AnswerBot: Automated Generation of Answer Summary to Developers' Technical Questions

Bowen Xu*§, Zhenchang Xing†, Xin Xia‡*✓, David Lo§
*College of Computer Science and Technology, Zhejiang University, China
†School of Engineering and Computer Science, Australian National University, Australia
‡Department of Computer Science, University of British Columbia, Canada
§School of Information Systems, Singapore Management University, Singapore
bowenxu.2017@phdis.smu.edu.sg, Zhenchang.Xing@anu.edu.au, xxia02@cs.ubc.ca, davidlo@smu.edu.sg

***Abstract*—The prevalence of questions and answers on domain-specific Q&A sites like Stack Overflow constitutes a core knowledge asset for software engineering domain. Although search engines can return a list of questions relevant to a user query of some technical question, the abundance of relevant posts and the sheer amount of information in them makes it difficult for developers to digest them and find the most needed answers to their questions. In this work, we aim to help developers who want to quickly capture the key points of several answer posts relevant to a technical question before they read the details of the posts. We formulate our task as a query-focused multi-answer-posts summarization task for a given technical question. Our proposed approach *AnswerBot* contains three main steps : 1) relevant question retrieval, 2) useful answer paragraph selection, 3) diverse answer summary generation. To evaluate our approach, we build a repository of 228,817 Java questions and their corresponding answers from Stack Overflow. We conduct user studies with 100 randomly selected Java questions (not in the question repository) to evaluate the quality of the answer summaries generated by our approach, and the effectiveness of its relevant question retrieval and answer paragraph selection components. The user study results demonstrate that answer summaries generated by our approach are relevant, useful and diverse; moreover, the two components are able to effectively retrieve relevant questions and select salient answer paragraphs for summarization.**

***Index Terms*—summary generation, question retrieval**

I. Introduction

Answers on Stack Overflow have become an important body of knowledge for solving developers' technical questions. Typically, developers formulate their questions as a query to some search engine, and the search engine returns a list of relevant posts that may contain answers. Then, developers need to read the returned posts and digest the information in them to find the answers to their questions. Information seeking is rendered difficult by the sheer amount of questions and answers available on the Q&A site.

We survey 72 developers in two IT companies (i.e., Hengtian and Insigma Global Service) with two questions: *(1) whether you need a technique to provide direct answers when you post a question/query online and why?* and (2) *what is your expectation of the automatically generated answers, e.g., must the answers be accurate?* All the developers agree that they need some automated technique to provide direct answers to a question/query posted online. The reasons they give include (1) sometimes it is hard to describe the problem they meet, so some hints would be useful, (2) there is too much noisy and redundant information online, (3) the answers in long posts are hard to find, and (4) even the answer they found may cover only one aspect of the problem. Developers expect the answer generation tool to provide a succinct and diverse summary of potential answers, which can help them understand the problem and refine the queries/questions.

Our survey reveals a great need to provide improved techniques for information retrieval and exploration. In this work, we aim to develop an automated technique for generating answer summary to developers' technical questions, instead of merely returning answer ports containing answers. Many developers' technical questions are non-factoid questions [1], for example, *what are differences between HashTable and HashMap?*, *How do I write logs and display them realtime in Java Swing?* For such non-factoid technical questions, multiple sparse and diverse sentences may make up the answer summary together.

We formulate our task as a *query-focused multi-answer-posts summarization* task for a given input question. This task is closely related to question answering task [2], [3], [4], which aims to find information from a huge text base to answer a question. An answer summary from the text base should provide related information with respect to the query question. However, the answer sentences and the query question are highly asymmetric on the information they convey. They may not share lexical units. Instead, they may only be semantically related (see examples in Table I and Table II). The inherent lexical gap between the answer sentences and the questions imposes a major challenge for the non-factoid question answering task.

To tackle this challenge, we develop a three-stage framework to achieve the goal of generating an answer summary for a non-factoid technical question. In the first stage, we retrieve a set of relevant questions based on the question titles' relevance to a query question. The question titles and the query question are "parallel text" whose relevance is easier to determine. However, the question titles and the query question often still have lexical gaps. Inspired by the recent success of

✓Corresponding author.

word embeddings in handling document lexical gaps [5], we learn word embeddings from a large corpus of Stack Overflow posts to encode the question titles and the query question and measure their relevance.

In the second stage, we collect all the answers of the relevant questions retrieved in the first stage. We extract answer paragraphs from the collected answers. We develop a multi-criteria ranking method to select a small subset of relevant and salient answer paragraphs for summarization, based on query-related, user-oriented, and paragraph content features. In the third stage, given a set of answer paragraphs to be summarized, the generated answer summary needs to cover as much diverse information as possible. To generate a diverse summary, we measure novelty and diversity between the selected answer paragraphs by Maximal Marginal Relevance (MMR) [6].

We build a question repository of 228,817 Java questions and their corresponding answers from Stack Overflow. We randomly select another 100 Java questions as query questions and use our approach to generate an answer summary for each query question. Our user studies confirm the relevance, usefulness and diversity of the generated summary, and the effectiveness of our approach's relevant question retrieval and answer paragraphs selection components. Our error analysis identifies four main challenges in generating high-quality answer summary: vague queries, lexical gap between query and question description, missing long code-snippet answers, and erroneous answer paragraph splitting, which reveal the future enhancements of our automated answer generation technique.

The main contributions of this paper are the following:

- We conduct a formative study to assess the necessity of automated question answering techniques to provide answer summary to developers' technical questions.
- We formulate the problem of automated question answering as a query-focused multi-answer-posts summarization task for an input technical question.
- We propose a three-stage framework to solve the task, i.e., 1) relevant question retrieval, 2) answer paragraphs selection, 3) answer summary generation.
- We conduct user studies to evaluate the effectiveness of our approach and its components, and identify several areas for future improvements.

Paper Organization. Section II presents our formative study for automated question answering. Section III describes our approach. Section IV reports our experimental methods and results. Section V analyzes the strengths and improvements of our approach and discusses threats to validity of our experiments. Section VI reviews related work. Section VII concludes our work and presents future plan.

