AnswerBot: An Answer Summary Generation Tool Based on Stack Overflow

Liang Cai, Haoye Wang
Zhejiang University
China

Bowen Xu
Singapore Management University
Singapore

Qiao Huang
Zhejiang University
China

Xin Xia
Monash University
Australia

David Lo
Singapore Management University
Singapore

Zhenchang Xing
Australian National University
Australia

ABSTRACT

Software Q&A sites (like Stack Overflow) play an essential role in developers' day-to-day work for problem-solving. Although search engines (like Google) are widely used to obtain a list of relevant posts for technical problems, we observed that the redundant relevant posts and sheer amount of information barriers developers to digest and identify the useful answers. In this paper, we propose a tool AnswerBot which enables to automatically generate an answer summary for a technical problem. AnswerBot consists of three main stages, (1) relevant question retrieval, (2) useful answer paragraph selection, (3) diverse answer summary generation. We implement it in the form of a search engine website.

To evaluate AnswerBot, we first build a repository includes a large number of Java questions and their corresponding answers from Stack Overflow. Then, we conduct a user study that evaluates the answer summary generated by AnswerBot and two baselines (based on Google and Stack Overflow search engine) for 100 queries. The results show that the answer summaries generated by AnswerBot are more relevant, useful, and diverse. Moreover, we also substantially improved the efficiency of AnswerBot (from 309 to 8 seconds per query).

Demo tool website: http://answerbot.cn.

Demo video: https://youtu.be/EfHp_Cbeg2w.

Replication package: https://github.com/XBWer/AnswerBot.

CCS CONCEPTS

• **Software and its engineering** → *Software development techniques.*

KEYWORDS

Summary Generation, Relevant Question Retrieval, Stack Overflow

ACM Reference Format:

Liang Cai, Haoye Wang, Bowen Xu, Qiao Huang, Xin Xia, David Lo, and Zhenchang Xing. 2019. AnswerBot: An Answer Summary Generation Tool Based on Stack Overflow. In *Proceedings of the 27th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '19), August 26–30, 2019, Tallinn, Estonia.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3338906.3341186

ESEC/FSE '19, August 26–30, 2019, Tallinn, Estonia

ACM ISBN 978-1-4503-5572-8/19/08…$15.00
https://doi.org/10.1145/3338906.3341186

1 INTRODUCTION

Today, interacting with software question and answer (SQA) sites have been an essential part of developers' daily work for problem-solving. Typically, when a developer encounters a technical problem, he or she formulates the problem as a query and feeds to a search engine (e.g., Google) and obtain a list of possible relevant posts that may contain the answers. Next, developer has to digest the information included in the returned posts to identify possible solutions. The whole solution-seeking experience can be painful due to the massive amounts of information especially the rapid growth of the SQA sites nowadays.

In our prior work, we surveyed 72 developers in two IT companies and we found that an answer summary extracted from other answers is desired since manually generating answer summary is not an easy task [13]. Specifically, there are three main challenges as follows: 1) *Information relevance*, i.e., much low-quality and irrelevant information returned by search engines, 2) *Information redundancy*, i.e., the same aspect may be mentioned in many answer posts, 3) *Information diversity*, i.e., each answer in a post may not be sufficient enough even it is the best answer.

To address the three challenges mentioned above, we first formulate the task as a query-focused multi-answer-posts summarization task for a given technical question [13]. Then, we proposed an approach AnswerBot with a three-stage framework: 1) relevant question retrieval, 2) useful answer paragraph selection, 3) diverse answer summary generation. To implement a practical tool, we substantially improve the efficiency of each stage (on average from 309 to 8 seconds per query, i.e. 301 seconds faster) in three ways, (1) create index for specific attributes in the database to improve the speed of information retrieval. (2) reduce the redundant data loading process, i.e., put the frequently used data (e.g., the word embedding model used in two stages) into cache. (3) preprocessing the questions and answers in the repository (e.g., stop words removal, stemming) rather than process them on-the-fly. (4) replace multiple loops structure for relevance calculation with matrix computation.

