotes the n-gram length of the summary, $Count_{match}(gram_n)$ and $gram_n$ denote the maximum n-grams numbers of their coexistence in the ground-truth summaries and a new summary. ROUGE-L measures the overlapping of LCS (i.e., longest common sub-sequence) between the new summary and ground-truth summaries. By following baseline [17], we report ROUGE-1, ROUGE-2, and ROUGE-L. We implement ROUGE evaluation by using the pyrouge python library.[10]

5 EXPERIMENT RESULTS

This section presents our experiment results with corresponding analysis to answer the following three research questions:

RQ1: How effective are the existing approaches in summarizing answers for technical queries on our benchmark?
RQ2: Comparing with existing approaches, how effective is our approach TechSumBot?
RQ3: How much does each module contribute to the performance of TechSumBot?

5.1 RQ1: Effectiveness of Existing Approaches

Experimental setting. To answer this research question, we compare the performance of four existing approaches from both NLP and SE-domain as described in Section 4.1. In particular, to investigate the improvement space of each approach, we also calculate the upper bound of the considered evaluation metrics by assuming the human-annotated summaries (i.e., ground truth) as the output of the ideal approach. For each query in our benchmark, we calculate the ROUGE-N score (we set n = 1, 2) and ROUGE-L between the output produced by an approach and each of the ground truth summaries in our benchmark as described in Section 4.2. Then, we compute the mean of the ROUGE scores.

Table 2: Overall Performance of Existing Approaches

Approach	ROUGE-1	ROUGE-2	ROUGE-L
Upper Bound*	0.784	0.697	0.772
Software Engineering Domain			
AnswerBot	0.490	0.276	0.456
NLP Domain			
LexRank	0.501	**0.289**	0.448
Biased-TextRank	0.428	0.217	0.400
QuerySum	**0.508**	0.284	**0.476**

*: Upper Bound: assuming the human-annotated summaries (i.e., ground truth summary) as the output of the ideal approach.

Result & Analysis. As shown in the Table 2, we find that none of the considered approaches consistently performs better than the others. Specifically, QuerySum performs the best in terms of ROUGE-1 and ROUGE-L while LexRank performs the best in terms of ROUGE-2. More precisely, QuerySum and LexRank achieve comparable performance in terms of ROUGE-1 and ROUGE-2 but QuerySum outperforms LexRank on ROUGE-L by a large margin, i.e., 6%. AnswerBot performs worse than QuerySum and LexRank on all the evaluation metrics consistently but by a small margin. Biased-TextRank performs the worst and its performance differs from the others. Biased-TextRank and LexRank are unsupervised approaches and they are similar as both of them use the PageRank algorithm. There are two key differences between them, (1) whether query information is considered, (2) integrated sentence representation techniques. For the former difference, Biased-TextRank leverages query information (while LexRank does not) which has been demonstrated it is helpful in multiple related tasks, i.e., query-focus summarization. It indicates that the integrated sentence embedding technique (i.e., Sentence-BERT) in Biased-TextRank could be its main bottleneck. Overall, the performance ranking is QuerySum > LexRank > AnswerBot > Biased-TextRank.

In particular, comparing the best performing approach from general domain (i.e., QuerySum) with SE domain approach (i.e., AnswerBot), we find that QuerySum outperforms AnswerBot on all

[9] https://pypi.org/project/lexrank/
[10] https://github.com/bheinzerling/pyrouge

the evaluation metrics but only by a small margin, i.e., 3.6%, 2.8%, 4.3%, in terms of ROUGE-1, ROUGE-2, and ROUGE-L, respectively.

Comparing the existing approaches with the upper bound performance, we find that there is a big room for improvement. The ROUGE-1, ROUGE-2, and ROUGE-L scores of best performing approach QuerySum is 25.29%, 48.36%, 28.21% lower than the upper bound performance, respectively. Particularly, we observe that improvement space in terms of ROUGE-2 is much bigger than that of ROUGE-1 (1.9 times) and ROUGE-L (1.7 times). Considering ROUGE-2 refers to the bi-gram overlapping between generated summary and ground truth summary, it indicates that the existing approaches are unable to capture the information of domain-specific bi-gram terms well enough.

> Overall, QuerySum performs the best on our benchmark and Bias-TextRank performs the worst. Besides, we also find that there is still a big improvement space for the task.