II. Formative Study of Answer Summary

We contacted 120 developers by emails in two IT companies. We received 72 replies which help us understand the developers' difficulties in the current information retrieval (IR) practice and assess the necessity of automated question answering techniques. Some comments we received are listed as follows:

- ☞ *"Google will **return a number of "relevant" links for a query**, and I have to click into these links, and read a number of paragraphs ... It is really time-consuming ... Some links even contain viruses. A tool which generates potential answers can **save my time wasted on reading a lot of irrelevant content**. If the generated answer accurately solves my question, it is good. But I think it would be difficult. Anyway, I believe it is no harm to use an answering tool, at least **I can get some hints to solve my problem**."*
- ☞ *"Sometimes I **cannot accurately describe my questions**, which made it hard to find the answer. I have to browse a number of posts online to learn how to refine my query, and search again. Thus, I expect that the answer generation tool can **help me understand the information space better, so I can refine my query faster without browsing those noisy posts.**"*
- ☞ *"I notice even **the best answers** in Stack Overflow often answer the questions **only in one aspect**. Sometimes I need to know a diversity of aspects to understand the problem better, but they cannot be found in a single best answer. Thus, I expect the tool should **provide a diversity of potential answers**, even if some answers are not accurate. "*
- ☞ *"... Some questions received **too many long answers**, and many of these answers have **redundant content**. I expect the answer generation tool should **return succinct answers which covers many aspects of potential solutions. So I could have a high-level understanding of the question I posted.** "*
- ☞ ***"Even if the accuracy of the tool is only 10%, I will still use it.** In the worst case, I will use your tool first, and then search on Google again to find the solutions."*

We use a real-world example to illustrate the difficulties mentioned in the developers' replies and the desirable properties of automated answer generation tool. Assume a developer was interested in the differences between *HashMap* and *HashTable* in Java. He used Google to search for Stack Overflow posts and Table I lists the top 5 ranked Stack Overflow questions returned by Google. The question titles are very relevant to the developer's information need and he should be able to find the needed information in the answers to these questions. However, information overload can be detrimental to the developer. There are 51 answers which have 6,771 words in total. Reading all these answers may take 30 minutes (based on the average readers' reading speed of 200 words per minute [7]). Even just reading the best answers (i.e., the accepted answers) or top-voted answers may still take some time.

It would be desirable to have a answer summary extracted from the answers to the top 5 ranked questions, as the one shown in Table II. This answer summary helps the developer quickly capture the key points of the answers relevant to his technical question. These points reveal the salient and diverse differences between *HashMap* and *HashTable*. They may help the developer decides which API is more appropriate for his task, or provide *information scents* for guiding the developer performing further search or learning [8].

However, manually generating this answer summary is not an easy task. First, there is much low-quality and irrelevant information [9]. Table III shows two examples (eight answers in total). The first example discusses *HashMap* and *HashTable* in C. The second example discusses how to answer *HashMap* and *HashTable* related interview questions. These answers are valid in a particular questions context, but have nothing to do with the developer's technical question.

TABLE I
THE TOP 5 RANKED STACK OVERFLOW QUESTIONS BY GOOGLE SEARCH ENGINE FOR THE QUERY "DIFFERENCES HASHMAP HASHTABLE JAVA"

No.	Question Id	Title	#Answers	#Words
1	40471	Differences between HashMap and Hashtable?	37	4135
2	8875680	Difference between Hashtable and Collections.synchronizedMap(HashMap)	5	847
3	36313817	What are the differences between hashtable and hashmap? (Not specific to Java)	3	747
4	32274953	Difference between HashMap and HashTable purely in Data Structures	3	565
5	30110252	What are the differences between Hashmap vs Hashtable in theory?	3	477

TABLE II
DESIRABLE ANSWER SUMMARY WITH RELEVANT, SALIENT AND DIVERSE INFORMATION

No.	Answer Id	Content	Aspect
1	764418	HashMap is non synchronized whereas Hashtable is synchronized.	Synchronization/Thread Safety
2	25526024	Hashmap can store one key as null. Hashtable can't store null.	Null Keys/ Null Values
3	22491742	Second important difference between Hashtable and HashMap is performance, since HashMap is not synchronized it perform better than Hashtable.	Performance
4	16018266	HashTable is a legacy class in the jdk that shouldn't be used anymore.	Evolution History
5	20519518	Maps allows you to iterate and retrieve keys, values, and both key-value pairs as well, Where HashTable don't have all this capability.	Iteration

TABLE III
EXAMPLES OF IRRELEVANT AND LOW-QUALITY ANSWER PARAGRAPHS

No.	Type	Answer Id	Example
1	Irrelevant	42003464	The explaination between hashmap and hashtable is quite correct as it also fits to the header of a string hash map implementated in strmap. c where thestringmap is a hashtable for strings satisfying the properties of a key,value-structure. Here it says : /...code.../
3	Low-quality	36325577	The interviewer may have been looking for the insight that. A hash table is a lower-level concept that doesn't imply or necessarily support any distinction or separation of keys and values...even if in some implementations they're always stored side by side in memory, e.g. members of the same structure / std::pair<>...

TABLE IV
REDUNDANT ANSWER PARAGRAPHS

Aspect	Set of Redundant Answers' Id	Example
Synchronization/ Thread Safety	40512, 30108941, 17815037, 34618895, 8876289, 14452144, 40878, 42622789, 28426488, 41454, 8876205, 25526024, 764418, 10372667, 27293997, 25348157, 42315504, 11883473, 39785829, 20519518, 8876192, 22084149, 22491742,	[764418] HashMap is non synchronized whereas Hashtable is synchronized. [40512] Hashtable is synchronized, whereas HashMap isn't. That makes Hashtable slower than Hashmap. [39785829] HashTable is internally synchronized. Whereas HashMap is not internally synchronized.
Null Keys/ Null Values	40878, 39785829, 20519518, 25348157, 7644118, 14452144, 25526024, 30108941, 40548, 17815037, 34618895, 10372667, 28426488, 42622789,	[25526024] Hashmap can store one key as null. Hashtable can't store null. [40878] Hashtable does not allow null keys or values. HashMap allow sone null key and any number of null values. [10372667] HashTable can only contain non-null object as a key or as a value. HashMap can contain one null key and null values.
Performance	13797704, 22491742, 40848, 34618895, 30108941, 28426488, 39785829, 24583680	[40848] For threaded apps, you can often get away with ConcurrentHashMap- depends on your performance requirements. [22491742] Second important difference between Hashtable and HashMap is performance, since HashMap is not synchronized it perform better than Hashtable. [34618895] As HashMap is not synchronized it is faster as compared to Hashtable.
Evolution History	1041798, 30108941, 34618895, 22629569, 39785829, 42315504, 40522, 16018266, 10372667, 14627155,	[16018266] HashTable is a legacy class in the jdk that shouldn't be used anymore. [39785829] HashMap extends AbstractMap class where as HashTable extends Dictionary class which is the legacy class in java. [42315504] Second difference is HashMap extends Map Interface and whether HashSet Dictionary interface.
Iteration	40878, 7344090, 39785829, 42622789, 7644118, 41454, 20519518, 8832544, 10372667, 14452144, 40483, 30108941, 34618895,	[30108941] Iterating the values: Hashmap object values are iterated by using iterator. HashTable is the only class other than vector which uses enumerator to iterate the values of HashTable object. [20519518] Maps allows you to iterate and retrieve keys, values, and both key-value pairs as well, Where HashTable don't have all this capability. [42622789] Iterator in HashMap is fail-fast. Enumerator in Hashtable is not fail-fast.