A key benefit of our tool is that developers only need to formulate queries for their problems and then our tool offers an end-to-end service, i.e., directly returns an answer summary in a few seconds. Moreover, our tool offers the corresponding Stack Overflow URL

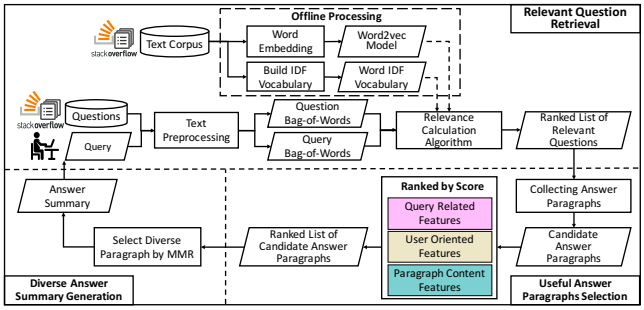

Figure 1: Framework of AnswerBot

of each related answer, hence developers can take a more in-depth look if needed.

2 APPROACH

Figure 1 provides an overview of AnswerBot which generates an answer summary for a given technical problem. AnswerBot consists of three main stages as follow, and we elaborate them in the following sections, 1) *Relevant question retrieval.* Given a technical problem formulated as a query, this stage returns a ranked list of relevant questions, 2) *Useful answer paragraphs selection.* Based on the ranked list of relevant questions, all the answer paragraphs in the answers to these questions are collected. This stage ranks answer paragraphs based on three kinds of features to select relevant and salient answer paragraphs for summarization, 3) *Diverse answer summary generation.* Based on the list of candidate answer paragraphs, this stage applies a text summarization algorithm to generate the answer summary.

2.1 Relevant Question Retrieval

To identify relevant questions for a given query, we first utilize bag-of-words representation to represent the query and the question. We design our approach from two perspectives of capabilities, (1) perceive the importance of words in the software domain, (2) measure the semantic relatedness between two different words. To achieve (1), we introduce a statistical metric named inverse document frequency (IDF) [11] as it tends to filter out common terms (such as "hello", "great"). To achieve (2), we leverage word embedding technique [9] since many works (e.g., [14, 15]) have shown its advantages for measuring text relevance in the presence of lexical gap.

The common step of building an IDF vocabulary and word embedding model is to construct a domain-specific corpus. Specifically, we extract title and description of Java questions (i.e., tagged as *java*) from Stack Overflow to build the corpus. We use Gensim (a Python implementation of the word2vec model [10]) to train the word embedding model based on the corpus.

To build the IDF vocabulary, we further remove the stop words from the corpus based on the list of stop words for text [3].We use a popular stemming tool [7] to reduce each word to its root form for it's typically easier to implement and run faster than lemmatization. Then, we compute the IDF value of each word in the corpus.

Moreover, we propose an algorithm to calculate the relevance between the given query and the question, denoted as W_q and W_Q respectively. The asymmetric correlation $rel(W_q \rightarrow W_Q)$ is calculated as below.

$$rel(W_q \rightarrow W_Q) = \frac{\sum_{w_q \in W_q} rel(w_q, W_Q) * idf(w_q)}{\sum_{w_q \in W_q} idf(w_q)}$$

where $idf(w_q)$ is the IDF metric of the word w_q, $rel(w_q, W_Q)$ is $\max_{w_Q \in W_Q} rel(w_q, w_Q)$, and $rel(w_q, w_Q)$ is the cosine similarity of the two word embeddings w_q and w_Q. And the asymmetric relevance $rel(W_Q \rightarrow W_q)$ is computed in the same way. Then, the symmetric relevance between the query q and the question Q is the average of the two asymmetric relevance between W_q and W_Q as below. In this way, we can get a list of retrieved top-k similar questions and apply them to the next stage of input.

$$rel(q, Q) = (rel(W_q \rightarrow W_Q) + rel(W_Q \rightarrow W_q))/2$$

2.2 Useful Answer Paragraphs Selection

To make the sentences logically fluent, we use the granularity of the answer paragraphs and select the relevant prominent answer paragraphs to summarize. We propose three types of functions to determine whether the answer paragraphs can be used for summarization: query related functions, paragraph content functions, and user-oriented functions.

Query related features measure the relevance between an answer paragraph and the query.

- *Relevance to query.* We set the relevance between a query and an answer paragraph as the relevance between the query and the question from which the answer paragraph is extracted.
- *Entity overlap.* We identify entity mentions in a query or an answer paragraph by matching words by Stack Overflow tags [2]. Let E_q and E_{ap} be the set of entities mentioned in the query and the answer paragraph, respectively. The entity overlap between the query and the answer paragraph is computed as $|E_q \cap E_{ap}| / |E_q|$ ($|E_q| \neq 0$). If the query does not mention any entities ($|E_q| = 0$), we set entity overlap at 0.