5.2 RQ2: Effectiveness of TechSumBot

Experimental setting. To answer this research question, we firstly follow the same methodology of RQ1 and collect the performance (i.e., in terms of ROUGE-1, ROUGE-2, and ROUGE-L) of our approach on our benchmark. In such a way, we can compare our proposed approach with the best performing baseline approaches from both software engineering (i.e., AnswerBot) and natural language processing (i.e., QuerySum) domains identified in RQ1. Moreover, we conduct a user study on the three approaches to comprehensively understand their effectiveness.

The user study involves five participants, four of them are PhD students and the other one is a postdoctoral fellow. Note that none of participants are involved in the benchmark construction. Considering the expensive human resources, we randomly sample a subset of our TechSumBench with ten technical queries as the experimental data. Half of the queries are Python-related and the other half are Java-related. All the participants have at least three years of development experience in Java or Python. For each question, we collect three answer summaries generated by QuerySum, AnswerBot, and our approach, respectively. By following AnswerBot [41], we then ask all the participants to provide 5-point likert scores to all the summaries separately in terms of usefulness and diversity with respect to the technical query. In other words, participants are required to label each summary from "extremely useless" (denote as 1) to "extremely useful" (denote as 5) and from "extremely redundant" (denote as 1) to "extremely diverse" (denote as 5). For all the approaches, we report the average score labeled by participants across all the 10 queries as their final score. For each of the queries, we anonymize the corresponding approaches of all the summaries and randomly sort them to avoid bias.

Result & Analysis. We report the automatic evaluation result in Table 3. The result shows that TechSumBot consistently outperforms best performing baseline approaches from both SE and NLP domains in terms of all evaluation metrics. Comparing with the Software Engineering domain state-of-the-art (SOTA) baseline, i.e., AnswerBot, TechSumBot achieves better performance in terms of ROUGE-1, ROUGE-2, and ROUGE-L by 14.90%, 36.59%, and 17.54%, respectively. Comparing with QuerySum, the SOTA approach in NLP community, TechSumBot outperforms it by 10.83%, 32.75%, and 12.61%, in terms of ROUGE-1, ROUGE-2, and ROUGE-L, respectively. In summary, TechSumBot outperforms the existing approaches by a large margin.

Table 3: Performance Comparison Between TechSumBot and Selected Approaches

System	ROUGE-1 (%)	ROUGE-2 (%)	ROUGE-L (%)
Software Engineering Domain			
AnswerBot	0.490 (14.90)	0.276 (36.59)	0.456 (17.54)
NLP Domain			
QuerySum	0.508 (10.83)	0.284 (32.75)	0.476 (12.61)
Our Approach			
TechSumBot	**0.563**	**0.377**	**0.536**

%: relative improvement of TechSumBot, i.e., (TechSumBot-baseline)/baseline.

Table 4: User Study Result

System	Usefulness	Diversity
AnswerBot	3.68	3.62
QuerySum	3.80	3.64
TechSumBot	**4.02**	**4.26**

The result of the user study is shown in Table 4. We report the mean of usefulness and diversity score of our approach and the two baselines. We observe that our approach achieves the best performance consistently and the improvement is significant ($p<0.05$) in terms of usefulness and diversity. For the usefulness score, the average score of our approach is 4.02, i.e., between "useful" and "extremely useful", while AnswerBot and QuerySum are 3.68 and 3.80, i.e., between "natural" and "useful". In other words, our approach outperforms AnswerBot and QuerySum by 9.23% and 5.79% in terms of average usefulness score. The average diversity score of our approach is 4.26, i.e., "diverse", while AnswerBot and QuerySum are 3.62 and 3.64, i.e., between "neutral" and "diverse". Our approach outperforms AnswerBot and QuerySum by 17.68% and 17.03% in terms of average diversity score. Overall, both automatic evaluation and user study demonstrate that our approach outperforms the existing approaches. Meanwhile, we observe that Answerbot's diversity and usefulness scores in our user study are very close to those in the user study of Answerbot [41]. We also find that the result derived from automatic evaluation based on our benchmark and human evaluation are consistent. Both of them indicate that our approach produces the most promising answer summaries.