Another issue is information redundancy. As shown in Table IV, the same aspect may be mentioned in many answer posts. The information redundancy and diversity creates a dilemma for the developer. If he reads every post, he is likely to come across the same information again and again, which is a waste of time. If he skips some posts, he risks missing some important aspects he has not seen. Reading only the best answers can address the information overload issue, but not the information redundancy and diversity. For example, the best answer (http://stackoverflow.com/a/40878) to the 1st ranked question in Table I discusses only three aspects (*Synchronization or Thread Safety*, *Null Keys and Null Values*, and *Iteration*) listed in Table II.

To tackle the above information relevance, redundancy and diversity issues for finding answers to developers' technical questions, we need an effective technique to generate an answer summary with relevant, salient and diverse information from unstructured text of answer posts.

III. PROPOSED APPROACH

As shown in Figure 1, our approach (called ***AnswerBot***) takes as input a software-engineering-related technical question as a query from the user, and produces as output an answer summary for the question. Next, we describe the three components of our approach, i.e., relevant question retrieval,

useful answer paragraphs selection, and diverse answer summary generation.

A. Relevant Question Retrieval

The relevant question retrieval component takes a technical question as an input query q and ranks all questions Q in a large question repository (e.g., questions from Q&A sites like Stack Overflow). The questions that are ranked at the top are more likely to have answers that can answer the input technical question. We combine word embedding technique and traditional IDF metric to measure the relevance between the input query and the questions in the repository. Word embedding has been shown to be robust in measuring text relevance in the presence of lexical gap [10]. IDF metric helps to measure the importance of a word in the corpus.

To train the word embedding model and compute the word IDF metrics, we build a domain-specific text corpus using the question title and body of Stack Overflow questions. Each question is considered as a document in the corpus. As the text is from the website, we follow the text cleaning steps commonly used for preprocessing web content [11]. We preserve textual content but remove HTML tags. We remove long code snippets enclosed in HTML tag $\langle pre \rangle$, but not short code fragments in $\langle code \rangle$ in natural language paragraphs. We use software-specific tokenizer [12] to tokenize the sentence. This tokenizer can preserve the integrity of code-like tokens and the sentence structure. We use Gensim (a Python implementation of the word2vec model [13]) to learn the word embedding model on this domain-specific text corpus. To compute the word IDF metrics, we build a vocabulary from the text corpus by removing stop words based on the list of stop words for English text[1] and using a popular stemming tool [14] to reduce each word to its root form. We then compute the IDF metric of each word in the vocabulary over the text corpus.

Given a query q and the title of a question Q in the repository, our relevance calculation algorithm computes their relevance based on an IDF-weighted word embedding similarity between the query and the question title. We use question title in relevance calculation because query and question title are "parallel text" [15]. The query and the question title are transformed into a bag of words, respectively, following the same text preprocessing steps described above. Let W_q be the bag of words for the query q and W_Q be the bag of words for the title of the question Q. An asymmetric relevance $rel(W_q \rightarrow W_Q)$ is computed as:

$$rel(W_q \rightarrow W_Q) = \frac{\sum_{w_q \in W_q} rel(w_q, W_Q) * idf(w_q)}{\sum_{w_q \in W_q} idf(w_q)} \quad (1)$$

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. Intuitively, the word embedding similarity of a more important word in the query and the words in the question title carries more weight towards the relevance measurement between the query and the question title. An 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, i.e., $rel(q, Q) = (rel(W_q \rightarrow W_Q) + rel(W_Q \rightarrow W_q))/2$.

Based on the symmetric relevance between the query and each question in the repository, the questions in the repository are ranked and the top N ranked questions are returned as the relevant questions for the query.

B. Useful Answer Paragraphs Selection

Given a ranked list of relevant questions, all the answer paragraphs (split by HTML tag $\langle p \rangle$) in the answers to these questions are collected. We decide to use the granularity of answer paragraphs because they are the logical text units that answerers create when writing the posts. To select relevant and salient answer paragraphs for summarization, our approach ranks answer paragraphs based on three kinds of features, i.e., query related features, paragraph content features and user oriented features.

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

- *Relevance to query*. As the query and the answer paragraphs usually have lexical gap between the information they convey, it is hard to directly measure their relevance. In this work, 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.[2] The underlying intuition is that the more relevant the question is to the query, the more likely the answers to the question contain relevant answer paragraphs.
- *Entity overlap*. If an answer paragraph contains software-specific entities mentioned in the query, it is very likely that the paragraph is relevant to the query. For example, all desirable answer paragraphs in Table II contain *HashMap* and/or *HashTable* mentioned in the query. Software-specific entities can be programming languages, libraries/frameworks, APIs, data format, and domain-specific concepts [12]. In this work, we consider tags and tag synonyms on Stack Overflow as entities. We identify entity mentions in a query or answer paragraph by matching words in the query or answer paragraph with tag names and tag synonyms. 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 $|E_q \bigcap E_{ap}| \, / \, |E_q|$ ($|E_q| \neq 0$). 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*. Salient answer paragraphs would contain high-entropy words. A word with higher IDF metric indicates that the word is less common in the corpus (i.e.,

[1] http://snowball.tartarus.org/algorithms/english/stop.txt

[2] The relevance between the query and question is calculated during the relevant question retrieval process – see Section III-A.

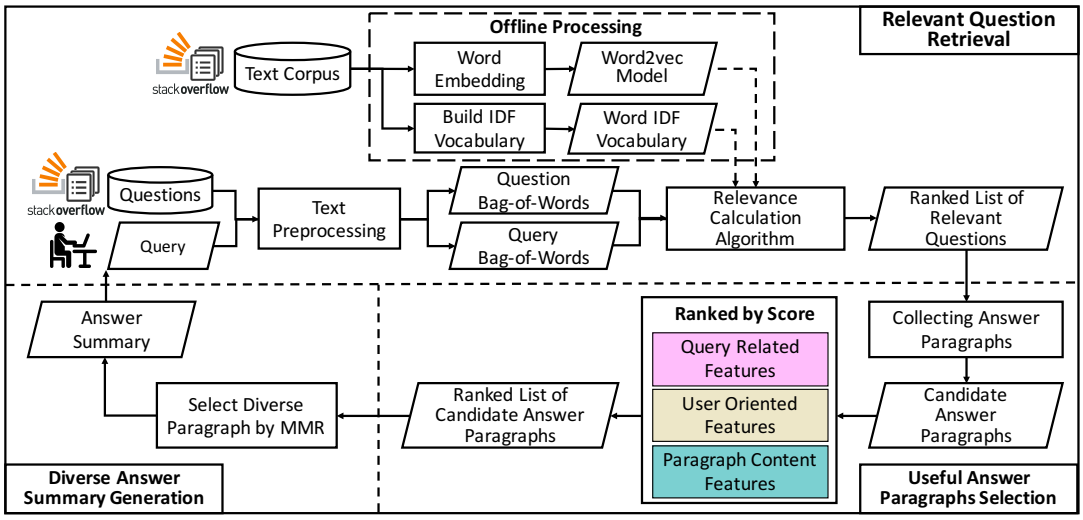

Fig. 1. The Framework of Our Approach ***AnswerBot***

TABLE V
SEMANTIC PATTERNS FOR SALIENT INFORMATION

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

higher entropy). Thus, we sum the IDF metrics of words (after removing stop words and stemming) in a paragraph to represent the paragraph's entropy. Using this feature, many paragraphs with low information entropy, e.g., "*I cannot agree more.*", will be filtered out.