Paragraph content features measure the salience of an answer paragraph's content.

- *Information entropy.* A word with higher IDF metric indicates that the word is less common in the corpus and the answer paragraphs may be more useful. Thus, we sum the IDF metrics of words in a paragraph to represent the paragraph's entropy.
- *Semantic patterns.* We summarize 12 sentence patterns (see Figure 2) based on our empirical observations of 300 randomly selected best answers on Stack Overflow. If an answer paragraph contains at least one of the sentence patterns, we set the paragraph's semantic pattern value at 1, otherwise 0.
- *Format patterns.* At last, We observe that HTML tags are often used to emphasize salient information in the discussions. If an answer paragraph contains HTML tags such as $\langle strong \rangle$ and $\langle strike \rangle$, we set its format pattern score at 1, otherwise 0.

No.	Pattern	No.	Pattern
1	Please check XX	7	In short, XX
2	Pls check XX	8	The most important is XX
3	You should XX	9	I'd recommend XX
4	You can try XX	10	In summary, XX
5	You could try XX	11	Keep in mind that XX
6	Check out XX	12	I suggest that XX

Figure 2: Semantic Patterns For Salient Information

User oriented features select summary and high-quality answer paragraphs based on user behavior patterns.

- *Paragraph position.* We observed that when authors write answer posts, they usually start with some summary information and then go into detail. Thus, we set the summary value of the paragraph to be inversely proportional to the position of the paragraph in the post of the first m paragraph, i.e., $summary = 1/pos$ $(1 \leq pos \leq m)$ $(m = 3$ in our current implementation). The summary values of the subsequent (exceeding the m^{th}) paragraphs are set at 0.
- *Vote on answer.* Answers with higher vote indicate that the community believes that they contain high-quality information to answer the corresponding question, so we set an answer paragraph's vote as the vote on the answer post from which the paragraph is extracted.

Based on the above seven features, AnswerBot computes an overall score for each answer paragraph by multiplying the normalized value of each feature. Based on the above seven features, AnswerBot computes an overall score for each answer paragraph by multiplying all the feature scores which are normalized to (0,1]. Answer paragraphs are ranked by their overall scores and the top-10 ranked answer paragraphs are selected as candidate answer paragraphs for summarization.

2.3 Diverse Answer Summary Generation

Given a list of candidate answer paragraphs, maximal marginal relevance (MMR) algorithm is used to select a subset of answer paragraphs to maximize novelty and diversity between the selected answer paragraphs [8]. MMR first builds a similarity matrix between each pair of candidate answer paragraphs. The similarity is computed as the symmetric relevance between the two answer paragraphs as described in the Relevant Question Retrieval section. It then iteratively selects 5 candidate answer paragraphs with maximal marginal relevance. The selected answer paragraphs form an answer summary to the user's technical question.

3 IMPLEMENTATION

3.1 Data Collection

The data source of AnswerBot is from Stack Overflow data dump of March 2016 [1]. In this way, we collect all Java questions (i.e., questions tagged with *Java*) and their corresponding answers. These questions have at least one answer. Moreover, to ensure the quality of the question repository, we require that at least one of the answers of the selected questions is the accepted answer or has vote > 0. At the end, we obtain a repository with 228,817 Java questions and collect their corresponding answers. We construct the text corpus (described in Section 2.1) by using the title and body of these Java questions and then build the word embedding model and the word IDF vocabulary.

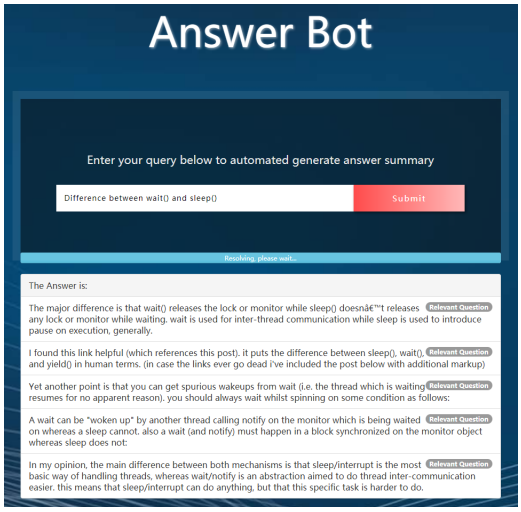

Figure 3: Homepage of AnswerBot

3.2 Tool Implementation

We implement AnswerBot in the form of a search engine website based on Browser/Server architecture. Figure 3 shows the homepage of AnswerBot. When the user enters a technical question as a query, AnswerBot processes it in the background and displays a bar to show the progress of the process. Once the whole process is finished, the answer summary would be showed at the bottom of the page. In details, we use Flask [4], a lightweight web framework, to develop our web application AnswerBot. Then we use uWSGI [5] as a web application server to host, monitor, and manage applications, and apply Nginx [6] as a reverse proxy server to handle high concurrent short connection requests, HTTP caching and provide request forwarding service.