> The results of automatic evaluation show that TechSumBot outperforms the state-of-the-art summarization baselines by 10.83%–14.90%, 32.75%–36.59%, and 12.61%–17.54% in terms of ROUGE-1, ROUGE-2, and ROUGE-L respectively. The results of user study also show our approach outperforms the state-of-the-art summarization baselines by 5.79%–9.23%, 17.03%–17.68% in terms of average usefulness and diversity score.

Table 5: Ablation Study of Each Module

System	ROUGE-1	ROUGE-2	ROUGE-L
TechSumBot			
Module I	0.507	0.298	0.478
Module I & II	0.543	0.340	0.512
Module I & II & III	**0.563**	**0.377**	**0.536**

5.3 RQ3: Ablation Study

Experimental setting. We conduct an ablation study on the performance of each module following the same setting for evaluating QuerySum in [42]. Since each module outputs a rank list of answer sentences, we iteratively integrate them one by one and collect the top-5 ranked sentences as the final summary. Then we follow the same setting of evaluating the end-to-end approach and calculate the evaluation metrics, i.e., ROUGE-1, ROUGE-2, and ROUGE-L.

Result & Analysis. We report the result in Table 5. We observe that each module contributes its own part to the system performance. By only applying *Module I*, we observe that it can already consistently outperforms AnswerBot in terms of all evaluation metrics by a small margin. Besides, *Module I* can achieve comparable result with QuerySum. The result provides another evidence on the effectiveness of pre-trained model. Comparing with *Module I*, integrating *Module II* on the top of *Module I* can further boost the performance by 7.10%, 17.11%, and 7.11% in terms of ROUGE-1, ROUGE-2, and ROUGE-L, respectively. Besides, we find that it can already achieve better performance than all the existing approaches in terms of all the evaluation metrics consistently. By combining all the modules, i.e., the complete version of our approach, it produces the best performance. Specifically, it improves the performance over applying the first two modules by 3.68%, 8.02%, and 4.69% in terms of ROUGE-1, ROUGE-2, and ROUGE-L respectively.

> Every module of TechSumBot contributes its own part to the effectiveness of the whole solution. Combining the first two modules can already produce a better performance than the existing approaches.

6 DISCUSSION

6.1 Qualitative Analysis

We conduct a qualitative analysis by comparing the summaries generated by each baseline with the ground-truth summaries in TechSumBench. More precisely, we analyze the sentences in ground-truth summaries that are incorrectly discarded by baseline approaches and sentences that appear in the summaries generated by baselines but excluded in the ground-truth summaries. We identified multiple weaknesses of existing approaches.

First, we observe that the semantic patterns used in AnswerBot sometimes lead to incorrect results. For example, given a query in TechSumBench: “*What is the difference between a primary key and a unique constraint*”, there is one relevant answer sentence[11] that annotators do not consider useful: “*I suggest that if you are the kind of coder, ..., then PRIMARY KEY is just the thing for you*”. TechSumBot successfully determines that this sentence is not useful to the query. However, a heuristic semantic pattern in AnswerBot regards sentences that start with “*I suggest that ...*” as useful, which leads AnswerBot to incorrectly identify the sentence to be useful. As curating high-quality patterns for extractive summarization task relies on the deep understanding of the dataset characteristics [48], pattern-based summarization approaches on Stack Overflow may requires in-depth exploration of Stack Overflow data.

Second, we observe that existing approaches are weak at distinguishing the redundant sentences between answer sentences. For example, annotators consider both sentences “2**2 = 2 power-of 2.” and “The ** operator in Python is really "power;" that is, 2**3 = 8.” useful to the query, but redundant to each other since both sentences explain ‘**’ operator in Python. To avoid the redundancy in the summary, only one of the two sentences should be retained in the final summary. However, QuerySum and AnswerBot put both sentences into their final summaries. The root cause of such a failure is that the two queries are semantically close but lexically different. When removing redundant sentences, both QuerySum and AnswerBot represent sentences by bag-of-words model and word embedding, which cannot capture the semantic information.

6.2 How Automatic Evaluation Complements User Evaluation

Though user evaluation has advantages [3], researchers have perceived some barriers when performing user evaluations [3, 10, 18]. Especially, recruiting issues and time commitment are the two most common barriers[3]. Fabbri et al. [10] also report that existing summarization approaches apply user evaluations that are largely inconsistent with previous works. Furthermore, they observe that the annotation judges of evaluating summarization approaches are different among annotators with dissimilar knowledge backgrounds (e.g., experts and crowd-sourced workers) [10].