- *Semantic patterns*. We observe that there are certain sentence patterns that often indicate recommendation or summarization of salient information in Stack Overflow discussions (see Table V for examples). For example, a question on Stack Overflow asks "*Array or List in Java. Which is faster?*". The best answer to this question is "***I suggest that** you use a profiler to test which is faster.*". In this work, we summarize 12 sentence patterns 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 pattern value at 1, otherwise 0.
- *Format patterns*. We observe that HTML tags are often used to emphasize salient information in the discussions. For example, $\langle strong \rangle$ highlights some text by bold font and $\langle strike \rangle$ points out some incorrect information. If an answer paragraph contains such HTML tags, we set its format pattern score at 1, otherwise 0.

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

- *Paragraph position*. We observe that when answerers write answer posts, they usually start with some summary information and then go into details. For example, a question asks "*How do I compare strings in Java?*", The first three paragraphs of the best answer of this question present "== *for reference equality*", "*.equals() for value equality*", and "*Objects.equals() checks for nulls*". Therefore, we set a paragraph's summary value to be inversely proportional to the paragraph's position in the post for the first m paragraphs, i.e., $summary = 1/pos$ $(1 \leq pos \leq m)$ $(m = 3$ in our current implementation). The summary values of the subsequent (beyond 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. In this work, 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, an overall score is computed for each answer paragraph by multiplying the normalized value of each feature. To avoid the feature scores being 0, all the feature scores are normalized to (0,1] by adding a smooth factor 0.0001 [16]. Answer paragraphs are ranked by their overall scores and the top M ranked answer paragraphs are selected as candidate answer paragraphs for summarization.

C. Diverse Answer Summary Generation

As shown in Table IV, there are often many redundant answer paragraphs from the answers to relevant questions. The generated answer summary should avoid such redundant information. Given a list of candidate answer paragraphs, maximal marginal relevance (MMR) algorithm is used to select a subset of answer paragraphs in order to maximize novelty and diversity between the selected answer paragraphs [6]. 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 Section III-A. It then iteratively selects K candidate answer paragraphs with maximal marginal relevance. The selected answer paragraphs form an answer summary to the user's technical question.

IV. Experiments & Results

We conduct three user studies to answer the following three research questions, respectively:

RQ1 How effective is our approach in generating answer summaries with relevance, useful and diverse information for developers' technical questions?

RQ2 How effective is our approach's relevant question retrieval component?

RQ3 How effective is our approach's answer paragraph selection component?

In this section, we first describe our repository of questions and answers and tool implementation. We then describe our experimental query questions, and how we select participants and allocate tasks in our user studies. Finally, we elaborate the motivation, approach and results for the three research questions.

A. Question Repository and Tool Implementation

We collect 228,817 Java questions (i.e., questions tagged with Java) and their corresponding answers from Stack Overflow Data Dump of March 2016. These questions have at least one answer. To ensure the quality of question repository, we require that at least one of the answers of the selected questions is the accepted answer or has vote > 0. When collecting questions, we avoid duplicate questions of the already-selected questions, because duplicate questions discuss the same question in different ways and can be answered by the same answer. We use these Java questions and their answers as a repository for answering Java-related technical questions. We build a text corpus using the title and body of these Java questions to train the word embedding model and build the word IDF vocabulary. Considering the conciseness of the generated answer summary and the fact that searchers tend to browse only the top ranked search results [17], our current implementation returns top 5 relevant questions for a query and selects top 10 candidate answer paragraphs for summarization. The generated answer summary contains 5 answer paragraphs.

B. Experimental Queries

We randomly select another 100 questions[3] and use the titles of these questions as query questions. We ensure that our question repository does not contain these 100 query questions and their duplicate questions. The randomly selected 100 query questions cover a diversity of aspects of Java programming. For example, some of them are related to language features, such as multi-threading (e.g., *How does volatile actually work?*) and I/O (e.g., *Can BufferedReader read bytes?*), while others are related to many third-party libraries, such as TEST-Assertions (e.g., *Testing API which returns multiple values with JUnit*) and REST (e.g., *Is there an equivalent to ASP.NET WEB API in JAVA world?*). Some of the query questions are easy to answer (e.g., *How to convert String into DateFormat in java?*), while others are difficult (e.g., *How does volatile actually work?*). The diversity of these 100 questions can improve the generality of our study, and reduce the bias that our approach might be only effective for a specific type of questions. We index these 100 questions as Q1 to Q100.

[3]See our replication package at http://bit.ly/2qBEUhi

TABLE VI
Task Allocation to Participants

RQs	Group 1 (P1, D1, D2, D3)	Group 2 (P2, D4, D5, D6)
RQ1	Q1-Q50	Q51-Q100
RQ2	Q51-Q100	Q1-Q50
RQ3	Q51-Q100	Q1-Q50

C. Participant Selection and Task Allocation

We recruited participants through our school's mailing lists and select 2 postdoctoral fellows (P1 and P2) and 6 PhD students (D1 to D6) to join our user study. All the selected participants have industrial experience on Java development, and they have used Java to develop commercial projects in their work before they went to graduate school. The years of their working experience on Java are vary from 2 to 8 years, with an average 4.6 years. The diversity of these participants' working experience on Java can improve the generality of our results. In practice, our tool aims to help all levels of developers, from novice to senior developers. During our user study, no participants report being unable to understand the query questions and answers.

We divided the eight participants into two groups, i.e., P1, D1, D2 and D3 in Group1 and P2, D4, D5 and D6 in Group2. Furthermore, we divided the 100 questions into two tasks, i.e., Q1-Q50 in Task1 and Q51-Q100 in Task2. Table VI present the task allocation to the two participant groups. For RQ1, Group1 worked on Task1 questions, while Group2 worked on Task2 questions. For RQ2 and RQ3, Group1 worked on Task2 questions, while Group2 worked on Task1 questions. All four participants in these two groups were required to review the answers of the assigned 50 questions independently. With this task allocation, we have all 100 query questions evaluated for the three research questions. As RQ1 evaluates the overall performance of our approach, and RQ2 and RQ3 evaluates its components, using the same set of query questions to evaluate RQ1 and RQ2/RQ3 may bias the participants' results. However, since RQ2 and RQ3 evaluates the two components independently and the two components deal with completely different input/output, using the same questions would have little impact on the participants' results. We asked the participants to complete the study in three 2-hour sessions; the first session evaluates RQ1, while the second and third evaluate RQ2 and RQ3 respectively.