Besides, to meet the efficiency requirement as an online search engine, we optimize the code basis of the previous work [13] in five different ways. First, a common optimization is building the index for the attributes in database. Specifically, we utilize MySQL as our database and create index for the attribute `ID` of each question and the attributes `PostTypeId` and `ParentId` of each answer to speed up the database search process. Second, considering that some data (such as word embedding model) is used in multiple stages, we put it in the memory cache instead of hard disk. Third, before users feed their queries, we preprocess the questions and answers in the repository by removing the stop words and stemming each included word. Moreover, we represent each question and answer as a matrix based on the trained word embedding model. When a new query is fed to AnswerBot, it firstly represents the query into a matrix based on the word embedding model. Then, it leverages matrix computation technique for relevance calculation instead of using multiple loops. In the end, we reduce the average query time from 304 to 8 seconds per query on average.

4 EVALUATION

4.1 User Study

To conduct our user study, we invite 2 postdoctoral fellows and 6 PhD students whose years of working experience on Java are at least 2 years. We randomly select 100 questions (which cover a diversity of aspects of Java programming) and use the titles of these questions as queries. We ensure that our question repository does not contain these 100 query questions and their duplicate questions.

Moreover, we present two baselines based on the Google (add "site:stackoverflow.com" at the end of the query) and the Stack Overflow search engine, respectively. We use the first ranked post returned by search engine as the most relevant question and assume that developers read the accepted answer or the answer with the highest vote if there is no accepted answer of the relevant question. We refer to the collected best or highest-vote answer of the two baseline approaches as their corresponding answer summaries.

For a fair comparison, we provide the answer summaries generated by *Baseline_Google*, *Baseline_SO* and AnswerBot and the participants do not know which answer summary is generated by which approach. Then, they are asked to score the three answer summaries from three aspects, i.e., *relevance*, *usefulness*, and *diversity*. The score is a 5 point likert scale, with 1 being "Highly Irrelevant/Useless/Identical" and 5 being "Highly Relevant/Useful/Diverse".

Table 1: Mean of Relevance, Usefulness and Diversity of Our Approach and the Baseline Approaches

	Relevance	Usefulness	Diversity
Our Approach	**3.450**	**3.720**	**3.830**
Baseline_Google	3.440	3.480*	2.930***
Baseline_SO	2.576***	2.712***	2.305***

***p<0.001, **p<0.01, *p<0.05

Table 1 shows the mean of relevance, usefulness and diversity scores of AnswerBot and the two baselines. The result shows that AnswerBot achieves the best performance in all three aspects, while the *Baseline_SO* achieves the worst performance. Although AnswerBot and *Baseline_Google* have comparable relevance score, AnswerBot has a higher score in terms of usefulness and diversity, especially diversity.

Moreover, we utilize the Wilcoxon signed-rank test [12] to evaluate whether the differences between AnswerBot and the baselines are statistically significant. The improvement of AnswerBot over the *Baseline_SO* is statistically significant on all three aspects at the confidence level of 99.9% while the improvement of AnswerBot over the *Baseline_Google* on usefulness and diversity is statistically significant at the confidence level of 95%. Considering the capability of Google, it is not surprising the difference in relevance is not statistically significant. However, AnswerBot achieves statistically significant better performance in terms of usefulness and diversity.