In our case, for query-focused Stack Overflow answer summarization, an automatic evaluation approach by using our benchmark TechSumBench can mitigate the aforementioned barriers. By offering a ground-truth summarization benchmark, automatically evaluating a summarization approach no longer requires recruiting annotators and long waiting periods. Besides, the automatic evaluation result is reusable for the subsequent works by applying the same evaluation metric, e.g., ROUGE [20] on the same benchmark.

We acknowledge that the current version of our benchmark may still contain subjective bias due to the limited size of our benchmark and the annotators’ background and judgment preferences, which are also the biases that exist when conducting a user study. We mitigate such a bias by offering multiple ground-truth summaries that are produced by different annotators independently for the same query. The automatic evaluation’s result is calculated by using the averaged values of the considered evaluation metrics. The biggest advantage of our benchmark is that it’s highly extendable as compared with human evaluation. We provide the replication package as well as the guideline so that the benchmark can be further expanded by community effort. For example, researchers can independently provide more summaries for each query or expand the pool of the queries. With such contributions from the community, the subjective bias can be continuously mitigated.

[11] https://stackoverflow.com/a/9391610

6.3 Threats to Validity

Threats to internal validity are related to the implementation errors of TechSumBot and baselines. To mitigate the first threat, we have double checked our code. For the implementation of baseline approaches, the threat is limited as we reuse their published replication packages. The experiment bias of the annotators may also affect the internal validity. To minimize the threat, we conduct an iterative guideline refinement process. In addition, all the annotators have Java or Python development experience for at least three years.
Threats to external validity are related to the generalizability of our benchmark and experiment results. To ensure the quality of the benchmark, we follow the standard process to produce summaries and updated the guidelines iteratively. The programming language considered in our experiments is also a threat to external validity. Different from prior work [41] that only considers Stack Overflow posts tagged as Java, our work mitigates this threat by considering two popular programming languages, i.e., Java and Python.
Threats to construct validity are related to the used evaluation metrics. ROUGE is widely used as an automatic evaluation metric for summarization approaches in both NLP and SE domains [17, 42, 44]. We use usefulness and diversity in our human evaluation, which are widely used to evaluate the SE domain summarization tasks [8, 25, 41]. Thus, we believe the threat is minimal.

7 RELATED WORK

Summarization of software artifacts has gained much research interest. There are multiple previous works on summarizing different contents in Stack Overflow. Xu et al. [41] proposed AnswerBot which is a Stack Overflow answer summary generator with respect to a specific query. Besides, Opiner [38] aims to help developers efficiently understand API by summarizing API reviews. Opiner adopts available summarization algorithms, such as Textrank [27] (extractive) and Opinosis [11] (abstractive), to produce API reviews from Stack Overflow posts. Furthermore, Nadi and Treude [28] extracted the essential sentences from Stack Overflow to navigate the developers reading answers. Naghshzan et al. [29] proposed an approach based on TextRank algorithm to summarize Android API methods discussed in Stack Overflow. Their approach is based on extractive summarization that extracts the most important sentences from the Stack Overflow posts. The differences between prior works and ours are three-fold. First, different from the aforementioned works that extract useful Stack Overflow sentences by using handcrafted features, we achieve the goal by proposing a transfer learning-based approach. We train the pre-trained models to learn pre-knowledge from a large-scale QA dataset. Second, previous approaches remove redundant sentences in a simple way which carries certain limitations. For example, to calculate sentence similarity, AnswerBot represents sentences into vectors through word embedding and word IDF metrics, which cannot capture the semantic-related sequential information and is poor at distinguishing sentences that are semantically different but lexically similar [40]. Differently, TechSumBot combines both transformer-based sentence representation model and contrastive learning training approach to address the limitations. Third, unlike aforementioned works that perform manual evaluation like user studies, we construct the first benchmark for SO query-focused multi-answer summarization task and enable automatic evaluation, which carries unique advantages as compared with human evaluation.