D. Research Questions

RQ1: Effectiveness of the overall approach and the relevance, usefulness and diversity of answer summary

Motivation. In the current IR practice, developers retrieve relevant questions by entering their technical questions to a search engine. Then they have to manually browse the answers of the returned relevant questions to find the needed information. In contrast, our approach can automatically generate an answer summary of the key points in the answers of the returned relevant questions. We would like to investigate the

overall performance of our approach in terms of the relevance, usefulness and diversity of the generated summary, compared with manual information seeking.

Approach. We compare our approach against two baselines built based on Google search engine and Stack Overflow search engine, respectively. We add "site:stackoverflow.com" to the query of Google search engine so that it searches only posts on Stack Overflow. For a query question, we use the first ranked Stack Overflow question returned by a search engine as the most relevant question. We assume that a developer would read the best answer (i.e., 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 the *Baseline_Google* answer summary and the *Baseline_SO* answer summary, respectively.

For each query question, we provide the participants the answer summary generated by *Baseline_Google*, *Baseline_SO* and our approach, respectively. The participants do not know which answer summary is generated by which approach. They are asked to score the three answer summaries from three aspects, i.e., *relevance*, *usefulness* and *diversity*. *Relevance* refers to how relevant the generated summary is to the query. *Usefulness* refers to how useful the generated summary is for guiding the developer's further search or learning. *Diversity* refers to whether the generated summary involves diverse aspects of information. The score is a 5 point likert scale, with 1 being "Highly Irrelevant/Useless/Identical" and 5 being "Highly Relevant/Useful/Diverse".

Results. Table VII shows the mean of relevance, usefulness and diversity scores of our approach and the two baselines. The result shows that our approach achieves the best performance in all three aspects, while the *Baseline_SO* achieves the worst performance. Our approach and *Baseline_Google* have comparable relevance score, but our approach has higher score in usefulness and diversity, especially diversity. The average number of paragraphs in *Baseline_Google* answer summaries and *Baseline_SO* answer summaries are 4.18 and 4.09, respectively.

We use Wilcoxon signed-rank test [18] to evaluate whether the differences between our approach and the baseline approaches are statistically significant. The improvement of our approach over the *Baseline_SO* is statistically significant on all three aspects at the confidence level of 99.9%. The improvement of our approach over the *Baseline_Google* on usefulness and diversity is statistically significant at the confidence level of 95%. We use the best answer of the most relevant question returned by Google as the *Baseline_Google*'s answer for the query. Considering the Google's capability, it is not surprising the difference in relevance is not statistically significant. However, our approach achieves statistically significant better performance on usefulness and diversity (especially diversity). This indicates that the best or highest-vote answers may not cover as diverse information as the developers need. Therefore, it is worth reading more answer posts to summarize more complete information. Our approach automates this summarization process for a diversity of answers.

TABLE VII
MEAN OF RELEVANCE, USEFULNESS AND DIVERSITY OF OUR APPROACH AND THE BASELINE APPROACHES (RQ1)

	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

RQ2: The Effectiveness of relevant question retrieval

Motivation. An answer summary is generated from the answers of some questions relevant to a query question. If the retrieved questions are not relevant to the query question, it is unlikely to generate a high-quality answer summary for the query question. Our question retrieval approach combines word embeddings with traditional IDF metrics. We would like to investigate the effectiveness of our method, compared with the traditional TF-IDF based methods and other word- or document-embedding based methods.

Approach. We build three baseline approaches: TF-IDF based IR [19], word-embedding based document retrieval [5], and document-to-vector (Dov2Vec) based document retrieval [20]. TF-IDF is a traditional IR metric that is often used to rank a document's relevance to a user query in software engineering tasks, such as question retrieval [21] and code search [22]. Yang et al. [5] average word embeddings of words in a document to obtain a document vector which can be used to measure document relevance. Dov2Vec learns document embeddings together with the underlying word embeddings using a neural network and is also applied to measure document relevance [20].

For each query question, we collect the top-10 ranked questions retrieved by our question retrieval approach or one of the baseline approaches. We ask the participants to identify the relevant questions in the top-10 ranked questions. The participants do not know which approach generates which list of top-10 ranked questions. We use the following metrics in comparison of different approaches.

Top-K Accuracy: Given a query question *q*, if at least one of the top-k ranked questions is relevant, we consider the retrieval to be successful, and set the value $Success(q, top-k)$ to 1. Otherwise, we consider the retrieval to be unsuccessful, and set the value $Success(q, top-k)$ to 0. Given a set of queries, denoted as *qs*, its top-k accuracy Top@k is computed as: $Top@k(qs) = \sum_{q \in qs} Success(q, top\text{-}k)/|qs|$. The higher the top-k accuracy score is, the better a relevant question retrieval approach ranks the first relevant question. In this paper, we set k = 1, 5 and 10.

Mean Reciprocal Rank: Given a query question, its reciprocal rank is the multiplicative inverse of the rank of the first relevant question in a ranked list of questions. Mean Reciprocal Rank (MRR) is the average of the reciprocal ranks of all queries in a set of queries: $MRR(qs) = \frac{1}{|qs|} \sum_{q \in qs} \frac{1}{Rank(q)}$. *Rank(q)* refers to the position of the first relevant question in the ranked list of questions returned by a relevant question retrieval approach for a query. The higher the MRR is, the

TABLE VIII
TOP@K ACCURACY AND MRR OF OUR APPROACH AND THE BASELINE APPROACHES IN RELEVANT QUESTION RETRIEVAL (RQ2)

	Top@1	Top@5	Top@10	MRR
Our Approach	**0.460**	**0.610**	**0.660**	**0.520**
Doc2Vec	0.180***	0.580	0.650	0.335***
Word Embeddings	0.340*	0.520	0.550*	0.408*
TF-IDF	0.320*	0.490*	0.530**	0.388*

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

TABLE IX
TOP@K ACCURACY AND MRR OF FEATURE ABLATION FOR ANSWER PARAGRAPH SELECTION (RQ3)

	Top@1	Top@5	Top@10	MRR
all features	**0.660**	**0.990**	**1.000**	**0.803**
w/o query related	0.610*	0.970	1.000	0.758**
w/o user oriented	0.500***	0.980	1.000	0.699***
w/o paragraph content	0.570*	1.000	1.000	0.744**

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

higher the first relevant questions is ranked for a set of queries.