4.2 Qualitative Analysis

Table 2 shows the answer summary generated for the question *Difference between wait() and sleep()*. The first paragraph provides the most useful answer that *wait()* releases the lock but *sleep()* does not while the other paragraphs describe other aspects of the differences. The second paragraph provides a useful URL that redirects a page

Table 2: Answer Summary for *Difference between wait() and sleep()*

No.	Answer Paragraphs
1	**The major difference is that wait() releases the lock or monitor while sleep() doesn't releases any lock or monitor** while waiting. wait is used for inter-thread communication while sleep is used to introduce pause on execution, generally.
2	I found this link helpful (which references this post). it puts the difference between sleep(), wait(), and yield() in human terms.
3	**Yet another point is that you can get spurious wakeups from wait** (i.e. the thread which is waiting resumes for no apparent reason).
4	**A wait can be "woken up" by another thread calling notify on the monitor which is being waited on whereas a sleep cannot.** also a wait (and notify) must happen in a block synchronized on the monitor object whereas sleep does not.
5	**In my opinion, the main difference between both mechanisms is that sleep/interrupt is the most basic way of handling threads**, whereas wait/notify is an abstraction aimed to do thread inter-communication easier.

summarizes the differences. In this case, AnswerBot provides the original URL of the answer to help users easily find the link. It indicates that the answer summary generated by AnswerBot take both relevance, usefulness, and diversity into consideration.

We also observed that developers could avoid some potential pitfalls when using AnswerBot. For example, the accepted answer for the query *Remove trailing zeros from double* is *If you are willing to switch to* `BigDecimal`*, there is a* `stripTrailingZeroes()` *method that accomplishes this.*. However, this solution would transform a number like 1000.0 into a scientific notation, which cause some potential bugs when directly printing the result. We found that the answer summary generated by AnswerBot provides more solutions, e.g., the first paragraph of the answer summary *You can use string manipulation to remove trailing zeros* provides an additional useful warning that calls `BigDecimal.toPlainString` after stripping the trailing zeros.

5 CONCLUSION

We propose a tool AnswerBot with a three-stage framework for automated generation of answer summary. Our user studies demonstrate the relevance, usefulness, and diversity of the answer summaries generated by AnswerBot. The results indicate that AnswerBot can potentially help software developers improve their efficiency for problem-solving. In the future, we would like to enhance AnswerBot to support more programming languages (e.g. C/C++, Python) and provide more customized search options. Moreover, we are interested in integrating AnswerBot into an intelligent Q&A system to help developers solve their problems efficiently.

ACKNOWLEDGMENTS

This research was partially supported by the National Key Research and Development Program of China (2018YFB1003904), NSFC Program (No. 61602403), Project of Science and Technology Research and Development Program of China Railway Corporation (P2018X002), and the Fundamental Research Funds for the Central Universities.

REFERENCES

[1] 2016. Official data dump of Stack Overflow. https://archive.org/download/stackexchange.

[2] 2016. Official tags of Stack Overflow. https://stackoverflow.com/tags/.

[3] 2018. Common stopwords list of SnowBall Stemmer. http://snowball.tartarus.org/algorithms/english/stop.txt.

[4] 2019. Official document of Flask. http://flask.pocoo.org.

[5] 2019. Official document of uWSGI. https://uwsgi-docs.readthedocs.io/en/latest/#.

[6] 2019. Official website of Nginx. https://www.nginx.com/.

[7] Steven Bird. 2006. NLTK: the natural language toolkit. In *Proceedings of the COLING/ACL on Interactive presentation sessions*. Association for Computational Linguistics, 69–72.

[8] Jaime Carbonell and Jade Goldstein. 1998. The use of MMR, diversity-based reranking for reordering documents and producing summaries. In *Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval*. ACM, 335–336.

[9] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*. 3111–3119.

[10] Radim Řehůřek and Petr Sojka. 2010. Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*. ELRA, Valletta, Malta, 45–50. http://is.muni.cz/publication/884893/en.

[11] Karen Sparck Jones. 1972. A statistical interpretation of term specificity and its application in retrieval. *Journal of documentation* 28, 1 (1972), 11–21.

[12] Frank Wilcoxon. 1945. Individual comparisons by ranking methods. *Biometrics bulletin* 1, 6 (1945), 80–83.

[13] Bowen Xu, Zhenchang Xing, Xin Xia, and David Lo. 2017. AnswerBot: automated generation of answer summary to developersź technical questions. In *Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering*. IEEE Press, 706–716.

[14] Bowen Xu, Deheng Ye, Zhenchang Xing, Xin Xia, Guibin Chen, and Shanping Li. 2016. Predicting semantically linkable knowledge in developer online forums via convolutional neural network. In *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering*. ACM, 51–62.

[15] Xin Ye, Hui Shen, Xiao Ma, Razvan Bunescu, and Chang Liu. 2016. From word embeddings to document similarities for improved information retrieval in software engineering. In *Proceedings of the 38th International Conference on Software Engineering*. ACM, 404–415.