Apart from automatic summarization for Stack Overflow, NLP community researchers also work on summarization tasks, e.g., query-focused summarization and multi-documentation summarization. Xu et al. [42] presents that the obstacles to multi-doc summarization over the setting of single-doc are (1) difficulty in obtaining training data, (2) large size and number of input resources, and (3) redundancy among input resources. Traditional summarization approaches (e.g., LexRank [9]) rely on the sentence graph and apply PageRank algorithm to rank the sentences. Biased-TextRank [17] is query-focused. It considers embedding the query bias into the graph and leveraging Sentence-BERT [46] to represent the sentences. QuerySum, to our best knowledge, is the state-of-the-art query-focused and multi-doc summarization approach in NLP field. It follows the 'coarse-to-fine' principle to select and filter sentences. QuerySum also considers the relationship between queries and sentences as a useful sentence selection task. Different from the above mentioned works, our approach's framework is specifically designed for Stack Overflow answer summarization. It takes into account the question-answer data structure as well as the widespread content redundancy in software Q&A sites [41]. Besides, our approach leverages the characteristic of Stack Overflow data. For instance, we observed that Stack Overflow users actively label the duplication relationship between posts. Hence, we leverage the information to train the in-domain sentence representation model.

8 CONCLUSION AND FUTURE WORK

In this paper, we focus on the problem of Stack Overflow answer summarization for technical queries. Considering the limitations of only conducting human evaluation without automatic evaluation in the past studies, we find that there is a need of a benchmark to enable automatic evaluation as a complementary evaluation method. Thus, we manually construct the first Stack Overflow multi-answer summarization benchmark TechSumBench, which consists of 111 query-answer summary pairs from 382 Stack Overflow answers with 2,014 sentence candidates. Based on TechSumBench, we evaluate the performance of existing and potential applicable approaches from both SE and NLP fields. The results indicate the approach from the NLP field (i.e., QuerySum [42]) achieves the best performance and there is still a big room for improvement. Motivated by this, we further propose a new approach TechSumBot to tackle the problem. We perform both automatic and human evaluations to evaluate TechSumBot and the results show that TechSumBot outperforms all SOTA baselines from both SE and NLP fields by a large margin. In the future, we plan to integrate our approach TechSumBot into an IDE to help developers in searching their needed information from SQA sites more efficiently and accurately.

ACKNOWLEDGMENTS

This research / project is supported by the Ministry of Education, Singapore, under its Academic Research Fund Tier 2 (Award No.: MOE2019-T2-1-193). Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not reflect the views of the Ministry of Education, Singapore.

REFERENCES

[1] Stefanos Angelidis, Reinald Kim Amplayo, Yoshihiko Suhara, Xiaolan Wang, and Mirella Lapata. 2021. Extractive opinion summarization in quantized transformer spaces. *Transactions of the Association for Computational Linguistics* 9 (2021), 277–293.

[2] Stefanos Angelidis and Maria Lapata. 2018. Summarizing Opinions: Aspect Extraction Meets Sentiment Prediction and They Are Both Weakly Supervised. In *2018 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 3675–3686.

[3] Raymond PL Buse, Caitlin Sadowski, and Westley Weimer. 2011. Benefits and barriers of user evaluation in software engineering research. In *Proceedings of the 2011 ACM international conference on Object oriented programming systems languages and applications*. 643–656.

[4] Ziqiang Cao, Furu Wei, Li Dong, Sujian Li, and Ming Zhou. 2015. Ranking with recursive neural networks and its application to multi-document summarization. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 29.

[5] Joseph Chee Chang, Saleema Amershi, and Ece Kamar. 2017. Revolt: Collaborative crowdsourcing for labeling machine learning datasets. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*. 2334–2346.

[6] Jacob Cohen. 1960. A coefficient of agreement for nominal scales. *Educational and psychological measurement* 20, 1 (1960), 37–46.

[7] Hoa Trang Dang. 2005. Overview of DUC 2005. In *Proceedings of the document understanding conference*, Vol. 2005. 1–12.

[8] Andrea Di Sorbo, Sebastiano Panichella, Carol V. Alexandru, Junji Shimagaki, Corrado A. Visaggio, Gerardo Canfora, and Harald C. Gall. 2016. What Would Users Change in My App? Summarizing App Reviews for Recommending Software Changes. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE 2016)*. 499–510.

[9] Günes Erkan and Dragomir R Radev. 2004. Lexrank: Graph-based lexical centrality as salience in text summarization. *Journal of artificial intelligence research* 22 (2004), 457–479.