Results. Table VIII shows the Top@k and MRR metrics of our approach and the three baseline approaches for retrieving relevant questions for the 100 query questions. We notice that our approach achieves the best performance in all the evaluated metrics, especially for Top@1 and MRR. The *Doc2Vec* baseline achieves comparable performance as our approach on Top@5 and Top@10, but it has the worst performance on Top@1 and MRR. The *word-embedding* baseline performs slightly better than the *TF-IDF* baseline in Top@k and MRR. The differences between the three baseline approaches actually indicate that the traditional TF-IDF based method and the word or document-embedding based methods can complement each other. In fact, our approach that combines word embeddings and IDF metrics achieves the best performance than either TF-IDF or word/document embedding alone.

We apply Wilcoxon signed-rank test to test the statistical significance of the differences between our approach and the baseline approaches. The improvement of our approach over the *TF-IDF* baseline is statistically significant on all the metrics at the confidence level of 95%. Compared with the *word-embedding* and *Doc2Vec* baselines, our approach is statistically significant better on Top@1 and MRR at the confidence level of 95%. Although the differences between our approach and the *word-embeddings* and *Doc2Vec* baselines on Top@5 and Top@10 are not statistically significant, the significantly better MRR indicates that our approach can rank relevant questions higher than the other two baselines.

RQ3: The Effectiveness of answer paragraph selection

Motivation. To select the most relevant and salient answer paragraphs for summarization, our approach considers three types of features of answer paragraphs, i.e., query-related, user-oriented and paragraph content features. We would like to investigate the impact of different types of features on the results of answer paragraphs selection.

Approach. We remove one type of features at a time from the full feature set and reserve the other two types. Thus, three baseline approaches are built, i.e., without (w/o) query-related features, w/o user-oriented features, and w/o paragraph content features. We let each approach output a ranked list of 10 answer paragraphs with the highest overall score. We take the union of the 40 answer paragraphs selected by our approach (with all features) and the three baselines. Participants are asked to judge whether the selected paragraphs contain salient information relevant to the query question. They do not know which answer paragraph is selected by which approach. We use Top@k accuracy and MRR in the top 10 ranked candidate answer paragraphs to evaluate the performance of answer paragraph selection with and without certain type of features.

Results. Table IX presents Top@k and MRR metrics of using all features or adopting certain type of features for answer paragraph selection. Using all features achieves the best performance in all metrics compared with adopting certain type of features. This suggests that all types of features are useful for answer paragraph selection. Using query-related features achieves better performance than adopting the other two types of features, and using user-oriented features achieves the worst performance. This suggests that user-oriented features play the most important role for answer paragraph selection, paragraph content features take a second place, and query-related features are relatively less important.

Wilcoxon signed-rank test shows that the improvement of using all features over adopting certain type of features is statistically significant on Top@1 and MMR at the confidence level of 95%. Top@5 and Top@10 results in Table IX show that there is almost always at least one relevant answer paragraph in the top 5 or 10 ranked list of candidate answer paragraphs. This demonstrates the general effectiveness of our answer paragraph selection features. Therefore, the differences between using all features and adopting certain type of features on Top@5 and Top@10 are not statistically significant.

V. DISCUSSION

In this section, we qualitatively compare our question answering approach with community question answering practice. We then discuss cases where our approach is ineffective, followed by some threats to validity.

A. Comparison with Community Question Answering

In community question answering, developers post their questions on a Q&A site like Stack Overflow and wait for answers from the community of developers. To understand the differences between the community-provided answers to a technical question and the answer summary that our approach generates from answers of relevant questions, we manually compare the best answers of the questions we use as queries and the answer summary that our approach generates for these questions.

Table X presents two examples. The query question in first example is "calculating time difference" in which the developer runs into some errors in using $getTime()$ to calculate time difference. The best answer of this question suggests to use $long$ to store $getTime()$'s return value, rather than casting it to int. For this query question, our generated answer summary consists of five paragraphs from the four answers of two relevant questions. Except for

the fifth paragraph "You can try this" (due to the limitation of paragraph splitting, see Section V-B), the other four paragraphs provide two alternative solutions (using $System.nanoTime()$ or $currentTimeMillis()$) to calculate time difference. They also describe the reason and cautions of using the two APIs, such as $System.nanoTime()$ is more precise, $System.nanoTime()$ must not be used as a wall clock, $currentTimeMillis()$ may not be a good method for time due to method overhead.

The query question in the second example is about hostname mismatch problem in $HTTPClient$. The best answer provides a solution and explains the advantage of the provided solution. Our answer summary consists of five paragraphs from the five answers of four relevant questions. Except for the fifth paragraph from the fourth question (about a fact of Amazon AWS service), the other four paragraphs provide valuable information which can complement the best answer of the query question. The third and fourth paragraphs provide two different solutions for solving the problem, i.e., using $HttpClientBuilder.create().build()$ or using a fixed version of $HTTPClient$ instead of the buggy version. Although the first and second paragraphs do not provide direct solutions, they point out some important aspects related to the hostname mismatch problem, such as avoiding the action which allows all hosts without any verification, checking the DNS name of the certificate presented by the server.

As the above two examples show, community answers to a technical question usually focus on a specific aspect technical issue in the question. Our generated answer summary derived from answers of several relevant questions can well complement community answers to a technical question by providing more alternative solutions and/or broader information useful for further search and learning.

B. Error Analysis

Through the analysis of the cases in which our approach generates a low-quality even unrelated answer summary, we identify four main challenges in generating high-quality answer summary: vague queries, lexical gap between query and question description, answers involving long code snippet, and erroneous answer paragraph splitting.

In our study, we randomly select 100 Stack Overflow questions and use their titles as queries to search the question repository. However, question titles are short and some of them are vague, e.g., "*Why is this code working without volatile?*", "*Fast processing of data*". It is hard to understand such question titles without looking into the question bodies. Furthermore, the lexical gap between query question and titles of questions in the repository makes it difficult to measure their semantic relevance. Due to these two challenges, our relevant question retrieval component fails to retrieve at least one relevant question in the top 10 results for 34 of the 100 query questions. The low-quality question retrieval results directly result in the low quality of the generated answer summary. To improve relevant question retrieval, we will consider question bodies which contain richer information and adopt deep neural network [15] which are more robust for handling lexical gaps in text.

Our current approach keeps short code fragments (enclosed in HTML tag $\langle code \rangle$) in natural language paragraphs but removes long code snippets (enclosed in HTML tag $\langle pre \rangle$). However, for some query questions, such as "*How to send an Image from Web Service in Spring*", "*How to implement a db listener in Java*" and "*XML to Json using Json-lib*", relevant answers are code snippets, rather than natural language paragraphs. To generate high-quality answer summary for this type of questions, we must take into account long code snippets.

Our current approach splits the text in an answer post into answer paragraphs by HTML tag $\langle p \rangle$. This simple strategy may sometimes break the logic relationships between several consecutive physical paragraphs. For example, people often write some introductory paragraph like "*Try with the following:* ", "*From here you can go to*", and "*There are many solutions to this problem*", followed by a separate paragraph explaining the details. To generate more sensible answer summary, an introductory paragraph and the following detailed explanation should be treated as a whole, using more robust paragraph splitting method.

C. Threats to Validity

Threats to internal validity are related to experimental bias of participants in manual examination of relevant questions and answer summaries. First, the participants' lack of knowledge on Java may affect their judgements about question/answer's relevance and usefulness. This threat is limited by selecting only participants who have at least 2 years industrial experience on Java development. Still, there could be errors because an experienced Java developer is not necessarily familiar with all Java frameworks and APIs. Second, the participants' degree of carefulness and effort in manual examination may affect the validity of judgements. We minimize this threat by choosing participants who express interests in our research, and giving the participants enough time to complete the evaluation tasks.
Threats to external validity are related to the generalizability of our research and experiments. Stack Overflow questions are related to many domains other than Java (e.g. Python, Eclipse, database), or combinations of multiple domains. Our approach is general, but considering the background knowledge of available participants for the user studies, we use only Java questions in this work. Furthermore, as user studies require significant human efforts, we only use 100 query questions in this study. In the future, we will use more queries, larger question repository, and questions of more domains to reduce these threats.
Threats to construct validity are related to the suitability of our evaluation metrics. Relevance [23], usefulness [24] and diversity [25] are widely used to evaluate summarization tasks in software engineering, Both relevant question retrieval and answer paragraphs selection are ranking problems. Thus, we use Top@k accuracy (k=1, 5, 10) and MRR. These two metrics are the most widely used metrics for evaluating IR techniques [26], [15], [27].

TABLE X
COMPARISON OF THE BEST COMMUNITY-PROVIDED ANSWERS AND THE ANSWER SUMMARY GENERATED BY OUR APPROACH

#1 Query Question: Calculating time diffrence			
Best Answer of Query Question(Id: 18575058)	Answer Summary Generated by Our Approach	Relevant Questions	
		Id	Title

Try changing
int diff=(int)d2.getTime()-(int)d1.getTime();
to long diff=d2.getTime()-d1.getTime();
Explicit typecasting from long to int
will cause precision loss and may result in
a negative value on subtraction.
long a = 90000000000000L;
long b = 10000001000000L;
a>b (int)a-(int)b=>negative value
	[9707986] Try this
	long start_time = System.nanoTime();
	resp = GeoLocationService.getLocationByIp(ipAddress);
	long end_time = System.nanoTime();
	double difference = (end_time - start_time)/1e6;
		9707938	Calculating time difference in Milliseconds

	[9707972] No, it doesn't mean it's taking 0ms - it shows it's taking a smaller amount of time than you can measure with currentTimeMillis(). That may well be 10ms or 15ms. It's not a good method to call for timing; it's more appropriate for getting the current time.		
	[9707972] To measure how long something takes, consider using System.nanoTime instead. The important point here isn't that the precision is greater, but that the resolution will be greater... but only when used to measure the time between two calls. It must not be used as a "wall clock".		
	[9707982] Since Java 1.5, you can get a more precise time value with System.nanoTime(), which obviously returns nanoseconds instead.		
	[24907491] You can try this :	24907002	Calculating Time Difference in Java

#2 Query Question: Android HttpClient - hostname in certificate didn't match <example.com>!= <*.example.com>			
Best Answer of Query Question(Id: 3136980)	Answer Summary Generated by Our Approach	Relevant Questions	
		Id	Title
This is my (edited) solution: /...code.../ It has the advantage of not changing the default behavior unless there is a wildcard domain, and in that case it revalidates as though the 2 part domain (e.g., someUrl.com) were part of the certificate, otherwise the original exception is rethrown. That means truly invalid certs will still fail.	[15497467] Important - if you are allowing all hosts (that is, disabling host name verification), then it is certainly NOT safe. You shouldn't be doing this in production.	15497372	Java HTTP post using HTTPS Confusion - javax.net.ssl.SSLException: hostname in certificate didn't match
	[7257060] The certificate verification process will always verify the DNS name of the certificate presented by the server, with the hostname of the server in the URL used by the client.	7256955	Java SSLException: hostname in certificate didn't match
	[24526126] This HttpClientBuilder.create().build() will return org.apache.http.impl.client.InternalHttpClient. It can handle the this hostname in certificate didn't match issue.		
	[34494091] This problem is described in Apache HttpClient resolving domain to IP address and not matching certificate. It appears to be a bug in the version of HTTPClient you are using, where it compares the target IP instead of the target hostname with the subject certificate. Please use a fixed version of HTTPClient instead.	34493872	SSLException: hostname in certificate didn't match <50.19.233.255>!=<*.heroku.com>
	[12755039] Buckets whose name contains periods can now be correctly addressed again over HTTPS.	12755038	SSL problems with S3/AWS using the Java API: hostname in certificate didn't match

VI. RELATED WORK

Many text summarization approaches have been applied in different software engineering tasks, aiming to reduce the developers' effort to read an immense quantity of information. Rastkar et al. propose an extractive approach for automatic bug report summarization [28]. Their approach selects a subset of bug report comments by training a binary classifier to determine whether a comment should be selected or not. Andrea et al. propose an approach *SURF* to produce a summary for user reviews [24]. *SURF* classifies each user review sentence to one of the user-intention categories, and groups together sentences covering the same topic. It then uses a sentence selection and scoring mechanism to generate the user-review summary. Different from the above studies, we focus on answer summarization on online Q&A sites which requires a different set of features. Additionally, we formulate our problem as a ranking problem instead of a classification one.

A number of studies have also proposed methods to identify relevant or high-quality posts in question and answering sites. Gottipati et al. propose a semantic search engine framework to process posts in discussion threads to recover relevant answers to a user query [29]. They classify posts in the discussion threads into 7 categories (e.g., questions, answers, clarifying question, junk, etc.) and use the category labels to filter less useful information to improve the search experience. Yao et al. propose an approach to detect high-quality posts in community question answering sites [30]. Different from the above studies, we not only identify relevant posts but also create summaries of these posts.

Popular search engines like Google can provide direct answers for some types of queries. For example, Google can extract a paragraph from the best answer of the Stack Overflow question "*what are the differences between hashtable and hashmap?*" as the answer for a query like "differences between hashtable and hashmap". However, Google does this only for specific kinds of 5W+1H (who, what, where, when, why, how) questions, and it does not generate direct answers to all 5W+1H questions, e.g., "*What are the differences between quick sort and bubble sort?*". More importantly, Google extracts only a paragraph from one answer of one question, while our approach is multi-answer-posts summarization.

VII. CONCLUSION

Our formative study indicates that developers need some automated answer generation tools to extract a succinct and diverse summary of potential answers to their technical questions from the sheer amount of information in Q&A discussions. To meet this need, we propose a three-stage framework for automated generation of answer summary. Our user studies demonstrate the relevance, usefulness and diversity of our automated generated answer summaries.