[10] Alexander R Fabbri, Wojciech Kryściński, Bryan McCann, Caiming Xiong, Richard Socher, and Dragomir Radev. 2021. Summeval: Re-evaluating summarization evaluation. *Transactions of the Association for Computational Linguistics* 9 (2021), 391–409.

[11] Kavita Ganesan, ChengXiang Zhai, and Jiawei Han. 2010. Opinosis: A graph based approach to abstractive summarization of highly redundant opinions. (2010).

[12] Tianyu Gao, Xingcheng Yao, and Danqi Chen. 2021. SimCSE: Simple Contrastive Learning of Sentence Embeddings. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*. 6894–6910.

[13] Siddhant Garg, Thuy Vu, and Alessandro Moschitti. 2020. Tanda: Transfer and adapt pre-trained transformer models for answer sentence selection. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 34. 7780–7788.

[14] Junda He, Bowen Xu, Zhou Yang, DongGyun Han, Chengran Yang, and David Lo. 2022. PTM4Tag: Sharpening Tag Recommendation of Stack Overflow Posts with Pre-trained Models. (2022).

[15] Devlin Jacob, Chang Ming-Wei, Lee Kenton, and Toutanova Kristina. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of NAACL-HLT*. 4171–4186.

[16] Mitchell Joblin, Sven Apel, Claus Hunsen, and Wolfgang Mauerer. 2017. Classifying developers into core and peripheral: An empirical study on count and network metrics. In *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*. IEEE, 164–174.

[17] Ashkan Kazemi, Verónica Pérez-Rosas, and Rada Mihalcea. 2020. Biased TextRank: Unsupervised Graph-Based Content Extraction. In *Proceedings of the 28th International Conference on Computational Linguistics*. 1642–1652.

[18] Xuan-Bach D Le, Lingfeng Bao, David Lo, Xin Xia, Shanping Li, and Corina Pasareanu. 2019. On reliability of patch correctness assessment. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. IEEE, 524–535.

[19] Amanda Lee, Jeffrey C Carver, and Amiangshu Bosu. 2017. Understanding the impressions, motivations, and barriers of one time code contributors to FLOSS projects: a survey. In *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*. IEEE, 187–197.

[20] Chin-Yew Lin. 2004. Rouge: A package for automatic evaluation of summaries. In *Text summarization branches out*. 74–81.

[21] Jiakun Liu, Sebastian Baltes, Christoph Treude, David Lo, Yun Zhang, and Xin Xia. 2021. Characterizing search activities on stack overflow. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 919–931.

[22] Yang Liu and Mirella Lapata. 2019. Text Summarization with Pretrained Encoders. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. 3730–3740.

[23] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692* (2019).

[24] Sonal Mahajan, Negarsadat Abolhassani, and Mukul R Prasad. 2020. Recommending stack overflow posts for fixing runtime exceptions using failure scenario matching. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 1052–1064.

[25] Senthil Mani, Rose Catherine, Vibha Singhal Sinha, and Avinava Dubey. 2012. Ausum: approach for unsupervised bug report summarization. In *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*. 1–11.

[26] Mika V Mäntylä, Bram Adams, Foutse Khomh, Emelie Engström, and Kai Petersen. 2015. On rapid releases and software testing: a case study and a semi-systematic literature review. *Empirical Software Engineering* 20, 5 (2015), 1384–1425.

[27] Rada Mihalcea and Paul Tarau. 2004. Textrank: Bringing order into text. In *Proceedings of the 2004 conference on empirical methods in natural language processing*. 404–411.

[28] Sarah Nadi and Christoph Treude. 2020. Essential sentences for navigating Stack Overflow answers. In *2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 229–239.

[29] AmirHossein Naghshzan, Latifa Guerrouj, and Olga Baysal. 2021. Leveraging Unsupervised Learning to Summarize APIs Discussed in Stack Overflow. In *2021 IEEE 21st International Working Conference on Source Code Analysis and Manipulation (SCAM)*. IEEE, 142–152.

[30] Ramesh Nallapati, Bowen Zhou, Caglar Gulcehre, Bing Xiang, et al. 2016. Abstractive Text Summarization using Sequence-to-sequence RNNs and Beyond. In *Proceedings of The 20th SIGNLL Conference on Computational Natural Language Learning*. 280–290.