In the future, we will investigate more robust relevant question retrieval algorithms, and explore more features for answer paragraph selection. We will develop our approach into practical tools (e.g., browser plugins) to provide developers with direct answers and extended suggestions to help them find the needed information more efficiently on the Internet.

Acknowledgment. This work was partially supported by NSFC Program (No. 61602403 and 61572426).

REFERENCES

[1] H. Song, Z. Ren, S. Liang, P. Li, J. Ma, and M. de Rijke, "Summarizing Answers in Non-Factoid Community Question-Answering."

[2] C. Unger, L. Bühmann, J. Lehmann, A.-C. Ngonga Ngomo, D. Gerber, and P. Cimiano, "Template-based question answering over RDF data," in *Proceedings of the 21st international conference on World Wide Web*. ACM, 2012, pp. 639–648.

[3] M. Iyyer, J. L. Boyd-Graber, L. M. B. Claudino, R. Socher, and H. Daumé III, "A Neural Network for Factoid Question Answering over Paragraphs." in *EMNLP*, 2014, pp. 633–644.

[4] M. Asaduzzaman, A. S. Mashiyat, C. K. Roy, and K. A. Schneider, "Answering questions about unanswered questions of stack overflow," in *Mining Software Repositories (MSR), 2013 10th IEEE Working Conference on*. IEEE, 2013, pp. 97–100.

[5] X. Yang, D. Lo, X. Xia, L. Bao, and J. Sun, "Combining Word Embedding with Information Retrieval to Recommend Similar Bug Reports," in *Software Reliability Engineering (ISSRE), 2016 IEEE 27th International Symposium on*. IEEE, 2016, pp. 127–137.

[6] J. Carbonell and J. Goldstein, "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, 1998, pp. 335–336.

[7] M. A. Just and P. A. Carpenter, "Speedreading," 1987.

[8] W.-C. Wu, "How far will you go?: characterizing and predicting online search stopping behavior using information scent and need for cognition," in *Proceedings of the 36th international ACM SIGIR conference on Research and development in information retrieval*. ACM, 2013, pp. 1149–1149.

[9] X. Xia, L. Bao, D. Lo, P. S. Kochhar, A. E. Hassan, and Z. Xing, "What do developers search for on the web?" *Empirical Software Engineering*, pp. 1–37, 2017.

[10] B. Xu, D. Ye, Z. Xing, X. Xia, G. Chen, and S. Li, "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, 2016, pp. 51–62.

[11] C. Chen, Z. Xing, and X. Wang, "Unsupervised Software-Specific Morphological Forms Inference from Informal Discussions."

[12] D. Ye, Z. Xing, C. Y. Foo, Z. Q. Ang, J. Li, and N. Kapre, "Software-specific named entity recognition in software engineering social content," in *Software Analysis, Evolution, and Reengineering (SANER), 2016 IEEE 23rd International Conference on*, vol. 1. IEEE, 2016, pp. 90–101.

[13] R. Řehůřek and P. Sojka, "Software Framework for Topic Modelling with Large Corpora," in *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*. Valletta, Malta: ELRA, May 2010, pp. 45–50, http://is.muni.cz/publication/884893/en.

[14] S. Bird, "Nltk: the natural language toolkit," in *Proceedings of the COLING/ACL on Interactive presentation sessions*. Association for Computational Linguistics, 2006, pp. 69–72.

[15] G. Chen, C. Chen, Z. Xing, and B. Xu, "Learning a dual-language vector space for domain-specific cross-lingual question retrieval," in *Ieee/acm International Conference on Automated Software Engineering*, 2016, pp. 744–755.

[16] S. F. Chen and J. Goodman, "An empirical study of smoothing techniques for language modeling," in *Proceedings of the 34th annual meeting on Association for Computational Linguistics*. Association for Computational Linguistics, 1996, pp. 310–318.

[17] "Models of the Information Seeking Process," http://searchuserinterfaces.com/book/sui_ch3_models_of_information_seeking.html.

[18] F. Wilcoxon, "Individual comparisons by ranking methods," *Biometrics bulletin*, vol. 1, no. 6, pp. 80–83, 1945.

[19] S. Haiduc, G. Bavota, A. Marcus, R. Oliveto, A. De Lucia, and T. Menzies, "Automatic query reformulations for text retrieval in software engineering," in *Software Engineering (ICSE), 2013 35th International Conference on*. IEEE, 2013, pp. 842–851.

[20] Q. Le and T. Mikolov, "Distributed representations of sentences and documents," in *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, 2014, pp. 1188–1196.

[21] X. Cao, G. Cong, B. Cui, and C. S. Jensen, "A generalized framework of exploring category information for question retrieval in community question answer archives," in *Proceedings of the 19th international conference on World wide web*. ACM, 2010, pp. 201–210.

[22] D. Shepherd, K. Damevski, B. Ropski, and T. Fritz, "Sando: an extensible local code search framework," in *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*. ACM, 2012, p. 15.

[23] D. R. Radev and W. Fan, "Automatic summarization of search engine hit lists," in *Proceedings of the ACL-2000 workshop on Recent advances in natural language processing and information retrieval: held in conjunction with the 38th Annual Meeting of the Association for Computational Linguistics-Volume 11*. Association for Computational Linguistics, 2000, pp. 99–109.

[24] A. Di Sorbo, S. Panichella, C. V. Alexandru, J. Shimagaki, C. A. Visaggio, G. Canfora, and H. C. Gall, "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*. ACM, 2016, pp. 499–510.

[25] S. Mani, R. Catherine, V. S. Sinha, and A. Dubey, "Ausum: approach for unsupervised bug report summarization," in *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*. ACM, 2012, p. 11.

[26] B. Xu, Z. Xing, X. Xia, D. Lo, Q. Wang, and S. Li, "Domain-specific cross-language relevant question retrieval," in *Proceedings of the 13th International Conference on Mining Software Repositories*. ACM, 2016, pp. 413–424.

[27] B. Xu, Z. Xing, X. Xia, D. Lo, and X.-B. D. Le, "Xsearch: a domain-specific cross-language relevant question retrieval tool," in *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*. ACM, 2017, pp. 1009–1013.

[28] S. Rastkar, G. C. Murphy, and G. Murray, "Automatic summarization of bug reports," *IEEE Transactions on Software Engineering*, vol. 40, no. 4, pp. 366–380, 2014.

[29] S. Gottipati, D. Lo, and J. Jiang, "Finding relevant answers in software forums," in *Proceedings of the 2011 26th IEEE/ACM International Conference on Automated Software Engineering*. IEEE Computer Society, 2011, pp. 323–332.

[30] Y. Yao, H. Tong, T. Xie, L. Akoglu, F. Xu, and J. Lu, "Detecting high-quality posts in community question answering sites," *Information Sciences*, vol. 302, pp. 70–82, 2015.