[31] Courtney Napoles, Matthew R Gormley, and Benjamin Van Durme. 2012. Annotated gigaword. In *Proceedings of the Joint Workshop on Automatic Knowledge Base Construction and Web-scale Knowledge Extraction (AKBC-WEKEX)*. 95–100.

[32] Daraksha Parveen, Hans-Martin Ramsl, and Michael Strube. 2015. Topical coherence for graph-based extractive summarization. In *Proceedings of the 2015 conference on empirical methods in natural language processing*. 1949–1954.

[33] Dragomir Radev, Simone Teufel, Horacio Saggion, Wai Lam, John Blitzer, Hong Qi, Arda Celebi, Danyu Liu, and Elliott Franco Drabek. 2003. Evaluation challenges in large-scale document summarization. In *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics*. 375–382.

[34] Nils Reimers, Iryna Gurevych, Nils Reimers, Iryna Gurevych, Nandan Thakur, Nils Reimers, Johannes Daxenberger, Iryna Gurevych, Nils Reimers, Iryna Gurevych, et al. 2019. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 671–688.

[35] Victor S Sheng, Foster Provost, and Panagiotis G Ipeirotis. 2008. Get another label? improving data quality and data mining using multiple, noisy labelers. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*. 614–622.

[36] Rodrigo FG Silva, Chanchal K Roy, Mohammad Masudur Rahman, Kevin A Schneider, Klerisson Paixao, and Marcelo de Almeida Maia. 2019. Recommending comprehensive solutions for programming tasks by mining crowd knowledge. In *2019 IEEE/ACM 27th International Conference on Program Comprehension (ICPC)*. IEEE, 358–368.

[37] Christoph Treude and Martin P Robillard. 2016. Augmenting api documentation with insights from stack overflow. In *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*. IEEE, 392–403.

[38] Gias Uddin and Foutse Khomh. 2017. Automatic summarization of API reviews. In *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 159–170.

[39] Shaohua Wang, NhatHai Phan, Yan Wang, and Yong Zhao. 2019. Extracting API tips from developer question and answer websites. In *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*. IEEE, 321–332.

[40] Moshi Wei, Nima Shiri Harzevili, Yuchao Huang, Junjie Wang, and Song Wang. 2022. CLEAR: Contrastive Learning for API Recommendation. In *2022 IEEE/ACM 44th International Conference on Software Engineering (ICSE)*. IEEE, 376–387.

[41] Bowen Xu, Zhenchang Xing, Xin Xia, and David Lo. 2017. AnswerBot: Automated generation of answer summary to developers' technical questions. In *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 706–716.

[42] Yumo Xu and Mirella Lapata. 2020. Coarse-to-fine query focused multi-document summarization. In *Proceedings of the 2020 Conference on empirical methods in natural language processing (EMNLP)*. 3632–3645.

[43] Chengran Yang, Bowen Xu, Junaed Younus Khan, Gias Uddin, Donggyun Han, Zhou Yang, and David Lo. 2022. Aspect-Based API Review Classification: How Far Can Pre-Trained Transformer Model Go?. In *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER). IEEE Computer Society*.

[44] Ting Zhang, Ivana Clairine Irsan, Ferdian Thung, DongGyun Han, David Lo, and Lingxiao Jiang. 2022. Automatic pull request title generation. In *2022 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE.

[45] Ting Zhang, Bowen Xu, Ferdian Thung, Stefanus Agus Haryono, David Lo, and Lingxiao Jiang. 2020. Sentiment analysis for software engineering: How far can pre-trained transformer models go?. In *2020 IEEE International Conference on*

Software Maintenance and Evolution (ICSME). IEEE, 70–80.

[46] Hao Zheng and Mirella Lapata. 2019. Sentence Centrality Revisited for Unsupervised Summarization. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. 6236–6247.

[47] Ming Zhong, Pengfei Liu, Yiran Chen, Danqing Wang, Xipeng Qiu, and Xuan-Jing Huang. 2020. Extractive Summarization as Text Matching. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. 6197–6208.

[48] Ming Zhong, Danqing Wang, Pengfei Liu, Xipeng Qiu, and Xuanjing Huang. 2019. A closer look at data bias in neural extractive summarization models. *arXiv preprint arXiv:1909.13705* (2019).